Entry Header

Name:

Aswina Sivaloganathan

Role:

Programmer

Short description:

- Built a simple, reflective camera experience in p5.js set in a dark ocean.
- The user controls a fish with the arrow keys, and the camera follows the fish through a world larger than the screen.
- Jellyfish are placed throughout the world to create small hidden elements as the player explores.

Small decisions or changes made:

- A background gradient was added to show a "dark ocean".
- Added a fish controlled by arrow keys
- Added simple jellyfish around the screen to show hidden elements. Tailored the jellyfish's appearance based on my likings.

Evidence:

Change #1 - Code comment of the jellyfish structure. I picked the size, colour and transparency.

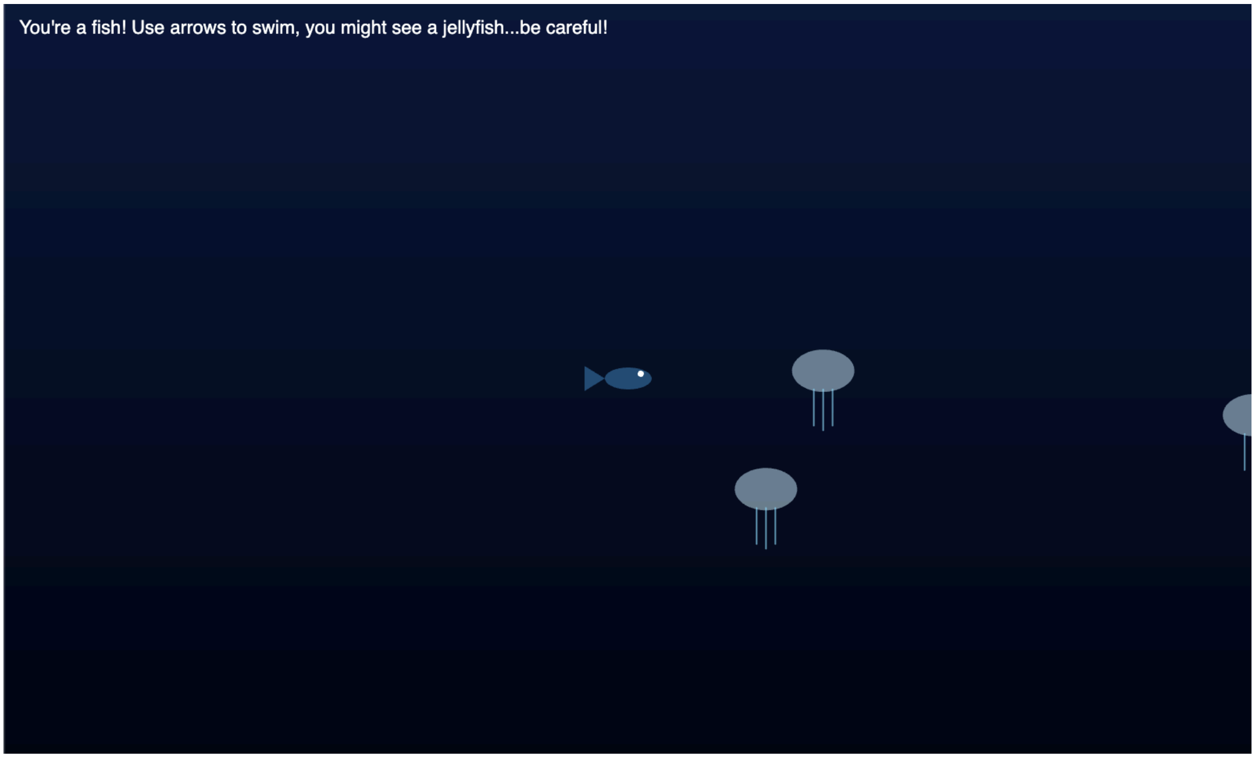
```
drawJellies() {
   for (let j of this.jellies) {
     push();
     translate(j.x, j.y);

     const floatY = sin(frameCount * 0.02 + j.x) * 8;

     fill(190, 240, 255, 130);
     noStroke();
     ellipse(0, floatY, 40, 27);

     stroke(120, 220, 255, 180);
     line(-6, floatY + 12, -6, floatY + 35);
     line(0, floatY + 12, 0, floatY + 38);
     line(6, floatY + 12, 6, floatY + 35);
```

Final Result:



You're a fish! Use arrows to swim, you might see a jellyfish...be careful!

