Date Used: February 23, 2026

Tool Disclosure: ChatGPT 5.2

Purpose of Use: Help plan and implement a calm, meditative camera experience in p5.js.

Summary of Interaction: Outlined a process to iterate towards completion, then applied updates to my working camera-follow sketch: smooth movement with velocity + friction, lerped camera follow, drift when idle, soft camera bounds, three mood areas with visuals only utilizing shape, slight breathing zoom, and basic comfortable toggles that sync with HUD state.

Human Decision Point(s): Opted for auto-drift on stillness (rather than rest zones) to create peaceful "pause" points, lerped camera follow + soft-clamped bounds with gentle overscroll so nothing snaps/jerks/stops abruptly, dialed breathing zoom way down (and disabled entirely via [M] Reduced Motion), tested [H] High Contrast for readability.

Integrity & Verification Note: Tested each feature incrementally and confirmed toggles work instantly and reset returns to a calm start.

Scope of GenAI Use: Planning, safe parameter ranges, integration/debug help for camera, movement, visuals, and HUD.

Limitations or Misfires: No misfires.

Summary of Process (Human + Tool)

I explained the Week 5 assignment and requirements then posted my current camera-follow code. GenAI responded with a numbered list detailing changes to make and where to make them (camera easing, move with velocity+friction, auto-drift when still, soft camera boundaries, three zones of mood, gentle breathing zoom, and M/H/R comfort settings). I then requested that it edit my code to meet those requirements, at which point I verified that each bullet point was addressed and that the edited version still followed the assignment parameters.

Decision Points & Trade-offs

I used auto-drift so users can pause anywhere there aren't goals, applied a gentle breathing zoom that users can disable with Reduced Motion, soft-clamped the camera with a little overscroll so it doesn't snap to hard edges, and made it asset-free by drawing everything programmatically with shapes and colors so it can be hosted anywhere.

Verification & Judgement

Equation For Success = Moves feel smooth. Camera doesn't stutter. Standing still feels natural. Set distinct mood zones without overpowering them. Keep HUD legible. And of course M/H/R work perfectly everytime.

Limitations, Dead Ends, or Open Questions

Check if motion timing varies across devices

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

You are my p5.js coding coach. I have working base code for a top-down scrolling world with a camera that centers on the player using: - player in WORLD coords - cam in WORLD coords - translate(-cam.x, -cam.y) to draw the world - HUD drawn in screen space after pop() Your job: give me a clear, numbered set of implementation steps (and brief "why" for each) to modify my base code into this Week 5 mission: "Create a reflective or meditative camera experience that scrolls through a world larger than the screen. Use pacing and motion to evoke emotion." Rules: - Start from my base code. Do NOT rewrite from scratch; refer to the exact parts of the code you're changing (e.g., "in the camera update section…", "in the draw world section…"). - Keep it GitHub Pages friendly. No external images. If you suggest sound, it must be optional and generated in code, and you must mention the user-gesture requirement. - The experience should feel calm, not challenging. No flashing, no sudden camera shake. - I need steps that are small and incremental: each step should say what to add/change + where in the code + how to test that step quickly. In your steps, make sure you cover ALL of these upgrades: 1) Calm camera follow (no snap): - Replace direct cam.x/cam.y assignment with eased follow (lerp/smoothing). - Explain how to tune the smoothing factor for "floaty" motion. 2) Pacing + stillness: - Add a system that creates "slow travel + pause" moments (choose ONE approach): A) "Rest zones" in the world that gently slow the camera/player and encourage lingering, OR B) "Auto-drift" when the player stops moving, with a slow return to stillness. - Explain how this pacing supports a meditative feel. 3) Gentle player movement: - Replace instant position changes with acceleration/deceleration (velocity + friction). - Optional: add SHIFT for "slow walk" precision. - Ensure diagonal speed is consistent. 4) World visuals that feel reflective: - Reduce the harsh grid feel; propose calmer visuals using only shapes/colors: e.g., soft background gradients, sparse landmarks, subtle particles, or flowing lines. - Create 3 distinct mood areas across the large world (different palettes/effects) and explain how you detect which area the camera/player is in. 5) Subtle "breathing" motion: - Add a very low-amplitude breathing effect (gentle zoom OR tiny oscillation). - Provide safe parameter ranges so it stays subtle and non-dizzying. 6) Comfort & inclusivity toggles (must be simple and readable in HUD): - [M] Reduced Motion: disable breathing/extra drift and use simpler camera easing. - [H] High

