

Process & Decision Documentation

Project/Assignment Decisions

Side Quests and A4 (Individual Work)

Keep this section brief, typically 2 to 4 sentences.

Focus on:

- One significant decision or change I made was to make the obstacles have a border around it so that players cannot pass through and would instead bounce off.
 - I did this because originally in the code that was generated, players could directly pass through and it was more for serving decorative purposes, which I thought was kind of useless as we already had the moving background
- I also decided myself on the type of game and vibe that I wanted going for my and how I would implement the assignment prompt into it

Examples:

- Added bounce to obstacles
- Chose game colour palette myself
- Implemented fun easter eggs disguised as start constellations/symbols
- Made the camera moving (like ocean waves) even when player was not

You are not expected to document every alternative or iteration

Role-Based Process Evidence

Entry Header

Name: Katherine Chen

Role(s): Code editor and AI prompter

Primary responsibility for this work: Using AI to fulfill side quest 5's requirements, as well as being the human decision point of how the level should progress.

Goal of Work Session

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

Goal:

- Create a dreamscape like world that expands beyond just what user's current line of sight/camera can see
- Add little easter eggs in the form of dream like symbols that float across a dreamscape

Tools, Resources, or Inputs Used

- GenAI:
 - Claude Sonnett 4.6
 - VSCode Agent
- VSCode
- GitHub

GenAI Documentation

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

Date Used: Mon Feb 23 2026

Tool Disclosure: Claude Sonnett 4.5, VSCode Agent

Purpose of Use: Similar to the previous sidequests, after reading the assignment description I had an idea of the ideal game that I wanted it to be and mental image of what I wanted the game to look like, but did not know the specific commands to get VSCode's AI to do exactly what I wanted to do. That's why I first fed the code to Claude and explained my idea in depth, letting it create better and more accurate prompts for VSCode to interpret. This cuts down on a lot of back and forth unnecessary conversations.

Summary of Interaction: Claude code was in charge of generating the prompts that I would give to VSCode to implement directly into the code. I mainly had some back and forths with VSCode regarding minor fixes to the overall structure, as not all of Claude's prompts hit the mark of what I wanted.

Human Decision Point(s): As mentioned, there is only so much AI can do with code and a vision, I had to be on the judging end of whether or not I liked what AI generated with the prompts given. If not, then I would have back and forth conversations to alter or fix those areas. For example, Claude started out by giving me an original prompt of having a hard blue background, resembling the night sky, but I felt it was a bit too heavy and out of place and lessens the impact of everything else, so I changed it to a pastel dreamlike purple.

Integrity & Verification Note: Since I personally know HTML, JavaScript, and CSS, I was able to personally verify most of which the code wrote and understood that it was mostly

correct. Additionally, as mentioned, I combatted/fixed the areas that I wasn't satisfied with by inputting my own prompts. Also, there were minor tweaks that I did directly to the code myself, as certain values (eg. The bounce when player interacts with an obstacle) were not up to par with my mental image.

Scope of GenAI Use: GenAI did not come up with the idea, I did that myself. Claude did, however, turn my idea into actionable prompts, to which VSCode then implemented directly onto the code itself.

Limitations or Misfires: There were certain parts in which the code misunderstood my intentions. For example, when I wanted a moving camera, I was looking more like a deep ocean, lucid dreaming vibe, but Claude's prompts + VSCode's implementations made it look more like a nightmare earthquake.

Summary of Process (Human + Tool)

- I utilized AI to make my creative game vision into code
- I utilized it to troubleshoot some errors
- I came up with my own ideas on how the code should look (dreamscape lucid dreaming vibe, with calm subtle moving camera) in order to fulfill the sidequest requirements
- I did some of my own tweaking directly to the code when it was too tedious to ask AI (e.g. Change how much "shake" the camera applied and the colour of the text on the top left as it was practically unreadable due to WCAG violations in earlier iterations)

Decision Points & Trade-offs

- The AI isn't always going to give me what I want, for example, when one of the prompts that Claude gave me was to include camera "breathing" moments, however VSCode's AI made it look like an earthquake instead, completely destroying the semblance of "dream" in the dreamscape. I then had to prompt VSCode to make it more "subtle" and natural, which was a decision that I came to after observing how the results didn't line up with expectations.
- Also the colour scheme that the AI chose for some parts I wasn't satisfied with so I made my own palette and manual tweaks
- Also, the moving background in earlier iterations was too harsh so I made it more of a gradient, "free" feel

Verification & Judgement

Explain how you evaluated whether your decision or change was appropriate:

- Playtesting on live server many times before publishing to github. Also playtested on github's links to make sure everything was working as different platforms take into consideration different issues (uppercase variables, ect.)
- WCAG colour contrast comparisons
- Cross referencing with my own mental image

Limitations, Dead Ends, or Open Questions

- I think the main limitation is that AI cannot 100% implement the image I have in mind, no matter how well I explain it. It can come close, but as a designer, there are always things to nitpick on.
 - Eg. Background colour
- Currently I am thinking about what this would look like if there were levels added to it. Currently it's more like open world exploration, but if there was actually a purpose to the game, that would be fun.

Appendix

Please include a full transcript of your conversation with the GenAI. GenAI transcripts are used for transparency and verification only and are not assessed for writing quality or completeness.

Note: If GenAI is used in another language, you must include both the original language and a translated transcript in the appendix

Prompt	Answer
Based off this [example 4 code files], and the following goal: “Create a reflective or meditative camera experience that scrolls through a world larger than the screen. Use pacing and motion to evoke emotion.”, craft me a series of prompts that will create a dreamscape/memory world sort of vibe. I want it to be pastely	<p>Dreamscape / Memory World </p> <p>Great choice — this is a really evocative theme. Here's the vision before we get into prompts:</p> <p>The concept: The player drifts through a fragmented dreamworld — pastel colours, soft floating geometry, a camera that "breathes" (gently pulses its zoom or sways). Hidden memory symbols are scattered across the world; when the player gets near one, it glows and fades like a half-remembered thought. The overall feeling should be quiet, slightly melancholic, and wonder-filled.</p>

with a surrealistic feel, with fragmented floating platforms and hidden symbols that get revealed/reremembered when touched

What We're Building

Before the prompts, here's a plain-English map of the changes across each file so you understand what you're asking the AI to do:

world.json — New world colours (pastels), more obstacles/symbols, a slower camera lerp, and new fields for "memory objects" with positions and icons.