GenAI Documentation

GenAI tools were used for this task.

Date Used:

February 22, 2026

Tools, Resources, Inputs Used

- ChatGPT 5.2
- GBDA 302 Week 5 Example Code

Purpose of use:

I used ChatGPT to help expand and refine an idea I had already had thought of for the side quest. After deciding on the overall concept, I used the chat to help with implementing code components needed to bring that idea to life. This included help with structuring the ocean gradient background, fish movement, and placing simple jellyfish elements within the screen. The tool supported the technical implementation, while the overall direction and scope of the project was my own.

Human decision point(s):

- I iterated and refined through each suggestion. Making decisions on my own.
- I chose to significantly simplify the generated suggestions by removing extra features such as auto-drifting cameras, reshuffling objects, and particle systems.
- I determined the scope of hidden objects (jellyfish only) to maintain simplicity and avoid overcomplicating the project.
- I manually adjusted some code to tailor it to my liking (jellyfish appearance) as well as some fixing (centering, positioning)..

Integrity and verification note:

All generated code was reviewed, tested, and modified by me. I verified functionality through manual testing and adjusted code where necessary. The final implementation reflects my design decisions and simplifications.

Limitations or misfires:

A lot of what GenAI initially suggested was way too much. It had overly complex systems (auto-drift camera, reshuffling objects), which didn't really align with my goal of keeping the project simple. Also, initial camera logic resulted in the fish spawning off-screen, which required debugging and some manual adjustment.

Appendix - Transcript

I want to create a reflective or meditative camera experience that scrolls through a world larger than the screen. Also using pacing and motion to evoke emotion. I want it to also hide small interactive symbols or objects for the camera to "discover objects". I was thinking a dark theme, like in a dark ocean maybe? could you help me brainstorm ways to do so in for my p5.js code.

Yes — and p5.js is *perfect* for this vibe. Here are a bunch of solid directions you can pick from (and mix), plus concrete ways to build it in code.

Core concept: "slow camera drifting through a huge dark ocean"

Think of your sketch as 2 coordinate spaces:

- **World space**: super large (like 8000×3000) with objects placed around.
- **Screen space**: the canvas (like 800×600) that shows only a window into the world.
  You move a **camera** (camX, camY) slowly through the world, and draw everything

offset by `-camX, -camY`.

Mood + motion ideas (to evoke emotion)

1) Slow drift + “breathing” pacing

- Camera movement is subtle and continuous (low speed).
- Add a gentle “breath” oscillation to feel alive:
    - `camY += sin(frameCount * 0.002) * 0.3`
    - `camX += cos(frameCount * 0.0015) * 0.25`
- Add *slow acceleration/deceleration* so it doesn’t feel robotic:
    - Use noise to control speed: `speed = map(noise(t), 0, 1, 0.05, 0.4)`

2) Parallax layers (instant depth + calm)

Draw multiple layers moving at different speeds:

- Far fog / faint particles: moves at `0.2x` camera offset
- Mid particles / plankton: `0.5x`
- Foreground silhouettes (kelp, rocks): `0.9x–1.1x`
  This makes the world feel big and dreamy.

3) Light cone / flashlight effect

In a dark ocean, you can “reveal” the world with a soft circular gradient around the center (or mouse).

- Everything outside is dim / tinted.
- When you “discover objects,” they appear in the light.

4) Ambient particles that react softly

Tiny drifting dots that move slowly + wobble with noise.
If the camera gets close to something "special," particles subtly swirl toward it.

"Discover objects" mechanics (hidden symbols)

A) Proximity reveal (most meditative)

Objects are invisible (or barely visible) until the camera gets close.

- If `dist(camCenter, object) < revealRadius`, fade it in.
- This feels like "finding" instead of "clicking."

Extra emotion: when discovered, play a tiny animation:

- pulse ring
- gentle glow
- small "bubble" burst

B) Symbol scavenger but subtle

Hide tiny glyphs:

- a rune
- a small bioluminescent fish
- a floating bottle
- a star-like speck that moves differently than particles
  They're placed so the camera might pass near them, but not guaranteed.

C) Gated discovery

Objects reveal only when:

- camera is moving slowly (encourages calm)
- OR user holds a key (like “hold space to listen”)
- OR user stops drifting for 2 seconds (stillness reveals)

This makes pacing *part of the interaction*.