Contrast: improve readability of world + HUD. - [R] Reset: return player/camera to a calm starting location. - Explain where to store these flags and how to display state in HUD. 7) World/camera bounds decision: - Decide whether to clamp camera to the world or allow slight overscroll. - If clamping, do it smoothly (no hard stop). Describe the math and where to apply it. Output format: - ONLY the numbered steps + short “why” + quick test per step. - After the steps, include a short “parameter tuning guide” (which numbers to tweak for calmer vs. more active motion). - Do NOT provide the full rewritten code unless I ask; I want the plan first. Here is my base code (modify this plan around it): /* Week 5 — Example 1: Top-Down Camera Follow (Centered, No Bounds) Course: GBDA302 | Instructors: Dr. Karen Cochrane & David Han Date: Feb. 12, 2026 Move: WASD/Arrows Goal: - Keep player position in world space - Compute a camera offset from the player (view state) - Draw world using translate(-cam.x, -cam.y) - Draw HUD in screen space (no translate) */ let player = { x: 300, y: 300, s: 3 }; // player in WORLD coords let cam = { x: 0, y: 0 }; // camera top-left in WORLD coords // World size (we draw a world rectangle + features, but we do NOT clamp camera) const WORLD_W = 2400; const WORLD_H = 1600; // Canvas / viewport size (SCREEN coords) const VIEW_W = 800; const VIEW_H = 480; function setup() { createCanvas(VIEW_W, VIEW_H); textFont("sans-serif"); textSize(14); noStroke(); } function draw() { // ---------- 1) UPDATE GAME STATE (WORLD) ---------- // Input becomes a direction vector (dx, dy) const dx = (keyIsDown(RIGHT_ARROW) || keyIsDown(68)) - (keyIsDown(LEFT_ARROW) || keyIsDown(65)); const dy = (keyIsDown(DOWN_ARROW) || keyIsDown(83)) - (keyIsDown(UP_ARROW) || keyIsDown(87)); // Cheap diagonal normalization so diagonals aren’t faster const len = max(1, abs(dx) + abs(dy)); // Move player in WORLD space (no bounds in Example 1) player.x += (dx / len) * player.s; player.y += (dy / len) * player.s; // ---------- 2) UPDATE VIEW STATE (CAMERA) ---------- // Center camera on player (NO constrain / bounds here) cam.x = player.x - width / 2; cam.y = player.y - height / 2; // ---------- 3) DRAW ---------- background(220); // Draw the WORLD (scrolling layer) in world space push(); translate(-cam.x, -cam.y); // World background rectangle (so you can see the “world area”) noStroke(); fill(235); rect(0, 0, WORLD_W, WORLD_H); // Grid lines make camera motion easy to see stroke(245); for (let x = 0; x <= WORLD_W; x += 160) line(x, 0, x, WORLD_H); for (let y = 0; y <= WORLD_H; y += 160) line(0, y, WORLD_W, y); // Obstacles (static world features) noStroke(); fill(170, 190, 210); for (let i = 0; i < 30; i++) { const x = (i * 280) % WORLD_W; const y = (i * 180) % WORLD_H; rect(x + 40, y + 40, 80, 80, 10); } // Player (in world space) fill(50, 110, 255); rect(player.x - 12, player.y - 12, 24, 24, 5); pop(); // HUD (screen space): drawn AFTER pop(), so it does not move with camera noStroke(); fill(20); text("Week 5 — Centered camera (no bounds). WASD/Arrows to move.", 12, 20); text("Player(world): " + (player.x | 0) + ", " + (player.y | 0) + " Cam(world): " + (cam.x | 0) + ", " + (cam.y | 0), 12, 40,); }

Thought for 1m 4s

1. **Add “Week 5 settings + state” globals (no behavior change yet)**
    - **Where:** Top of file, near `player` / `cam`.
    - **Add/change:**
        - Player motion state: `player.vx`, `player.vy` (or a `vel` object).
        - Camera target + smoothing: `cam.tx`, `cam.ty`, `camSmooth`.

      - Pacing/drift state: `driftAmt`, `driftSpeed`, `driftFade`.
      - Breathing params: `breathOn`, `breathAmp`, `breathRate`.
      - Toggles: `let opt = { reducedMotion:false, highContrast:false };`
      - Reset spawn: `const START_X = 300, START_Y = 300;`
   - **Why:** Centralizes the “calm feel” parameters so you can tune emotion without hunting through code.
   - **Quick test:** Run sketch; everything should look/feel identical (you haven’t wired it in yet).
2. **Replace instant player movement with acceleration + friction (gentle start/stop)**
   - **Where:** In `draw()` → **1) UPDATE GAME STATE (WORLD)**, replace the `player.x += ...` / `player.y += ...` lines.
   - **Add/change:**
      - Keep your `dx/dy` input, but compute a normalized input direction using Euclidean length: `sqrt(dx*dx + dy*dy)` (so diagonals are consistent).
      - Apply acceleration to velocity, then velocity to position:
         - `player.vx += ax; player.vy += ay;`
         - `player.vx *= friction; player.vy *= friction;`
         - `player.x += player.vx; player.y += player.vy;`
      - **Optional SHIFT slow-walk:** if `keyIsDown(SHIFT)` then lower max speed and/or acceleration.
   - **Why:** Accel/decel instantly makes movement feel calm and “floaty” instead of twitchy.
   - **Quick test:** Tap a movement key briefly; the player should glide a little and gently settle instead of stopping on a dime. Hold diagonals; speed should match cardinal directions.
3. **Add a max-speed clamp (keeps motion serene)**
   - **Where:** Right after you update `player.vx/player.vy` in the same **UPDATE GAME STATE** section.
   - **Add/change:**
      - Compute current speed `sp = sqrt(vx*vx + vy*vy)` and clamp to `maxSpeed` (or `maxSpeed * slowWalkMult` when SHIFT is held).
   - **Why:** Prevents accidental “fast skating,” keeping the experience meditative.
   - **Quick test:** Hold a direction for several seconds; the player should reach a steady, capped speed (not accelerate forever).
4. **(Optional but recommended) Soft player bounds inside the world**
   - **Where:** Still in **UPDATE GAME STATE**, after updating `player.x/y`.
   - **Add/change:**
      - Constrain player to `0..WORLD_W` and `0..WORLD_H`, but **dampen** velocity when hitting edges (e.g., if clamped on X, multiply `vx` by `0.2` instead of flipping it).