WorldLevel.js — Draw the dreamy background (gradient-like layered rects, floating soft shapes), draw memory symbols that pulse/glow when the player is near, and update the HUD to feel minimal and poetic.

Player.js — Slower, floatier movement with inertia (the player glides to a stop rather than stopping instantly), giving that weightless dream feeling.

sketch.js — Add a subtle camera "breathe" (slow sine-wave sway offset), and a vignette overlay drawn on top of everything.

The Prompts — In Order

Do these **one file at a time**. Open the file in VS Code, highlight all the code, and paste the prompt into Copilot Chat.

1. Start with `world.json`

Prompt: "Rewrite this `world.json` for a dreamscape / memory world theme in a p5.js game. The world should be large (at least 2400x1600). Set the background colour to a deep soft purple (like [28, 18, 52]). Add a `camera` section with a slow `lerp` value around 0.07 and a `breatheSpeed` of 0.4 and `breatheAmount` of 3. Add a `memories` array with at least 8 objects, each with an `x`, `y`, `symbol` (choose from: "△", "○", "◆", "▷", "∞", "◇"), and a `label` (a short poetic 3–5 word phrase like "the day you left" or "warm summer static"). Keep the existing `obstacles` array but replace the placeholder with 6–8 soft floating rectangular shapes scattered across the world."

2. Then `WorldLevel.js`

Prompt: "Rewrite this `WorldLevel.js` for a `p5.js` dreamscape / memory world. Keep the class structure and constructor reading from JSON. Make these changes: (1) `drawBackground()` should fill with the world's bg colour and draw 3 layers of soft translucent ellipses drifting across the background as ambient shapes — use `frameCount` with slow sine waves to gently shift their y position. (2) `drawWorld()` should draw obstacles as soft rounded rectangles with no stroke, in muted lavender/rose/mint pastel colours, cycling through a small palette array. (3) Add a `drawMemories(player)` method that loops through `this.memories` (loaded from JSON), draws each as a glowing pulsing symbol using a sine wave on `frameCount` for opacity, and if the player is within 80px, draws the label text fading in beneath it. (4) Update `drawHUD` to be minimal — just a small italic poetic line in the top left in a light colour, no coordinate debug info."

3. Then `Player.js`

Prompt: "Rewrite this `Player.js` for a dreamscape `p5.js` experience. Instead of instant movement, give the player inertia — use `this.vx` and `this.vy` velocity variables. Each frame, add input direction to velocity scaled by an acceleration of 0.4, then multiply velocity by a friction of 0.88 so the player glides to a stop. Cap speed at `this.s`. Draw the player as a soft glowing white ellipse (radius ~10) with a larger translucent halo ellipse behind it (radius ~22, low opacity). No stroke."

4. Finally `sketch.js`

Prompt: "Rewrite this `sketch.js` to work with the updated `WorldLevel`, `Player`, and `world.json` for a dreamscape experience. Keep the same setup structure. Add these features: (1) Camera breathe — each frame offset the camera by `sin(frameCount * breatheSpeed) * breatheAmount` in both x and y, where those values come from `level.breatheSpeed` and `level.breatheAmount`. (2) After drawing the world and player (inside the push/pop), call `level.drawMemories(player)`. (3) After the pop, draw a vignette overlay: a dark radial gradient rect covering the full canvas using `drawingContext` and a `canvas radial`

	<p><i>gradient from transparent centre to <code>rgba(10, 5, 20, 0.7)</code> at the edges. (4) Keep the R key reset. Remove the HUD coordinate debug line since the new HUD is poetic."</i></p>
	<h2>Tips for Using Copilot</h2> <ul style="list-style-type: none"> • Always paste the full existing file into Copilot Chat alongside the prompt — say "<i>Here is my current file: [paste code]. Now apply this prompt to it.</i>" • If the output breaks something, paste the error back and say "<i>I got this error, fix it while keeping the dreamscape style.</i>" • Do the files in the order above — <code>world.json</code> first because the other files depend on what fields exist in it. • After all 4 files are done, ask Copilot: "<i>In <code>sketch.js</code>, make sure <code>drawMemories</code> is being called with the right arguments and that <code>level.breatheSpeed</code> and <code>level.breatheAmount</code> are being read correctly from the <code>WorldLevel</code> constructor.</i>" — this is a good sanity-check pass.
	<p>Come back once you've got it running and I can help you debug, tweak the feeling, or add the bonus interactive discovery mechanic in more depth!</p>
"Rewrite this <code>world.json</code> for a dreamscape / memory world theme in a p5.js game. The world should be large (at least 2400x1600). Set the background colour to a deep soft purple (like <code>[28, 18, 52]</code>). Add a <code>camera</code> section with a slow lerp value around 0.07 and a	<p>Now I'll rewrite the world.json file with the dreamscape/memory theme:</p> <pre>Edited world.json+66-5</pre> <p>Done! I've rewritten your world.json with a dreamscape/memory theme featuring:</p> <ul style="list-style-type: none"> • World dimensions: 2400×1600 (maintained) • Background: Deep soft purple <code>[28, 18, 52]</code> • Camera: Slow lerp of 0.07 with breathing effect (speed 0.4, amount 3)

breatheSpeed of 0.4 and breatheAmount of 3. Add a memories array with at least 8 objects, each with an x, y, symbol (choose from: "△", "○", "◆", "▷", "∞", "◇"), and a label (a short poetic 3–5 word phrase like "the day you left" or "warm summer static"). Keep the existing obstacles array but replace the placeholder with 6–8 soft floating rectangular shapes scattered across the world."