Visual theme ideas for a dark ocean

Palette vibes (simple + strong)

- Background: deep navy to near-black gradient
- Accents: muted teal, cyan, soft violet (bioluminescence)
- Occasional warm accent (dim amber) = “rare discovery”

World elements to draw (cheap but effective)

- **Kelp forests**: long bezier lines waving with noise
- **Rock silhouettes**: big simple shapes with soft edges
- **Silt/fog**: translucent blobs or rectangles with noise alpha
- **Jellyfish**: soft bell + trailing lines, slow float
- **Sunken ruins**: repeating pillars or arches, barely visible until close

Camera design options (pick one)

Option 1: Auto-scroll “guided meditation”

The camera follows a slow path defined by noise:

- It always moves, but gently.

- User can “nudge” with mouse slightly, but can’t fully control (calm vibe).

Option 2: Slow user pan with inertia

Arrow keys / WASD moves the camera, but with smoothing:

- target velocity changes
- camera eases into it (feels floaty)

Option 3: “Current” zones

The ocean has invisible currents that carry the camera in certain directions.

- When you enter a zone, drift changes.
- Discovery objects can be near current boundaries.

How to structure your p5.js code (clean mental model)

Data you’ll want:

- `worldW, worldH`
- `camX, camY`
- `layers[]` (each layer has particles or shapes + parallax factor)
- `discoveries[]` (each has x,y,type,found,alpha)

Drawing loop:

1. Update camera (drift + easing)
2. Draw background gradient
3. For each parallax layer: draw using `(x - camX * factor)`

4. Draw discoveries (fade in if near camera center)
5. Draw light cone / vignette
6. Optional: minimal UI (like “3 found”)

Specific “discoverable object” ideas (ocean-themed + small)

- **Bioluminescent symbols** etched on rocks (glow increases when close)
- **Tiny anglerfish** that darts away if you approach too quickly
- **Message-in-a-bottle** that slowly spins
- **Constellation of plankton** that forms a shape only when you’re near
- **A bell chime moment**: when found, the whole screen slightly brightens for 2 seconds

---

A strong starter combo (if you want the cleanest, nicest experience)

If you choose one direction that will *look good fast*:

- Auto camera drift with “breathing” motion
- 3 parallax particle layers
- Discoveries that fade in by proximity
- Soft flashlight vignette centered on screen
- 8–15 hidden symbols across a huge map

That’s enough to feel *intentional and meditative*, without needing complex art.

If you want, I can also sketch you a simple p5.js starter structure (camera + world coords + discovery fade-in + vignette). Just tell me: do you want **auto-drift** or **WASD + inertia**?