    - **Why:** Keeps you within the designed mood areas and avoids drifting into “empty nothing.”
    - **Quick test:** Move into an edge; the player should press gently and slow, not jitter or bounce.
5. **Calm camera follow: replace snap with eased follow (lerp/smoothing)**
    - **Where:** In `draw()` → **2) UPDATE VIEW STATE (CAMERA)**, replace:
        - `cam.x = player.x - width/2; cam.y = player.y - height/2;`
    - **Add/change:**
        - Compute a **camera target**:
            - `cam.tx = player.x - width/2; cam.ty = player.y - height/2;`
        - Ease actual camera toward the target:
            - `cam.x = lerp(cam.x, cam.tx, camSmooth);`
            - `cam.y = lerp(cam.y, cam.ty, camSmooth);`
        - **Tuning note:** `camSmooth` smaller = floatier (try `0.06–0.12`).
    - **Why:** Removes “snap” and creates a reflective, drifting perspective.
    - **Quick test:** Move in a tight circle; camera should lag slightly and feel like it’s gliding on water.
6. **Pacing + stillness (choose B): Auto-drift when the player stops moving**
    - **Where:** In **UPDATE VIEW STATE (CAMERA)**, right after you compute `cam.tx/cam.ty` (before lerping `cam.x/y`).
    - **Add/change:**
        - Detect “stopped” using input and/or velocity (calmer): e.g., `moving = (abs(player.vx)+abs(player.vy) > tiny)` or `(dx!=0 || dy!=0)`.
        - When **not moving**, slowly raise `driftFade` toward 1; when moving, ease it back toward 0.
        - Add a gentle drift offset to the camera target using `noise()` (no sudden direction changes):
            - `driftX = (noise(frameCount*driftSpeed, 0)-0.5) * 2 * driftAmt * driftFade`
            - `driftY = (noise(0, frameCount*driftSpeed)-0.5) * 2 * driftAmt * driftFade`
            - `cam.tx += driftX; cam.ty += driftY;`
        - If **Reduced Motion** is on, force `driftFade = 0`.
    - **Why:** Encourages “slow travel + pause” moments—when you stop, the world continues breathing gently, inviting lingering.
    - **Quick test:** Stop pressing keys. After ~1–2 seconds the camera should begin a barely-noticeable drift. Start moving again; drift should fade out smoothly.

7. **World/camera bounds decision: soft-clamp camera with slight overscroll (no hard stop)**
    - **Decision:** Clamp camera **softly** to the world, but allow a small overscroll margin for calmness.
    - **Where:** In **UPDATE VIEW STATE (CAMERA)**, after applying drift to `cam.tx/cam.ty` and before `lerp()` to `cam.x/y`.
    - **Add/change (math concept):**
        - Define allowed range:
            - `minX = -overscrollPad`, `maxX = WORLD_W - width + overscrollPad` (same for Y).
        - Compute `clampedTx = constrain(cam.tx, minX, maxX)` (and Y).
        - **Softness:** instead of snapping the target, ease the target toward the clamp when out of range:
            - `cam.tx = lerp(cam.tx, clampedTx, clampSoft);` (same for Y)
        - Then do your normal camera follow lerp from Step 5.
    - **Why:** You stay inside the designed world, but edges feel like a gentle “pressure” instead of a wall.
    - **Quick test:** Walk to the world’s edge; the camera should slow and settle smoothly with a tiny, allowable overshoot—no jitter, no abrupt stop.
8. **Add subtle “breathing” motion (gentle zoom) in the WORLD draw transform**
    - **Where:** In `draw()` → **3) DRAW** → inside the world `push()` block, *before* `translate(-cam.x, -cam.y)`.
    - **Add/change:**
        - Compute a slow zoom value: `z = 1 + sin(frameCount * breathRate) * breathAmp`
        - Apply zoom around the screen center (so it feels like the “view” breathes):
            - `translate(width/2, height/2); scale(z); translate(-width/2, -height/2);`
            - then your existing `translate(-cam.x, -cam.y);`
        - If `opt.reducedMotion`, force `z = 1`.
        - **Safe ranges:** `breathAmp = 0.003–0.010` (0.3%–1.0%), `breathRate` tuned to ~8–16 seconds per cycle.
    - **Why:** A tiny, slow breath makes the camera feel contemplative without dizziness.
    - **Quick test:** Stand still and watch edges/landmarks—there should be a barely perceptible inhale/exhale. Toggle Reduced Motion later; it should stop entirely.
9. **Replace harsh grid with calmer world visuals (base layer)**
    - **Where:** In the world draw section (inside `push()` after your transforms), replace:
        - the flat `rect(0,0,WORLD_W,WORLD_H)` + both grid `for` loops.
    - **Add/change:**
        - Draw a soft background using only shapes/colors:

        - Option A (simple): large translucent bands/rectangles with slight alpha variation.
        - Option B (nicer): use `drawingContext.createLinearGradient(...)` for a subtle gradient fill (still GitHub Pages friendly).
    - **Why:** The grid reads as “debug/challenge”; removing it instantly shifts mood toward reflective.
    - **Quick test:** Run and move—camera motion should feel calmer because the world isn’t screaming “measurement.”

10. **Create 3 distinct mood areas across the world + detect which area you’re in**

- **Where:** Add a helper function near bottom: `function getMoodAt(wx, wy) { ... }`
- **Add/change:**
    - Decide zones (simple + readable): split the world into thirds by X using the **camera center** (or player position):
        - `cx = cam.x + width/2` (or `player.x`)
        - left third = “Dawn”, middle = “Day”, right = “Dusk/Night”
    - Each mood sets a small palette + effect settings (particle count, line opacity, landmark color).
    - In the world draw section, call `mood = getMoodAt(cam.x + width/2, cam.y + height/2)` and branch your drawing choices.
- **Why:** Crossing space becomes emotional pacing: you “travel” between feelings.
- **Quick test:** Move horizontally across the big world; you should see clear but gentle shifts in palette/effects at boundaries (no flashing, no abrupt contrast spikes).

11. **Add sparse landmarks + slow “flow” details (reflective motion without gameplay)**

- **Where:** In the world draw section, where your obstacles loop currently is.
- **Add/change:**
    - Keep your obstacle loop concept, but redesign them as calmer landmarks: rounded pools, stones, or lantern-like circles with low alpha.
    - Add one subtle “flow” element per mood:
        - Dawn: faint drifting circles (very low alpha)
        - Day: slow horizontal “current” lines (few, widely spaced)
        - Night: tiny star points that twinkle *very slowly* (sin-based alpha, tiny amplitude)
    - Tie colors/effects to the `mood` from Step 10.
    - If `opt.highContrast`, increase alpha and darken/brighten key elements for readability.
- **Why:** Gives your camera something soothing to watch during pauses, without turning it into an obstacle course.
- **Quick test:** Stop moving in each mood area; you should notice a gentle ambient motion that’s calm and consistent.

12. **Comfort & inclusivity toggles + HUD states (M / H / R)**

- **Where:**

  - Add a `keyPressed()` function (new) for toggles/reset.
  - Update the HUD section (after `pop()`) to display toggle states clearly.
- **Add/change:**
  - `[M] Reduced Motion`: toggles `opt.reducedMotion`. When ON:
    - disable breathing (Step 8) and auto-drift (Step 6), and optionally use a slightly higher `camSmooth` (less floaty, more stable).
  - `[H] High Contrast`: toggles `opt.highContrast`. When ON:
    - choose higher-contrast palettes for moods, add a HUD background panel, and ensure text color strongly contrasts.
  - `[R] Reset`: set `player.x/y = START_X/START_Y`, zero `player.vx/vy`, and set `cam.x/y` and `cam.tx/ty` to the centered start view (so reset feels calm, not a jump).
  - HUD: add lines like:
    - `M: Reduced Motion [ON/OFF]`
    - `H: High Contrast [ON/OFF]`
    - `R: Reset`
- **Why:** Lets more people enjoy the piece comfortably, and makes your intent explicit.
- **Quick test:** Press M/H/R; confirm the HUD updates immediately, motion changes are obvious-but-gentle, and reset always returns to a serene composition.

Parameter tuning guide (calmer ↔ more active)

- **Camera floatiness:** `camSmooth`
  - Calmer/floatier: `0.05–0.10`
  - More responsive (still smooth): `0.12–0.20`
  - Reduced Motion suggestion: `0.18–0.35`
- **Player feel:** `accel`, `friction`, `maxSpeed`
  - Calmer glide: lower `accel`, higher `friction` (closer to 1 is *less* damping), moderate `maxSpeed`
  - More precise: higher `accel`, stronger damping (e.g., `friction` a bit lower), lower `maxSpeed`
  - SHIFT slow-walk multiplier: `0.35–0.60` of normal speed
- **Auto-drift pacing:** `driftAmt`, `driftSpeed`, `driftFade` easing rate
  - Calmer: `driftAmt` ~ `6–18 px`, `driftSpeed` very small, slow fade-in/out
  - More noticeable (still gentle): `driftAmt` ~ `20–35 px` (avoid higher to prevent dizziness)
- **Breathing (keep subtle):** `breathAmp`, `breathRate`
  - Safe comfort: `breathAmp 0.003–0.010`, cycle ~ `8–16s`
  - If anyone feels woozy: reduce `breathAmp` first, then turn off via Reduced Motion

- **Soft camera bounds:** `overscrollPad`, `clampSoft`
    - Calmer edges: `overscrollPad 30–80`, `clampSoft 0.05–0.12`
    - Tighter framing: smaller pad, slightly higher clampSoft (but avoid hard snapping)
- **Mood readability:** palette contrast + effect density
    - Calmer: fewer particles/lines, lower alpha
    - High Contrast mode: increase alpha and simplify background so HUD/world edges are clearly readable

Implement each step into the base code, after double check all requirements and go through a check list making sure everything was implemented.