- **Memories array:** 8 objects scattered throughout the world with poetic labels ("the day you left", "warm summer static", "echoes of laughter", etc.) and symbolic characters (△, ○, ◆, ▷, ∞, ◇)
- **Obstacles:** 8 soft floating rectangular shapes with varied dimensions and rounded corners, distributed across the world

The memories are positioned strategically across the world space to create a sense of exploration and discovery in this dreamlike environment.

-
-
-
-

Claude Haiku 4.5 • 1x

```
(-) world.json > ...
  3   "world": {
  4     "bg": [28, 18, 52],
  5     "gridStep": 160
  6   },
  7   "camera": {
  8     "lerp": 0.12
  9     "lerp": 0.07,
10     "breatheSpeed": 0.4,
11     "breatheAmount": 3
12   },
13   "playerStart": {
14     "x": 300,
15     "y": 300,
16     "x": 400,
17     "y": 400,
18     "speed": 3
19   },
20   "obstacles": [{"x": 40, "y": 40, "w": 80, "h": 80, "r": 10}]
21   "memories": [
22     {
23       "x": 300,
24       "y": 500,
25       "symbol": "△",
26       "label": "the day you left"
27     },
28     {
29       "x": 800,
30       "y": 300,
31       "symbol": "○",
32       "label": "warm summer static"
33     },
34     {
35       "x": 1200,
36       "y": 700,
37       "symbol": "◆",
38       "label": "echoes of laughter"
39     }
40   ]
41 }
```

actually change the background colour to a more dreamy like purple

Changed the background to [120, 100, 160] — a softer, more ethereal purple that feels more dreamy and magical.

Rewrite this WorldLevel.js for a p5.js dreamscape / memory world. Keep the class structure and constructor reading from JSON. Make these changes:

(1) drawBackground() should fill with the world's bg colour and draw 3 layers of soft translucent ellipses drifting across the background as ambient shapes — use frameCount with slow sine waves to gently shift their y position.

(2) drawWorld() should draw obstacles as soft rounded rectangles with no stroke, in muted lavender/rose/mint pastel colours, cycling through a small palette array.

(3) Add a drawMemories(player) method that loops through this.memories (loaded from JSON), draws each as a glowing pulsing symbol using a sine wave on frameCount for opacity, and if the player is within 80px, draws the label text fading in beneath it.

(4) Update drawHUD to

Done! I've rewritten [WorldLevel.js](#) with a dreamy memory world aesthetic:

- **drawBackground()**: Fills with the world's bg color and renders 3 layers of soft translucent ellipses that drift vertically using sine waves at different speeds, creating a gentle ambient movement
 - **drawWorld()**: Draws obstacles as soft rounded rectangles with no stroke, cycling through a muted pastel palette (rose, blue, mint, peach, lavender, sage)
 - **drawMemories(player)**: New method that loops through memories, draws glowing pulsing symbols using sine waves for opacity, and displays poetic labels when the player is within 80px with smooth fade-in
 - **drawHUD()**: Minimal, just a small italic poetic line ("drifting through memories...") in light purple in the top left, no debug info