I'm gonna upload the files, please help me implement these ideas! this is sketch.js - /* Week 5 — Example 5: Side-Scroller Platformer with JSON Levels + Modular Camera Course: GBDA302 | Instructors: Dr. Karen Cochrane & David Han Date: Feb. 12, 2026 Move: WASD/Arrows | Jump: Space Learning goals: - Build a side-scrolling platformer using modular game systems - Load complete level definitions from external JSON (LevelLoader + levels.json) - Separate responsibilities across classes (Player, Platform, Camera, World) - Implement gravity, jumping, and collision with platforms - Use a dedicated Camera2D class for smooth horizontal tracking - Support multiple levels and easy tuning through data files - Explore scalable project architecture for larger games */ const VIEW_W = 800; const VIEW_H = 480; let allLevelsData; let levelIndex = 0; let level; let player; let cam; function preload() { allLevelsData = loadJSON("levels.json"); // levels.json beside index.html [web:122] } function setup() { createCanvas(VIEW_W, VIEW_H); textFont("sans-serif"); textSize(14); cam = new Camera2D(width, height); loadLevel(levelIndex); } function loadLevel(i) { level = LevelLoader.fromLevelsJson(allLevelsData, i); player = new BlobPlayer(); player.spawnFromLevel(level); cam.x = player.x - width / 2; cam.y = 0; cam.clampToWorld(level.w, level.h); } function draw() { // --- game state --- player.update(level); // Fall death → respawn if (player.y - player.r > level.deathY) { loadLevel(levelIndex); return; } // --- view state (data-driven smoothing) --- cam.followSideScrollerX(player.x, level.camLerp); cam.y = 0; cam.clampToWorld(level.w, level.h); // --- draw --- cam.begin(); level.drawWorld(); player.draw(level.theme.blob); cam.end(); // HUD fill(0); noStroke(); text(level.name + " (Example 5)", 10, 18); text("A/D or ←/→ move • Space/W/↑ jump • Fall = respawn", 10, 36); text("camLerp(JSON): " + level.camLerp + " world.w: " + level.w, 10, 54); text("cam: " + cam.x + ", " + cam.y, 10, 90); const p0 = level.platforms[0]; text(`p0: x=${p0.x} y=${p0.y} w=${p0.w} h=${p0.h}`, 10, 108); text("platforms: " + level.platforms.length + " start: " + level.start.x + "," + level.start.y, 10, 72,); } function keyPressed() { if (key === " " || key === "W" || key === "w" || keyCode === UP_ARROW) { player.tryJump(); } if (key === "r" || key === "R") loadLevel(levelIndex); } this is worldLevel.js - class WorldLevel { constructor(levelJson) { this.name = levelJson.name ?? "Level"; this.theme = Object.assign({ bg: "#F0F0F0", platform: "#C8C8C8", blob: "#1478FF" }, levelJson.theme ?? {},); // Physics knobs this.gravity = levelJson.gravity ?? 0.65; this.jumpV = levelJson.jumpV ?? -11.0; // Camera knob (data-driven view state) this.camLerp = levelJson.camera?.lerp ?? 0.12; // World size + death line this.w = levelJson.world?.w ?? 2400; this.h = levelJson.world?.h ?? 360; this.deathY = levelJson.world?.deathY ?? this.h + 200; // Start this.start = Object.assign({ x: 80, y: 220, r: 26 }, levelJson.start ?? {}); // Platforms this.platforms = (levelJson.platforms ?? []).map((p) => new Platform(p.x, p.y, p.w, p.h),); } drawWorld() { background(this.theme.bg); push(); rectMode(CORNER); // critical: undo any global rectMode(CENTER) [web:230] noStroke(); fill(this.theme.platform); for (const p of this.platforms) rect(p.x, p.y, p.w, p.h); // x,y = top-left [web:234] pop(); } }and this is Cam2d.js - class Camera2D { constructor(viewW, viewH) { this.viewW = viewW; this.viewH = viewH; this.x = 0; this.y = 0; } followSideScrollerX(targetX, lerpAmt) { const desired = targetX - this.viewW / 2; this.x = lerp(this.x, desired, lerpAmt); } clampToWorld(worldW, worldH) { const maxX = max(0, worldW - this.viewW); const maxY = max(0, worldH - this.viewH); this.x = constrain(this.x, 0, maxX); this.y = constrain(this.y, 0, maxY); } begin() { push(); translate(-this.x, -this.y); } end() { pop(); } }

Yep — easiest place to do this is **`WorldLevel.drawWorld()`**, since that's where your background is currently drawn.

One important detail: you call `level.drawWorld()` **after `cam.begin()`**, which applies a `translate(-cam.x, -cam.y)`. So if we draw a gradient using lines *without resetting transforms*, the gradient will "slide" with the camera (looks wrong).
Fix: **reset the matrix for the gradient**, then draw world objects normally.

✅ Replace your `WorldLevel.js` with this (only changes are theme defaults + gradient background)

1) Replace `sketch.js` (remove player + platforms gameplay)

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

let allLevelsData;
let levelIndex = 0;

let level;
let cam;

function preload() {
  allLevelsData = loadJSON("levels.json");
}

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

  cam = new Camera2D(width, height);
  loadLevel(levelIndex);
}

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

  cam.x = 0;
  cam.y = 0;
  cam.clampToWorld(level.w, level.h);
  level.spawnOceanObjects();
}

function draw() {
  cam.updateDrift(level.w, level.h);

  cam.begin();
  level.drawWorld(cam);
  cam.end();

  fill(255);
  noStroke();
  text(level.name + " (Ocean Drift)", 10, 18);
  text("Press R to reshuffle fish/jelly", 10, 36);
}

function keyPressed() {
  if (key === "r" || key === "R") level.spawnOceanObjects();
}
```

✅ This keeps: level loading + camera system
❌ Removes: BlobPlayer, gravity, jumping, platforms/death logic

2) Replace `Camera2D.js` (add meditative drift)