/*

Week 5 — Reflective / Meditative Camera World (GitHub Pages friendly)

Based on your Week 5 Example 1 base:

- player in WORLD coords

- cam in WORLD coords

- translate(-cam.x, -cam.y) to draw world

- HUD drawn in screen space after pop()

Upgrades implemented:

1) Calm camera follow (lerp)

2) Pacing + stillness (auto-drift when stopped)

3) Gentle player movement (accel/friction + SHIFT slow walk + consistent diagonals)

4) Reflective visuals (no harsh grid + 3 mood areas + subtle ambient details)

5) Subtle breathing (tiny zoom)

6) Comfort toggles in HUD (M/H/R)

7) Soft camera bounds (smooth clamp + slight overscroll)

*/

```
let player = { x: 300, y: 300, s: 3, vx: 0, vy: 0 }; // WORLD
let cam = { x: 0, y: 0, tx: 0, ty: 0 }; // camera top-left in WORLD

// World size
const WORLD_W = 2400;
const WORLD_H = 1600;

// Canvas / viewport size
const VIEW_W = 800;
const VIEW_H = 480;

// ---- Motion + feel tuning ----
const MOVE = {
 accel: 0.38,
 friction: 0.90,       // higher = more glide (0.88–0.94 good range)
 maxSpeed: 3.0,
 slowMult: 0.50,       // SHIFT slow walk multiplier
 stopEps: 0.05         // below this, snap velocity to 0
};

const CAMERA = {
 smooth: 0.09,         // calm follow smoothing (0.06–0.12 floaty)
 smoothRM: 0.22,       // reduced motion easing (more stable, less float)
 overscrollPad: 60,    // allow slight overscroll beyond world edges
 clampSoft: 0.08       // soften target when out of bounds (no hard stop)
};

// Auto-drift when stopped (pacing + stillness)
const DRIFT = {
 amt: 18,              // pixels (6–25 subtle)
 speed: 0.0022,        // noise speed (very small)
 fadeIn: 0.02,
 fadeOut: 0.06
};
let driftFade = 0;

// Breathing zoom (subtle)
const BREATH = {
 amp: 0.006,           // 0.003–0.010 safe
 rate: (Math.PI * 2) / (60 * 12) // ~12s per cycle at ~60fps
};

// Comfort / inclusivity toggles
let opt = { reducedMotion: false, highContrast: false };

// Calm starting location
const START_X = 300;
const START_Y = 300;

// World “mood” content (pre-generated for stability)
let landmarks = []; // pools/stones
let stars = [];     // dusk zone twinkles
let motes = [];     // dawn drifting circles

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

 // Stable layout (not random every reload)
 randomSeed(5);
 noiseSeed(5);

 // Pre-generate sparse landmarks across the world
 for (let i = 0; i < 52; i++) {
   landmarks.push({
     x: random(80, WORLD_W - 80),
     y: random(80, WORLD_H - 80),
     r: random(20, 70),
     t: random(1000)
```

```
  });
}

// Gentle motes for left (dawn) zone
for (let i = 0; i < 90; i++) {
  motes.push({
    x: random(0, WORLD_W / 3),
    y: random(0, WORLD_H),
    r: random(4, 12),
    sp: random(0.15, 0.55),
    n: random(1000)
  });
}

// Stars for right (dusk) zone
for (let i = 0; i < 140; i++) {
  stars.push({
    x: random((WORLD_W * 2) / 3, WORLD_W),
    y: random(0, WORLD_H),
    a: random(0.05, 0.20),
    p: random(0, Math.PI * 2),
    r: random(1.0, 2.0)
  });
}
```

```
 resetToStart();
}

function draw() {
 // ---------- 1) UPDATE GAME STATE (WORLD) ----------
 // Input becomes a direction vector (dx, dy)
 const dx =
   (keyIsDown(RIGHT_ARROW) || keyIsDown(68)) -
   (keyIsDown(LEFT_ARROW) || keyIsDown(65));

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

 // Normalize with Euclidean length so diagonal speed is consistent
 const mag = Math.sqrt(dx * dx + dy * dy);
 const nx = mag > 0 ? dx / mag : 0;
 const ny = mag > 0 ? dy / mag : 0;

 // SHIFT = slow-walk precision
 const slow = keyIsDown(SHIFT);
 const accel = slow ? MOVE.accel * 0.65 : MOVE.accel;
 const maxSp = slow ? MOVE.maxSpeed * MOVE.slowMult : MOVE.maxSpeed;
```

```
// Acceleration into velocity (gentle start/stop)

player.vx += nx * accel;

player.vy += ny * accel;


// Clamp to max speed (prevents runaway skating)

const sp = Math.sqrt(player.vx * player.vx + player.vy * player.vy);

if (sp > maxSp) {

  const k = maxSp / sp;

  player.vx *= k;

  player.vy *= k;

}


// Friction / damping (gentle deceleration)

player.vx *= MOVE.friction;

player.vy *= MOVE.friction;


// Snap tiny velocities to 0 for calm stillness

if (Math.abs(player.vx) < MOVE.stopEps) player.vx = 0;

if (Math.abs(player.vy) < MOVE.stopEps) player.vy = 0;


// Apply velocity to position (WORLD space)

player.x += player.vx;

player.y += player.vy;
```

```
// Optional soft player bounds (keeps you inside designed mood areas)
// (soft: clamp position + damp velocity at edges)
const px0 = player.x, py0 = player.y;
player.x = constrain(player.x, 0, WORLD_W);
player.y = constrain(player.y, 0, WORLD_H);
if (player.x !== px0) player.vx *= 0.2;
if (player.y !== py0) player.vy *= 0.2;