Make sure to call `drawMemories(player)` in your `sketch.js` draw loop after

drawing the world!
1 CLASS WORLDLEVEL

```
2 class NoCamera {
3     constructor(json) {
4         // NEW: Camera tuning knob from JSON (data-driven)
5         // [camera tuning] from JSON]
6         this.camLerp = json.camera?.lerp ?? 0.12;
7         this.breatheSpeed = json.camera?.breatheSpeed ?? 0.4;
8         this.breatheAmount = json.camera?.breatheAmount ?? 3;
9
10        // Pastel palette for obstacles
11        this.pastelPalette = [
12            [200, 150, 180], // muted rose
13            [150, 180, 200], // muted blue
14            [180, 200, 150], // muted mint
15            [200, 180, 150], // muted peach
16            [170, 160, 190], // muted lavender
17            [160, 190, 180], // muted sage
18        ];
19    }
20
21    drawBackground() {
22        background(220);
23        background(this.bg[0], this.bg[1], this.bg[2]);
24
25        // 3 layers of drifting translucent ellipses
26        noStroke();
27
28        // Layer 1 - slow, large
29        fill(200, 180, 220, 30);
30        let y1 = 300 + sin(frameCount * 0.002) * 150;
31        ellipse(400, y1, 500, 300);
32        ellipse(1800, y1 + 200, 450, 280);
33
34        // Layer 2 - medium speed
35        fill(180, 160, 200, 25);
36        let y2 = 600 + sin(frameCount * 0.003 + 1) * 180;
37        ellipse(900, y2, 600, 350);
38        ellipse(1300, y2 - 150, 520, 320);
39
40        // Layer 3 - slightly faster
41        fill(160, 140, 180, 20);
42        let y3 = 900 + sin(frameCount * 0.004 + 2) * 120;
43        ellipse(600, y3, 700, 400);
44        ellipse(1000, y3 - 100, 650, 380);
45
46        // Layer 4 - fast
47        fill(140, 120, 160, 15);
48        let y4 = 1200 + sin(frameCount * 0.005 + 3) * 100;
49        ellipse(300, y4, 500, 300);
50        ellipse(700, y4 - 150, 450, 250);
51
52        // Layer 5 - very fast
53        fill(120, 100, 140, 10);
54        let y5 = 1500 + sin(frameCount * 0.006 + 4) * 80;
55        ellipse(100, y5, 400, 250);
56        ellipse(500, y5 - 100, 350, 200);
57
58        // Layer 6 - super fast
59        fill(100, 80, 120, 5);
60        let y6 = 1800 + sin(frameCount * 0.007 + 5) * 60;
61        ellipse(200, y6, 300, 200);
62        ellipse(400, y6 - 100, 250, 150);
63
64        // Layer 7 - super duper fast
65        fill(80, 60, 100, 3);
66        let y7 = 2100 + sin(frameCount * 0.008 + 6) * 40;
67        ellipse(300, y7, 200, 150);
68        ellipse(500, y7 - 80, 180, 120);
69
70        // Layer 8 - super duper duper fast
71        fill(60, 40, 80, 2);
72        let y8 = 2400 + sin(frameCount * 0.009 + 7) * 20;
73        ellipse(400, y8, 150, 100);
74        ellipse(600, y8 - 60, 140, 90);
75
76        // Layer 9 - super duper duper duper fast
77        fill(40, 20, 60, 1);
78        let y9 = 2700 + sin(frameCount * 0.01 + 8) * 10;
79        ellipse(500, y9, 100, 80);
80        ellipse(700, y9 - 40, 90, 70);
81
82        // Layer 10 - super duper duper duper duper fast
83        fill(20, 10, 40, 0.5);
84        let y10 = 3000 + sin(frameCount * 0.011 + 9) * 5;
85        ellipse(600, y10, 80, 60);
86        ellipse(800, y10 - 20, 70, 50);
87
88        // Layer 11 - super duper duper duper duper duper fast
89        fill(10, 5, 20, 0.2);
90        let y11 = 3300 + sin(frameCount * 0.012 + 10) * 3;
91        ellipse(700, y11, 60, 40);
92        ellipse(900, y11 - 10, 50, 30);
93
94        // Layer 12 - super duper duper duper duper duper duper fast
95        fill(5, 2.5, 10, 0.1);
96        let y12 = 3600 + sin(frameCount * 0.013 + 11) * 2;
97        ellipse(800, y12, 40, 30);
98        ellipse(1000, y12 - 5, 30, 20);
99
100        // Layer 13 - super duper duper duper duper duper duper duper fast
101        fill(2.5, 1.25, 5, 0.05);
102        let y13 = 3900 + sin(frameCount * 0.014 + 12) * 1;
103        ellipse(900, y13, 30, 20);
104        ellipse(1100, y13 - 2, 20, 10);
105
106        // Layer 14 - super duper duper duper duper duper duper duper duper fast
107        fill(1.25, 0.625, 2.5, 0.025);
108        let y14 = 4200 + sin(frameCount * 0.015 + 13) * 0.5;
109        ellipse(1000, y14, 20, 10);
110        ellipse(1200, y14 - 1, 10, 5);
111
112        // Layer 15 - super duper duper duper duper duper duper duper duper duper fast
113        fill(0.625, 0.3125, 1.25, 0.0125);
114        let y15 = 4500 + sin(frameCount * 0.016 + 14) * 0.25;
115        ellipse(1100, y15, 10, 5);
116        ellipse(1300, y15 - 0.5, 5, 2.5);
117
118        // Layer 16 - super duper fast
119        fill(0.3125, 0.15625, 0.625, 0.00625);
120        let y16 = 4800 + sin(frameCount * 0.017 + 15) * 0.125;
121        ellipse(1200, y16, 5, 2.5);
122        ellipse(1400, y16 - 0.25, 2.5, 1.25);
123
124        // Layer 17 - super duper fast
125        fill(0.15625, 0.078125, 0.3125, 0.003125);
126        let y17 = 5100 + sin(frameCount * 0.018 + 16) * 0.0625;
127        ellipse(1300, y17, 2.5, 1.25);
128        ellipse(1500, y17 - 0.125, 1.25, 0.625);
129
130        // Layer 18 - super duper fast
131        fill(0.078125, 0.0390625, 0.15625, 0.0015625);
132        let y18 = 5400 + sin(frameCount * 0.019 + 17) * 0.03125;
133        ellipse(1400, y18, 1.25, 0.625);
134        ellipse(1600, y18 - 0.0625, 0.625, 0.3125);
135
136        // Layer 19 - super duper fast
137        fill(0.0390625, 0.01953125, 0.078125, 0.00078125);
138        let y19 = 5700 + sin(frameCount * 0.02 + 18) * 0.015625;
139        ellipse(1500, y19, 0.625, 0.3125);
140        ellipse(1700, y19 - 0.03125, 0.3125, 0.15625);
141
142        // Layer 20 - super duper fast
143        fill(0.01953125, 0.009765625, 0.0390625, 0.000390625);
144        let y20 = 6000 + sin(frameCount * 0.021 + 19) * 0.0078125;
145        ellipse(1600, y20, 0.3125, 0.15625);
146        ellipse(1800, y20 - 0.015625, 0.15625, 0.078125);
147
148        // Layer 21 - super duper fast
149        fill(0.009765625, 0.0048828125, 0.01953125, 0.0001953125);
150        let y21 = 6300 + sin(frameCount * 0.022 + 20) * 0.00390625;
151        ellipse(1700, y21, 0.15625, 0.078125);
152        ellipse(1900, y21 - 0.0078125, 0.078125, 0.0390625);
153
154        // Layer 22 - super duper fast
155        fill(0.0048828125, 0.00244140625, 0.009765625, 0.00009765625);
156        let y22 = 6600 + sin(frameCount * 0.023 + 21) * 0.001953125;
157        ellipse(1800, y22, 0.078125, 0.0390625);
158        ellipse(2000, y22 - 0.00390625, 0.0390625, 0.01953125);
159
160        // Layer 23 - super duper fast
161        fill(0.00244140625, 0.001220703125, 0.0048828125, 0.000048828125);
162        let y23 = 6900 + sin(frameCount * 0.024 + 22) * 0.0009765625;
163        ellipse(1900, y23, 0.0390625, 0.01953125);
164        ellipse(2100, y23 - 0.001953125, 0.01953125, 0.009765625);
165
166        // Layer 24 - super duper fast
167        fill(0.001220703125, 0.0006103515625, 0.00244140625, 0.0000244140625);
168        let y24 = 7200 + sin(frameCount * 0.025 + 23) * 0.00048828125;
169        ellipse(2000, y24, 0.01953125, 0.009765625);
170        ellipse(2200, y24 - 0.0009765625, 0.009765625, 0.0048828125);
171
172        // Layer 25 - super duper fast
173        fill(0.0006103515625, 0.00030517578125, 0.001220703125, 0.00001220703125);
174        let y25 = 7500 + sin(frameCount * 0.026 + 24) * 0.000244140625;
175        ellipse(2100, y25, 0.009765625, 0.0048828125);
176        ellipse(2300, y25 - 0.00048828125, 0.0048828125, 0.00244140625);
177
178        // Layer 26 - super duper fast
179        fill(0.00030517578125, 0.000152587890625, 0.0006103515625, 0.000006103515625);
180        let y26 = 7800 + sin(frameCount * 0.027 + 25) * 0.0001220703125;
181        ellipse(2200, y26, 0.0048828125, 0.00244140625);
182        ellipse(2400, y26 - 0.000244140625, 0.00244140625, 0.001220703125);
183
184        // Layer 27 - super duper fast
185        fill(0.000152587890625, 0.0000762939453125, 0.00030517578125, 0.0000030517578125);
186        let y27 = 8100 + sin(frameCount * 0.028 + 26) * 0.00006103515625;
187        ellipse(2300, y27, 0.00244140625, 0.001220703125);
188        ellipse(2500, y27 - 0.0001220703125, 0.001220703125, 0.0006103515625);