```
class Camera2D {
  constructor(viewW, viewH) {
    this.viewW = viewW;
    this.viewH = viewH;
    this.x = 0;
    this.y = 0;

    this.t = random(1000);
    this.speedT = random(1000);
  }

  updateDrift(worldW, worldH) {
    const maxX = max(0, worldW - this.viewW);
    const maxY = max(0, worldH - this.viewH);

    const baseSpeed = map(noise(this.speedT), 0, 1, 0.25, 0.8);

    this.t += 0.002 * baseSpeed;
    this.speedT += 0.003;
    const nx = noise(this.t);
    const ny = noise(this.t + 500);

    const desiredX = nx * maxX;
    const desiredY = ny * maxY;

    this.x = lerp(this.x, desiredX, 0.02);
    this.y = lerp(this.y, desiredY, 0.02);

    this.x += cos(frameCount * 0.0015) * 0.15;
    this.y += sin(frameCount * 0.0020) * 0.20;

    this.clampToWorld(worldW, worldH);
  }

  clampToWorld(worldW, worldH) {
    const maxX = max(0, worldW - this.viewW);
    const maxY = max(0, worldH - this.viewH);
    this.x = constrain(this.x, 0, maxX);
    this.y = constrain(this.y, 0, maxY);
  }

  begin() {
    push();
    translate(-this.x, -this.y);
  }
  end() {
    pop();
  }
}
```

3) Replace `WorldLevel.js` (remove platforms + add fish/jelly discoveries)

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

    this.theme = Object.assign(
      {
        bgTop: "#07162a",
        bgBot: "#020611",
        fish: "#0f3a55",
        glow: "#7fd6ff",
      },
      levelJson.theme ?? {},
    );

    this.w = levelJson.world?.w ?? 2400;
    this.h = levelJson.world?.h ?? 1200;

    this.fish = [];
    this.jellies = [];
    this.particles = [];
  }

  spawnOceanObjects() {
    this.fish = [];
    this.jellies = [];
    this.particles = [];

    for (let i = 0; i < 70; i++) {
      this.particles.push({
        x: random(this.w),
        y: random(this.h),
        s: random(1, 2),
        t: random(1000),
      });
    }

    for (let i = 0; i < 12; i++) {
      this.jellies.push({
        x: random(120, this.w - 120),
        y: random(120, this.h - 120),
        a: 0,
      });
    }

    for (let i = 0; i < 14; i++) {
      this.fish.push({
        x: random(120, this.w - 120),
        y: random(120, this.h - 120),
        a: 0,
        dir: random() < 0.5 ? -1 : 1,
      });
    }
  }
```

```
drawOceanGradient() {
  const topC = color(this.theme.bgTop);
  const botC = color(this.theme.bgBot);

  for (let y = 0; y < height; y++) {
    const t = y / height;
    stroke(lerpColor(topC, botC, t));
    line(0, y, width, y);
  }
}

drawWorld(cam) {
  push();
  resetMatrix();
  this.drawOceanGradient();
  pop();

  this.drawParticles(cam);
  this.drawFish(cam);
  this.drawJellies(cam);
}

drawParticles(cam) {
  noStroke();
  fill(255, 18);

  for (const p of this.particles) {
    p.t += 0.01;
    const dy = sin(p.t) * 0.2;
    const px = p.x - cam.x * 0.5;
    const py = p.y + dy - cam.y * 0.5;

    circle(px, py, p.s);
  }
}

drawFish(cam) {
  const cx = cam.x + width / 2;
  const cy = cam.y + height / 2;

  for (const f of this.fish) {
    const d = dist(cx, cy, f.x, f.y);
    const targetA = d < 180 ? 180 : 0;
    f.a = lerp(f.a, targetA, 0.03);

    if (f.a < 1) continue;

    push();
    translate(f.x, f.y);

    const wiggle = sin(frameCount * 0.01 + f.x * 0.01) * 2;
    scale(f.dir, 1);

    noStroke();
    fill(this.theme.fish);
    ellipse(0, 0, 26, 12);

    triangle(-14, 0, -24, -6 + wiggle * 0.2, -24, 6 + wiggle * 0.2);

    fill(255, f.a);
    circle(8, -2, 2);

    pop();
  }
}

drawJellies(cam) {
  const cx = cam.x + width / 2;
  const cy = cam.y + height / 2;

  for (const j of this.jellies) {
    const d = dist(cx, cy, j.x, j.y);
    const targetA = d < 220 ? 255 : 0;
    j.a = lerp(j.a, targetA, 0.02);

    if (j.a < 1) continue;

    push();
    translate(j.x, j.y);

    const floatY = sin(frameCount * 0.01 + j.x * 0.01) * 6;

    noStroke();
    fill(127, 214, 255, j.a);
    ellipse(0, floatY, 34, 22);
```

```
      stroke(127, 214, 255, j.a);
      line(-6, floatY + 10, -8, floatY + 28);
      line(0, floatY + 10, 0, floatY + 32);
      line(6, floatY + 10, 8, floatY + 28);

      const pulse = sin(frameCount * 0.05) * 20;
      noFill();
      stroke(127, 214, 255, j.a * 0.4 + pulse);
      ellipse(0, floatY, 50, 34);

      pop();
    }
  }
}
```

✅ No platforms drawn at all
✅ Fish + jellyfish are **hidden** until camera gets close
✅ Still feels simple and calm

Quick note about platforms

You don't *have* to delete Platform.js or levels.json platform entries — we're just **not using them anymore**.