// ---------- 2) UPDATE VIEW STATE (CAMERA) ----------
// Base target: center camera on player (target, not snap)
cam.tx = player.x - width / 2;
cam.ty = player.y - height / 2;

// Pacing + stillness: auto-drift when player stops moving
const moving = (Math.abs(player.vx) + Math.abs(player.vy)) > 0;
if (opt.reducedMotion) {
  driftFade = 0;
} else {
  driftFade = lerp(driftFade, moving ? 0 : 1, moving ? DRIFT.fadeOut : DRIFT.fadeIn);
}

if (driftFade > 0.001) {
  const t = frameCount * DRIFT.speed;
  const driftX = (noise(t, 0.0) - 0.5) * 2 * DRIFT.amt * driftFade;
  const driftY = (noise(0.0, t) - 0.5) * 2 * DRIFT.amt * driftFade;
  cam.tx += driftX;
  cam.ty += driftY;
}

// World/camera bounds: soft-clamp target with slight overscroll (no hard stop)
const minX = -CAMERA.overscrollPad;
const minY = -CAMERA.overscrollPad;
const maxX = WORLD_W - width + CAMERA.overscrollPad;
const maxY = WORLD_H - height + CAMERA.overscrollPad;

const clampedTx = constrain(cam.tx, minX, maxX);
const clampedTy = constrain(cam.ty, minY, maxY);

// Ease target toward clamp so edges feel gentle
cam.tx = lerp(cam.tx, clampedTx, CAMERA.clampSoft);
cam.ty = lerp(cam.ty, clampedTy, CAMERA.clampSoft);

// Calm camera follow (lerp)
const camSmooth = opt.reducedMotion ? CAMERA.smoothRM : CAMERA.smooth;
cam.x = lerp(cam.x, cam.tx, camSmooth);
cam.y = lerp(cam.y, cam.ty, camSmooth);
```

```
// ---------- 3) DRAW ----------

// Screen-space clear (outside world)

background(opt.highContrast ? 10 : 22);


// Draw the WORLD (scrolling layer) in world space

push();


// Subtle breathing zoom (disabled in Reduced Motion)

let z = 1;

if (!opt.reducedMotion) {

  z = 1 + Math.sin(frameCount * BREATH.rate) * BREATH.amp;

}

translate(width / 2, height / 2);

scale(z);

translate(-width / 2, -height / 2);


// Your original camera translate (world scrolling)

translate(-cam.x, -cam.y);


// Draw mood areas + reflective visuals (no harsh grid)

drawWorld();


// Player (in world space)

drawPlayer();
```

```
 pop();


 // HUD (screen space): drawn AFTER pop(), so it does not move with camera
 drawHUD();
}


function drawWorld() {
 // Fixed 3-zone world: left/middle/right thirds
 const third = WORLD_W / 3;


 // Draw three large mood bands (soft gradients)
 drawMoodBand(0, 0, third, WORLD_H, 0);
 drawMoodBand(third, 0, third, WORLD_H, 1);
 drawMoodBand(third * 2, 0, third, WORLD_H, 2);


 // Sparse landmarks (pools/stones) with zone-tinted colors
 noStroke();
 for (let i = 0; i < landmarks.length; i++) {
   const L = landmarks[i];
   const zone = getZoneAt(L.x);
   const pal = getPalette(zone, opt.highContrast);


   // Very slow shimmer (non-flashy)
   const shimmer = 0.08 + 0.05 * Math.sin((frameCount * 0.006) + L.t);

   fill(pal.landA[0], pal.landA[1], pal.landA[2], pal.landA[3]);
   ellipse(L.x, L.y, L.r * 2.2, L.r * 1.7);

   fill(pal.landB[0], pal.landB[1], pal.landB[2], pal.landB[3] * (0.8 + shimmer));
   ellipse(L.x + L.r * 0.15, L.y - L.r * 0.1, L.r * 1.3, L.r * 0.9);
  }

 // Subtle ambient details per zone (calm, no flashing)
 drawAmbientDetails();
}

function drawMoodBand(x, y, w, h, zone) {
 const pal = getPalette(zone, opt.highContrast);
 const ctx = drawingContext;

 // Gradient fill (GitHub Pages friendly; no images)
 ctx.save();
 const grad = ctx.createLinearGradient(0, y, 0, y + h);
 grad.addColorStop(0, pal.bgTop);
 grad.addColorStop(1, pal.bgBot);
 ctx.fillStyle = grad;
 ctx.fillRect(x, y, w, h);
 ctx.restore();


 // Gentle overlay to reduce “flatness”

 noStroke();

 fill(pal.haze[0], pal.haze[1], pal.haze[2], pal.haze[3]);

 rect(x, y, w, h);


 // Optional high-contrast boundary hints (very subtle, not “grid”)

 if (opt.highContrast) {

   stroke(255, 255, 255, 18);

   line(x, 0, x, WORLD_H);

   noStroke();

 }

}


function drawAmbientDetails() {

 const third = WORLD_W / 3;


 // Dawn zone: drifting motes (left third)

 if (!opt.highContrast) {

   noStroke();

   fill(255, 255, 255, 22);

 } else {

   noStroke();
```

```
  fill(255, 255, 255, 42);

 }


 for (let i = 0; i < motes.length; i++) {

  const m = motes[i];

  // Slow vertical drift + slight sideways wander

  const t = frameCount * 0.003 + m.n;

  const ox = (noise(t, 1.3) - 0.5) * 14;

  const oy = (frameCount * m.sp) % (WORLD_H + 80);

  const yy = (m.y + oy) - 40;

  ellipse(m.x + ox, yy, m.r * 2, m.r * 2);

 }


 // Middle zone: slow “current” lines (wide spacing)

 const midX0 = third;

 const midX1 = third * 2;

 stroke(opt.highContrast ? 255 : 255, opt.highContrast ? 255 : 255, opt.highContrast ? 255 :
255, opt.highContrast ? 18 : 12);

 strokeWeight(1);

 for (let y = 120; y < WORLD_H; y += 180) {

  const wave = Math.sin((frameCount * 0.004) + y * 0.02) * 18;