189
190        // Layer 28 - super duper fast
191        fill(0.0000762939453125, 0.00003814697265625, 0.000152587890625, 0.00000152587890625);
192        let y28 = 8400 + sin(frameCount * 0.029 + 27) * 0.000030517578125;
193        ellipse(2400, y28, 0.001220703125, 0.0006103515625);
194        ellipse(2600, y28 - 0.00030517578125, 0.0006103515625, 0.00030517578125);
195
196        // Layer 29 - super duper fast
197        fill(0.00003814697265625, 0.000019073486328125, 0.0000762939453125, 0.000000762939453125);
198        let y29 = 8700 + sin(frameCount * 0.03 + 28) * 0.0000152587890625;
199        ellipse(2500, y29, 0.0006103515625, 0.00030517578125);
200        ellipse(2700, y29 - 0.000152587890625, 0.00030517578125, 0.000152587890625);
201
202        // Layer 30 - super duper fast
203        fill(0.000019073486328125, 0.0000095367431640625, 0.00003814697265625, 0.0000003814697265625);
204        let y30 = 9000 + sin(frameCount * 0.031 + 29) * 0.00000762939453125;
205        ellipse(2600, y30, 0.00030517578125, 0.000152587890625);
206        ellipse(2800, y30 - 0.0000762939453125, 0.000152587890625, 0.0000762939453125);
207
208        // Layer 31 - super duper fast
209        fill(0.0000095367431640625, 0.00000476837158203125, 0.000019073486328125, 0.00000019073486328125);
210        let y31 = 9300 + sin(frameCount * 0.032 + 30) * 0.000003814697265625;
211        ellipse(2700, y31, 0.000152587890625, 0.0000762939453125);
212        ellipse(2900, y31 - 0.00003814697265625, 0.0000762939453125, 0.00003814697265625);
213
214        // Layer 32 - super duper fast
215        fill(0.00000476837158203125, 0.000002384185791015625, 0.0000095367431640625, 0.000000095367431640625);
216        let y32 = 9600 + sin(frameCount * 0.033 + 31) * 0.0000019073486328125;
217        ellipse(2800, y32, 0.0000762939453125, 0.00003814697265625);
218        ellipse(3000, y32 - 0.000019073486328125, 0.00003814697265625, 0.000019073486328125);
219
220        // Layer 33 - super duper fast
221        fill(0.000002384185791015625, 0.0000011920928955078125, 0.00000476837158203125, 0.0000000476837158203125);
222        let y33 = 9900 + sin(frameCount * 0.034 + 32) * 0.0000019073486328125;
223        ellipse(2900, y33, 0.00003814697265625, 0.000019073486328125);
224        ellipse(3100, y33 - 0.0000019073486328125, 0.000019073486328125, 0.0000019073486328125);
225
226        // Layer 34 - super duper fast
227        fill(0.0000011920928955078125, 0.00000059604644775390625, 0.000002384185791015625, 0.00000002384185791015625);
228        let y34 = 10200 + sin(frameCount * 0.035 + 33) * 0.0000019073486328125;
229        ellipse(3000, y34, 0.000019073486328125, 0.0000095367431640625);
230        ellipse(3200, y34 - 0.0000019073486328125, 0.0000095367431640625, 0.0000019073486328125);
231
232        // Layer 35 - super duper fast
233        fill(0.00000059604644775390625, 0.000000298023223877453125, 0.0000011920928955078125, 0.000000011920928955078125);
234        let y35 = 10500 + sin(frameCount * 0.036 + 34) * 0.0000019073486328125;
235        ellipse(3100, y35, 0.0000095367431640625, 0.00000476837158203125);
236        ellipse(3300, y35 - 0.0000019073486328125, 0.00000476837158203125, 0.0000019073486328125);
237
238        // Layer 36 - super duper fast
239        fill(0.000000298023223877453125, 0.00000014901161193872891, 0.00000059604644775390625, 0.000000014901161193872891);
240        let y36 = 10800 + sin(frameCount * 0.037 + 35) * 0.0000019073486328125;
241        ellipse(3200, y36, 0.00000476837158203125, 0.000002384185791015625);
242        ellipse(3400, y36 - 0.0000019073486328125, 0.000002384185791015625, 0.0000019073486328125);
243
244        // Layer 37 - super duper fast
245        fill(0.00000014901161193872891, 0.00000007450580596936445, 0.000000298023223877453125, 0.0000000298023223877453125);
246        let y37 = 11100 + sin(frameCount * 0.038 + 36) * 0.0000019073486328125;
247        ellipse(3300, y37, 0.000002384185791015625, 0.0000011920928955078125);
248        ellipse(3500, y37 - 0.0000019073486328125, 0.0000011920928955078125, 0.0000019073486328125);
249
250        // Layer 38 - super duper fast
251        fill(0.00000007450580596936445, 0.00000003725290298468222, 0.00000014901161193872891, 0.000000014901161193872891);
252        let y38 = 11400 + sin(frameCount * 0.039 + 37) * 0.0000019073486328125;
253        ellipse(3400, y38, 0.0000011920928955078125, 0.00000059604644775390625);
254        ellipse(3600, y38 - 0.0000019073486328125, 0.00000059604644775390625, 0.0000019073486328125);
255
256        // Layer 39 - super duper fast
257        fill(0.00000003725290298468222, 0.00000001862645149234111, 0.00000007450580596936445, 0.00000001862645149234111);
258        let y39 = 11700 + sin(frameCount * 0.04 + 38) * 0.0000019073486328125;
259        ellipse(3500, y39, 0.00000059604644775390625, 0.000000298023223877453125);
260        ellipse(3700, y39 - 0.0000019073486328125, 0.000000298023223877453125, 0.0000019073486328125);
261
262        // Layer 40 - super duper fast
263        fill(0.00000001862645149234111, 0.000000009313225746170555, 0.00000003725290298468222, 0.00000001862645149234111);
264        let y40 = 12000 + sin(frameCount * 0.041 + 39) * 0.0000019073486328125;
265        ellipse(3600, y40, 0.000000298023223877453125, 0.00000014901161193872891);
266        ellipse(3800, y40 - 0.0000019073486328125, 0.00000014901161193872891, 0.0000019073486328125);
267
268        // Layer 41 - super duper fast
269        fill(0.000000009313225746170555, 0.000000004656612873085277, 0.00000001862645149234111, 0.00000001862645149234111);
270        let y41 = 12300 + sin(frameCount * 0.042 + 40) * 0.0000019073486328125;
271        ellipse(3700, y41, 0.00000014901161193872891, 0.00000007450580596936445);
272        ellipse(3900, y41 - 0.0000019073486328125, 0.00000007450580596936445, 0.0000019073486328125);
273
274        // Layer 42 - super duper fast
275        fill(0.000000004656612873085277, 0.000000002328306436542638, 0.000000009313225746170555, 0.00000001862645149234111);
276        let y42 = 12600 + sin(frameCount * 0.043 + 41) * 0.0000019073486328125;
277        ellipse(3800, y42, 0.00000007450580596936445, 0.00000003725290298468222);
278        ellipse(4000, y42 - 0.0000019073486328125, 0.00000003725290298468222, 0.0000019073486328125);
279
280        // Layer 43 - super duper fast
281        fill(0.000000002328306436542638, 0.000000001164153218271319, 0.000000004656612873085277, 0.00000001862645149234111);
282        let y43 = 12900 + sin(frameCount * 0.044 + 42) * 0.0000019073486328125;
283        ellipse(3900, y43, 0.00000003725290298468222, 0.00000001862645149234111);
284        ellipse(4100, y43 - 0.0000019073486328125, 0.00000001862645149234111, 0.0000019073486328125);
285
286        // Layer 44 - super duper fast
287        fill(0.000000001164153218271319, 0.000000000582076609135659, 0.000000002328306436542638, 0.00000001862645149234111);
288        let y44 = 13200 + sin(frameCount * 0.045 + 43) * 0.0000019073486328125;
289        ellipse(4000, y44, 0.00000001862645149234111, 0.000000009313225746170555);
290        ellipse(4200, y44 - 0.0000019073486328125, 0.000000009313225746170555, 0.0000019073486328125);
291
292        // Layer 45 - super duper fast
293        fill(0.000000000582076609135659, 0.000000000291038304587829, 0.000000001164153218271319, 0.00000001862645149234111);
294        let y45 = 13500 + sin(frameCount * 0.046 + 44) * 0.0000019073486328125;
295        ellipse(4100, y45, 0.000000009313225746170555, 0.000000004656612873085277);
296        ellipse(4300, y45 - 0.0000019073486328125, 0.000000004656612873085277, 0.0000019073486328125);
297
298        // Layer 46 - super duper fast
299        fill(0.000000000291038304587829, 0.000000000145519151443964, 0.000000000582076609135659, 0.00000001862645149234111);
300        let y50 = 13800 + sin(frameCount * 0.047 + 45) * 0.0000019073486328125;
301        ellipse(4200, y50, 0.000000004656612873085277, 0.000000002328306436542638);
302        ellipse(4400, y50 - 0.0000019073486328125, 0.0000000023283064365
```