If you want it extra clean later, we can also:

- remove `Platform.js` usage from `WorldLevel` entirely (already done)
- simplify `levels.json` to only contain world size + theme

This has way too many features. I don't want any of these, i need it the most simplest as possible. Please get rid of the auto drifting, the jelly fish reshuffling, the particles, etc.

HAHA okay okay 😭 we are going FULL minimal.

No drifting noise.
No reshuffle.
No particles.
No extra systems.

Just:

- Arrow keys move camera
- One fish you follow
- A few jellyfish placed in the world
- Ocean gradient
- No platforms
- No player physics

Simple. Clean. Calm.

✅ What We're Changing

We will:

1. Remove player + platform logic
2. Keep Camera2D
3. Use arrow keys to move camera
4. Draw:
    - One fish (center of world)
    - A few jellyfish
    - Gradient background

That's it.

Replace `sketch.js` with THIS

```
const VIEW_W = 800;

const VIEW_H = 480;
```

```
let cam;
let level;

function setup() {
  createCanvas(VIEW_W, VIEW_H);

  cam = new Camera2D(width, height);

  level = new WorldLevel({
    world: { w: 2400, h: 1400 }
  });

  cam.clampToWorld(level.w, level.h);
}

function draw() {
  handleCameraMovement();

  cam.begin();
  level.drawWorld();
  cam.end();

  fill(255);
  text("Use arrow keys to explore", 10, 20);
}

function handleCameraMovement() {
  const speed = 4;

  if (keyIsDown(LEFT_ARROW)) cam.x -= speed;
  if (keyIsDown(RIGHT_ARROW)) cam.x += speed;
  if (keyIsDown(UP_ARROW)) cam.y -= speed;
  if (keyIsDown(DOWN_ARROW)) cam.y += speed;

  cam.clampToWorld(level.w, level.h);
}
```

◆ Replace `WorldLevel.js` with THIS

```
class WorldLevel {
  constructor(levelJson) {
    this.w = levelJson.world?.w ?? 2400;
    this.h = levelJson.world?.h ?? 1400;

    this.jellies = [];

    for (let i = 0; i < 8; i++) {
      this.jellies.push({
        x: random(200, this.w - 200),
        y: random(200, this.h - 200)
      });
    }

    this.fish = {
      x: this.w / 2,
      y: this.h / 2
    };
  }

  drawWorld() {
    this.drawGradient();

    this.drawFish();

    this.drawJellies();
  }

  drawGradient() {
    const topC = color(8, 25, 60);
    const botC = color(2, 5, 20);

    push();
    resetMatrix();
    for (let y = 0; y < height; y++) {
      const t = y / height;
      stroke(lerpColor(topC, botC, t));
      line(0, y, width, y);
    }
    pop();
  }

  drawFish() {
    push();
    translate(this.fish.x, this.fish.y);

    fill(20, 80, 120);
    noStroke();
    ellipse(0, 0, 30, 14);
    triangle(-15, 0, -28, -8, -28, 8);

    fill(255);
    circle(8, -3, 4);

    pop();
  }

  drawJellies() {
    for (let j of this.jellies) {
      push();
      translate(j.x, j.y);
```

```
   const floatY = sin(frameCount * 0.02 + j.x) * 8;

   fill(120, 220, 255, 180);
   noStroke();
   ellipse(0, floatY, 36, 24);

   stroke(120, 220, 255, 180);
   line(-6, floatY + 12, -6, floatY + 35);
   line(0, floatY + 12, 0, floatY + 38);
   line(6, floatY + 12, 6, floatY + 35);

   pop();
  }
 }
}
```

Keep `Camera2D.js` EXACTLY as it is

No changes needed.

What You Now Have

- Arrow keys move camera
- Fish sits in world center
- Jellyfish float gently

- Big world larger than screen
- Ocean gradient background
- No platforms
- No physics
- No complexity

Ultra simple.