  line(midX0 + 60, y + wave, midX1 - 60, y - wave);

 }

 noStroke();
```

```
  // Dusk zone: slow twinkle stars (right third)
  noStroke();
  for (let i = 0; i < stars.length; i++) {
    const s = stars[i];
    // Twinkle very slowly + tiny amplitude (no flash)
    const tw = 0.03 * Math.sin(frameCount * 0.01 + s.p);
    const a = constrain(s.a + tw, 0.03, 0.22);
    fill(255, 255, 255, opt.highContrast ? a * 255 : a * 180);
    circle(s.x, s.y, s.r);
  }

  // Subtle vignette “edge softening” for reflective feel
  // (drawn in world space so it scrolls; gentle, not dark)
  const cx = cam.x + width / 2;
  const cy = cam.y + height / 2;
  noStroke();
  fill(0, 0, 0, opt.highContrast ? 18 : 10);
  // Four big soft corners around current view area
  circle(cx - width * 0.55, cy - height * 0.55, 520);
  circle(cx + width * 0.55, cy - height * 0.55, 520);
  circle(cx - width * 0.55, cy + height * 0.55, 520);
  circle(cx + width * 0.55, cy + height * 0.55, 520);
}
```

```
function drawPlayer() {
 const zone = getZoneAt(player.x);
 const pal = getPalette(zone, opt.highContrast);

 // Soft shadow under player
 noStroke();
 fill(0, 0, 0, opt.highContrast ? 70 : 30);
 ellipse(player.x + 2, player.y + 6, 28, 18);

 // Player body (calm, readable)
 fill(pal.player[0], pal.player[1], pal.player[2], 255);
 rect(player.x - 12, player.y - 12, 24, 24, 6);

 // Gentle “center” dot (helps focus, non-gamey)
 fill(255, 255, 255, opt.highContrast ? 220 : 160);
 circle(player.x, player.y, 4);
}

function drawHUD() {
 const pad = 10;

 // HUD panel for readability
 noStroke();
 if (opt.highContrast) {
   fill(0, 0, 0, 180);
 } else {
   fill(255, 255, 255, 140);
 }
 rect(pad, pad, 780, 86, 10);

 // HUD text
 if (opt.highContrast) fill(255);
 else fill(20);

 const cx = (cam.x + width / 2) | 0;
 const cy = (cam.y + height / 2) | 0;
 const zone = getZoneAt(cx);
 const zoneName = ["Dawn", "Meadow", "Dusk"][zone];

 text("Week 5 — Reflective camera world (calm travel + pause). Move: WASD/Arrows. SHIFT: slow walk.", pad + 10, pad + 22);
 text(`[M] Reduced Motion: ${opt.reducedMotion ? "ON" : "OFF"}   [H] High Contrast: ${opt.highContrast ? "ON" : "OFF"}   [R] Reset`, pad + 10, pad + 42);
 text(`Mood: ${zoneName}   Player(world): ${(player.x | 0)}, ${(player.y | 0)}   CamCenter(world): ${cx}, ${cy}`, pad + 10, pad + 62);
}

function keyPressed() {
 if (key === "m" || key === "M") {
   opt.reducedMotion = !opt.reducedMotion;
   // When enabling reduced motion, also quiet drift immediately
   if (opt.reducedMotion) driftFade = 0;
 }
 if (key === "h" || key === "H") {
   opt.highContrast = !opt.highContrast;
 }
 if (key === "r" || key === "R") {
   resetToStart();
 }
}

function resetToStart() {
 player.x = START_X;
 player.y = START_Y;
 player.vx = 0;
 player.vy = 0;

 cam.x = player.x - width / 2;
 cam.y = player.y - height / 2;
 cam.tx = cam.x;
 cam.ty = cam.y;

 driftFade = 0;
}
```

```
// Zone detection (3 distinct mood areas across the large world)

function getZoneAt(worldX) {

 const third = WORLD_W / 3;

 if (worldX < third) return 0;

 if (worldX < third * 2) return 1;

 return 2;

}


// Palettes (zone + high-contrast variants)

// bgTop/bgBot are CSS strings for Canvas gradients.

// landA/landB/player/haze are RGBA arrays for p5 fills.

function getPalette(zone, highContrast) {

 if (!highContrast) {

   if (zone === 0) {

     // Dawn (cool, airy)

     return {

       bgTop: "#E6F0FF",

       bgBot: "#CFE2FF",

       haze: [255, 255, 255, 22],

       landA: [160, 198, 220, 70],

       landB: [210, 235, 255, 65],

       player: [56, 120, 255]

     };
```

```
  }
  if (zone === 1) {
    // Meadow (warm, grounded)
    return {
      bgTop: "#F0F6E8",
      bgBot: "#DDEBD0",
      haze: [255, 255, 255, 18],
      landA: [160, 200, 170, 60],
      landB: [230, 245, 235, 55],
      player: [40, 140, 120]
    };
  }
  // Dusk (deep, quiet)
  return {
    bgTop: "#1F2A44",
    bgBot: "#131A2C",
    haze: [0, 0, 0, 16],
    landA: [90, 110, 160, 55],
    landB: [160, 170, 210, 35],
    player: [190, 210, 255]
  };
}

// High contrast variants (stronger separation + HUD already dark)
```

```
if (zone === 0) {
  return {
    bgTop: "#FFFFFF",
    bgBot: "#D7E6FF",
    haze: [255, 255, 255, 10],
    landA: [40, 90, 160, 110],
    landB: [220, 240, 255, 90],
    player: [0, 120, 255]
  };
}
if (zone === 1) {
  return {
    bgTop: "#F7FFE8",
    bgBot: "#CFE9A8",
    haze: [255, 255, 255, 10],
    landA: [20, 110, 60, 120],
    landB: [235, 255, 235, 90],
    player: [0, 180, 120]
  };
}
return {
  bgTop: "#05070F",
  bgBot: "#000000",
  haze: [0, 0, 0, 6],
  landA: [140, 170, 255, 120],

  landB: [220, 230, 255, 90],

  player: [255, 255, 255]

 };

}
```