<p>be minimal — just a small italic poetic line in the top left in a light colour, no coordinate debug info</p>	
<p>make the ellipses drifting across the background as ambient shapes more soft and it could have more of a gradient look</p>	<pre> 1 class WorldLevel { 2 3 drawBackground() { 4 // 3 layers of drifting translucent ellipses 5 // 3 layers of drifting translucent ellipses with soft gradient appearance 6 noStroke(); 7 8 // Layer 1 - slow, large 9 fill(200, 180, 220, 30); 10 let y1 = 300 + sin(frameCount * 0.002) * 150; 11 ellipse(400, y1, 500, 300); 12 ellipse(1800, y1 + 200, 450, 280); 13 for (let i = 0; i < 3; i++) { 14 fill(200, 180, 220, 15 - i * 5); 15 ellipse(400, y1, 500 + i * 60, 300 + i * 40); 16 } 17 for (let i = 0; i < 3; i++) { 18 fill(210, 190, 230, 12 - i * 4); 19 ellipse(1800, y1 + 200, 450 + i * 50, 280 + i * 35); 20 } 21 22 // Layer 2 - medium speed 23 fill(180, 160, 200, 25); 24 // Layer 2 - medium speed, gradient effect 25 let y2 = 600 + sin(frameCount * 0.003 + 1) * 180; 26 ellipse(900, y2, 600, 350); 27 ellipse(1300, y2 - 150, 520, 320); 28 for (let i = 0; i < 3; i++) { 29 fill(180, 160, 200, 12 - i * 4); 30 ellipse(900, y2, 600 + i * 70, 350 + i * 45); 31 } 32 for (let i = 0; i < 3; i++) { 33 fill(190, 170, 210, 10 - i * 3); 34 ellipse(1300, y2 - 150, 520 + i * 60, 320 + i * 40); 35 } 36 37 // Layer 3 - slightly faster 38 fill(220, 200, 240, 20); 39 // Layer 3 - slightly faster, very soft gradient 40 let v3 = 800 + sin(frameCount * 0.0025 + 2) * 120; 41 } 42 }</pre>
<p>make it so that the deeper or more down I go, the darker the background is and the top or higher I go it looks lighter. just make it sort of like gradient variations building around the current bg colour</p>	<p>Done! The background now has a vertical gradient that gets lighter toward the top and darker toward the bottom, based on your base purple color. The gradient amount is ± 50 RGB units from the center color, creating a subtle but noticeable shift as you explore the world.</p>

```

1  class WorldLevel {
2    constructor(json) {
25      [160, 190, 180], // muted sage
26    ];
27  }
28
29  drawBackground() {
30+   background(this.bg[0], this.bg[1], this.bg[2]);
31+   // Create vertical gradient: lighter at top, darker at bottom
32+   noStroke();
33+
34+   const baseR = this.bg[0];
35+   const baseG = this.bg[1];
36+   const baseB = this.bg[2];
37+
38+   // Draw gradient lines from top to bottom
39+   for (let y = 0; y < this.h; y++) {
40+     // Normalize y position (0 at top, 1 at bottom)
41+     const t = y / this.h;
42+
43+     // Lighter at top (add ~40-50), darker at bottom (subtract ~40-50)
44+     const gradientAmount = 50;
45+     const r = constrain(baseR + (1 - t) * gradientAmount - t * gradientAmount, 0, 255);
46+     const g = constrain(baseG + (1 - t) * gradientAmount - t * gradientAmount, 0, 255);
47+     const b = constrain(baseB + (1 - t) * gradientAmount - t * gradientAmount, 0, 255);
48+
49+     stroke(r, g, b);
50+     line(0, y, this.w, y);
51+
52+   noStroke();
53+
54   // 3 layers of drifting translucent ellipses with soft gradient appearance
55   noStroke();
56
57   // Layer 1 - slow, large, soft gradient with radial effect
58   let y1 = 300 + sin(frameCount * 0.002) * 150;
59   for (let i = 0; i < 3; i++) {
    fill(200, 180, 220, 15 - i * 5);

```

make it so that the obstacles have a glowing white border around them, making it look more ethereal

Done! The obstacles now have a glowing white border that creates a soft, layered ethereal effect. The glow consists of multiple strokes with decreasing opacity, radiating outward from each rectangle to give them a dreamy, luminous appearance.

```

101
102  drawWorld() {
103+   // Draw obstacles as soft rounded rectangles with pastel colors
104+   // Draw obstacles as soft rounded rectangles with pastel colors and glowing white borders
105   noStroke();
106   for (let i = 0; i < this.obstacles.length; i++) {
107     const o = this.obstacles[i];
108     const color = this.pastelPalette[i % this.pastelPalette.length];
109+
110+    // Draw glowing white border with multiple layers for ethereal effect
111+    for (let glow = 8; glow > 0; glow--) {
112+      stroke(255, 255, 255, 40 / glow);
113+      strokeWeight(glow);
114+      noFill();
115+      rect(o.x, o.y, o.w, o.h, o.r ?? 8);
116+
117+    // Draw the main pastel rectangle
118+    noStroke();
119    fill(color[0], color[1], color[2], 180);
120    rect(o.x, o.y, o.w, o.h, o.r ?? 8);
121  }

```

Rewrite this Player.js for a dreamscape p5.js experience. Instead of instant movement, give the player inertia — use

I'll open [Player.js](#) to inspect current code, then replace it with an inertia-based player.

I'll update [Player.js](#) to implement inertial movement and the soft glowing draw.

this.vx and this.vy velocity variables. Each frame, add input direction to velocity scaled by an acceleration of 0.4, then multiply velocity by a friction of 0.88 so the player glides to a stop. Cap speed at this.s. Draw the player as a soft glowing white ellipse (radius ~10) with a larger translucent halo ellipse behind it (radius ~22, low opacity). No stroke

I'll apply the same patch again and include a short explanation.

Edited

Player.js+49-6

Updated [Player.js](#) — the player now uses inertia with `vx`/`vy`, acceleration 0.4, friction 0.88, speed capped at `this.s`. Drawing is a soft glowing white ellipse with a translucent halo.

```
this.y = y;
this.s = speed ?? 3;
this.s = speed ?? 3; // max speed

Keep Undo ⌂
```

// velocity for inertia
this.vx = 0;
this.vy = 0;

// tuning
this.accel = 0.4;
this.friction = 0.88;

```
}
```

updateInput() {
 // directional input (-1, 0, 1)
 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;

    // normalize to avoid faster diagonal acceleration
    let nx = 0,
        ny = 0;
    const len = sqrt(dx * dx + dy * dy);
    if (len > 0) {
        nx = dx / len;
        ny = dy / len;
    }
}
```