Requirements checklist (double-checked)

1. **Calm camera follow (no snap)** ✅

- Uses `cam.tx/cam.ty` target + `cam.x/y = lerp(cam.x/y, cam.t*, camSmooth)`.

Tuning for floaty motion ✅

- `CAMERA.smooth` controls floatiness; smaller = floatier.

2. **Pacing + stillness** ✅ (Approach **B: auto-drift when stopped**)

- Detects `moving` from player velocity; fades `driftFade` in/out; adds noise-based drift to camera target.

Meditative pacing explanation (in behavior) ✅

- When you pause, the camera gently continues—inviting lingering without challenge.

3. **Gentle player movement** ✅

- Accel → velocity → friction; SHIFT slow walk; diagonal consistency via Euclidean normalization; max speed clamp.

4. **World visuals reflective (no harsh grid)** ✅

- Grid removed. World is three soft gradient bands + sparse landmarks + subtle ambient details.

3 distinct mood areas + detection ✅

- World split into thirds by X; `getZoneAt(worldX)` selects palettes/effects; HUD shows mood name.

5. **Subtle breathing motion** ✅

- Tiny zoom around screen center (`BREATH.amp` in safe range) and disabled under Reduced Motion.

6. **Comfort & inclusivity toggles (HUD readable)** ✅

- `[M]` Reduced Motion: disables drift + breathing and uses `CAMERA.smoothRM`.
- `[H]` High Contrast: boosts palettes + dark HUD panel with bright text.
- `[R]` Reset: returns player/camera to calm start; zeroes velocity; resets drift.

Flags storage + HUD state display ✅

- `opt = { reducedMotion, highContrast }` global; HUD prints ON/OFF.

7. **World/camera bounds decision ✅**

- Soft clamp camera target with overscroll margin; clamp easing prevents hard stops.

No external images / GitHub Pages friendly ✅

- Only shapes, colors, gradients; no assets.

Calm, not challenging; no flashing or shake ✅

- No shakes; twinkle is slow + low amplitude; drift + breathing are subtle and toggleable.