```
Keep Undo ⌂
```

Rewrite this sketch.js to work with the updated WorldLevel, Player, and world.json for a dreamscape experience. Keep the same setup structure. Add these features: (1) Camera breathe — each frame offset the camera by $\sin(\text{frameCount} * \text{breatheSpeed}) * \text{breatheAmount}$ in both x and y, where those values come

I'll add a short plan and then update [sketch.js](#) to implement camera breathe, call [drawMemories](#), add vignette, and keep/reset R key.

Now I'll update `sketch.js` to add camera breathing, call `drawMemories` inside the scene, and draw the vignette overlay. I'll replace the `draw()` function accordingly.

Edited

sketch.js+27-0

Starting: Run quick sanity checks (3/3)

Progress update: I updated `sketch.js` to add camera breathing, draw memories inside the scene, and render a radial vignette overlay. Next I'll run a quick read of the file to confirm the changes.

<p>from level.breatheSpeed and level.breatheAmount. (2) After drawing the world and player (inside the push/pop), call level.drawMemories (player). (3) After the pop, draw a vignette overlay: a dark radial gradient rect covering the full canvas using drawingContext and a canvas radial gradient from transparent centre to rgba(10,5,20,0.7) at the edges. (4) Keep the R key reset. Remove the HUD coordinate debug line since the new HUD is poetic</p>	<pre> // Camera breathe offset (gentle sinusoidal offset) const breatheSpeed = level.breatheSpeed ?? 0.07; const breatheAmount = level.breatheAmount ?? 0; const breatheX = sin(frameCount * breatheSpeed) * breatheAmount; const breatheY = cos(frameCount * breatheSpeed) * breatheAmount; camX += breatheX; camY += breatheY; level.drawBackground(); push(); translate(-camX, -camY); level.drawWorld(); player.draw(); // Draw memories within world coordinates so they follow camera translation if (typeof level.drawMemories === "function") level.drawMemories(); pop(); level.drawHUD(player, camX, camY); // Vignette overlay: radial gradient from transparent center to dark push(); // access raw canvas 2D context const ctx = drawingContext; const cx = width / 2; const cy = height / 2; const maxR = sqrt(cx * cx + cy * cy); const grad = ctx.createRadialGradient(cx, cy, 0, cx, cy, maxR); grad.addColorStop(0, "rgba(0,0,0,0)"); grad.addColorStop(1, "rgba(10,5,20,0.7)"); ctx.fillStyle = grad; noStroke(); // draw full-screen rect with gradient </pre>
--	--

<p>currently the camera is breathing too fast I want it to be more subtle and slower</p>	<p>I'll update sketch.js to scale down the effective breathe speed (and slightly damp the amount) so the camera breathes more slowly and subtly. I'll modify the breathe calculation.</p> <p>Edited sketch.js+7-4</p> <p>Done — I scaled the camera breathe to be much slower and slightly more subtle by multiplying <code>breatheSpeed</code> by 0.02 and <code>breatheAmount</code> by 0.6.</p> <p>Reload the sketch; if you'd like it even slower/subtler, tell me a preferred multiplier or I can set explicit values.</p>
--	---

<p>make the rectangles into actual obstacles that "bounce" the player back if</p>	<p>I'm going to add a <code>handleCollisions(player)</code> method to WorldLevel.js and call it from sketch.js after updating the player's position so obstacles will push and bounce the player.</p>
---	---

interacted with/touched	<p>Edited WorldLevel.js+67-0</p> <p>Edited sketch.js+3-0</p> <p>Done — obstacles now act as physical colliders: when the player touches a rectangle they are pushed out and their velocity reflects with a bounce. Test by running the sketch and moving into an obstacle; tell me if you'd like a firmer (less bouncy) or softer (more bouncy) response and I can adjust the restitution.</p>
	<pre> 1 // const o = this.obstacles; 2 const rx = o.x; 3 const ry = o.y; 4 const rw = o.w; 5 const rh = o.h; 6 7 // closest point on rectangle to circle center 8 const closestX = constrain(player.x, rx, rx + rw); 9 const closestY = constrain(player.y, ry, ry + rh); 10 11 let dx = player.x - closestX; 12 let dy = player.y - closestY; 13 let dist2 = dx * dx + dy * dy; 14 15 if (dist2 === 0) { 16 // Circle center is exactly inside or aligned - push out along 17 const left = abs(player.x - rx); 18 const right = abs(rx + rw - player.x); 19 const top = abs(player.y - ry); 20 const bottom = abs(ry + rh - player.y); 21 const minEdge = min(left, right, top, bottom); 22 if (minEdge === left) { 23 dx = 1; 24 dy = 0; 25 } else if (minEdge === right) { 26 dx = -1; 27 dy = 0; 28 } else if (minEdge === top) { 29 dx = 0; 30 dy = 1; 31 } else { 32 dx = 0; 33 dy = -1; 34 } 35 dist2 = dx * dx + dy * dy; 36 }</pre>
make the dynamix background a little more obvious, currently its not obvious enough there	

are things moving in
the background

```
6 |   const horizAmp1 = 80;
7 |   for (let i = 0; i < 4; i++) {
8 |     const a = layer1AlphaBase - i * 10 + sin(t * 0.6 + i) * 8;
9 |     fill(200, 180, 220, constrain(a, 18, 200));
10|     const xOff = sin(t * 0.4 + i) * horizAmp1;
11|     ellipse(400 + xOff, y1 + cos(t * 0.3 + i) * 30, 500 + i * 80,
12|     ellipse(1800 - xOff, y1 + 200 + cos(t * 0.28 + i) * 28, 450 +
13|   }
14|
15|   // Layer 2 - medium speed, gradient effect
16|   // Layer 2 - medium speed, more contrast
17|   let y2 = 600 + sin(frameCount * 0.003 + 1) * 180;
18|   for (let i = 0; i < 3; i++) {
19|     fill(180, 160, 200, 12 - i * 4);
20|     ellipse(900, y2, 600 + i * 70, 350 + i * 45);
21|   }
22|   for (let i = 0; i < 3; i++) {
23|     fill(190, 170, 210, 10 - i * 3);
24|     ellipse(1300, y2 - 150, 520 + i * 60, 320 + i * 40);
25|   }
26|   const layer2AlphaBase = 45;
27|   const horizAmp2 = 60;
28|   for (let i = 0; i < 4; i++) {
29|     const a = layer2AlphaBase - i * 8 + sin(t * 0.9 + i) * 6;
30|     fill(180, 160, 200, constrain(a, 12, 200));
31|     const xOff = cos(t * 0.35 + i) * horizAmp2;
32|     ellipse(900 + xOff, y2 + sin(t * 0.25 + i) * 40, 600 + i * 90,
33|     ellipse(1300 - xOff, y2 - 150 + sin(t * 0.22 + i) * 36, 520 +
34|   }
35|
36|   // Layer 3 - slightly faster, very soft gradient
37|   // Layer 3 - slightly faster, soft but visible
38|   let y3 = 800 + sin(frameCount * 0.0025 + 2) * 120;
39|   for (let i = 0; i < 3; i++) {
40|     fill(220, 200, 240, 8 - i * 2);
41|     ellipse(600 + y3 - 400 + i * 50, 250 + i * 35);
```