For the 2 weeks side quest I wanted to enhance the original code. My idea was for the blob and background to change color during jumps to project it's mood and for it to eat "snacks" that would award the user with points. I increased the canvas and spacing, as well as adding a restart button.

GenAI was used during this Side Quest to assist with debugging, organization and to answer code questions. During this Side Quest, I realized that my CS105 & 106 knowledge was coming back, which was honestly satisfying. I reviewed the code recommendations and integrated them myself

Prompts:
Can you please help me integrate an object jumping with an if and else function and having little "snacks" for the object to consume.(I attached code snippets from the example)

Response: Refreshed my memory on loops and helped me build the code for the blob to change colours

Conversation with chatgpt

can you make the blob in this code change depending on mood: // Y-position of the floor (ground level) let floorY3; // Player character (soft, animated blob) let blob3 = { // Position (centre of the blob) x: 80, y: 0, // Visual properties r: 26, // Base radius points: 48, // Number of points used to draw the blob wobble: 7, // Edge deformation amount wobbleFreq: 0.9, // Time values for breathing animation t: 0, tSpeed: 0.01, // Physics: velocity vx: 0, // Horizontal velocity vy: 0, // Vertical velocity // Movement tuning accel: 0.55, // Horizontal acceleration maxRun: 4.0, // Maximum horizontal speed gravity: 0.65, // Downward force jumpV: -11.0, // Initial jump impulse // State onGround: false, // True when standing on a platform // Friction frictionAir: 0.995, // Light friction in air frictionGround: 0.88, // Stronger friction on ground }; // List of solid platforms the blob can stand on // Each platform is an axis-aligned rectangle (AABB) let platforms = []; function setup() { createCanvas(640, 360); // Define the floor height floorY3 = height - 36; noStroke(); textFont("sans-serif"); textSize(14); // Create platforms (floor + steps) platforms = [{ x: 0, y: floorY3, w: width, h: height - floorY3 }, // floor { x: 120, y: floorY3 - 70, w: 120, h: 12 }, // low step { x: 300, y: floorY3 - 120, w: 90, h: 12 }, // mid step { x: 440, y: floorY3 - 180, w: 130, h: 12 }, // high step { x: 520, y: floorY3 - 70, w: 90, h: 12 }, // return ramp]; // Start the blob resting on the floor blob3.y = floorY3 - blob3.r - 1; } function draw() { background(240); // --- Draw all platforms --- fill(200); for (const p of platforms) { rect(p.x, p.y, p.w, p.h); } // --- Input: left/right movement --- let move = 0; if (keyIsDown(65) || keyIsDown(LEFT_ARROW)) move -= 1; // A or ← if (keyIsDown(68) || keyIsDown(RIGHT_ARROW)) move += 1; // D or → blob3.vx += blob3.accel * move; // --- Apply friction and clamp speed --- blob3.vx *= blob3.onGround ? blob3.frictionGround : blob3.frictionAir; blob3.vx = constrain(blob3.vx, -blob3.maxRun, blob3.maxRun); // --- Apply gravity --- blob3.vy += blob3.gravity; // --- Collision representation --- // We collide using a rectangle (AABB), // even though the blob is drawn as a circle let box = { x: blob3.x - blob3.r, y: blob3.y - blob3.r, w: blob3.r * 2, h: blob3.r * 2, }; // --- STEP 1: Move horizontally, then resolve X collisions --- box.x += blob3.vx; for (const s of platforms) { if

(overlap(box, s)) { if (blob3.vx > 0) { // Moving right → hit the left side of a platform box.x = s.x - box.w; } else if (blob3.vx < 0) { // Moving left → hit the right side of a platform box.x = s.x + s.w; } blob3.vx = 0; } } // --- STEP 2: Move vertically, then resolve Y collisions --- box.y += blob3.vy; blob3.onGround = false; for (const s of platforms) { if (overlap(box, s)) { if (blob3.vy > 0) { // Falling → land on top of a platform box.y = s.y - box.h; blob3.vy = 0; blob3.onGround = true; } else if (blob3.vy < 0) { // Rising → hit the underside of a platform box.y = s.y + s.h; blob3.vy = 0; } } } // --- Convert collision box back to blob centre --- blob3.x = box.x + box.w / 2; blob3.y = box.y + box.h / 2; // Keep blob inside the canvas horizontally blob3.x = constrain(blob3.x, blob3.r, width - blob3.r); // --- Draw the animated blob --- blob3.t += blob3.tSpeed; drawBlobCircle(blob3); // --- HUD --- fill(0); text("Move: A/D or ←/→ • Jump: Space/W/↑ • Land on platforms", 10, 18); } // Axis-Aligned Bounding Box (AABB) overlap test // Returns true if rectangles a and b intersect function overlap(a, b) { return (a.x < b.x + b.w && a.x + a.w > b.x && a.y < b.y + b.h && a.y + a.h > b.y); } // Draws the blob using Perlin noise for a soft, breathing effect function drawBlobCircle(b) { fill(20, 120, 255); beginShape(); for (let i = 0; i < b.points; i++) { const a = (i / b.points) * TAU; // Noise-based radius offset const n = noise(cos(a) * b.wobbleFreq + 100, sin(a) * b.wobbleFreq + 100, b.t,); const r = b.r + map(n, 0, 1, -b.wobble, b.wobble); vertex(b.x + cos(a) * r, b.y + sin(a) * r); } endShape(CLOSE); } // Jump input (only allowed when grounded) function keyPressed() { if ((key === " " || key === "W" || key === "w" || keyCode === UP_ARROW) && blob3.onGround) { blob3.vy = blob3.jumpV; blob3.onGround = false; } } /* In-class tweaks for experimentation: • Add a new platform: platforms.push({ x: 220, y: floorY3 - 150, w: 80, h: 12 }); • “Ice” feel → frictionGround = 0.95 • “Sand” feel → frictionGround = 0.80 */

Yes. The clean way is to introduce a `mood` state (e.g., `"calm" | "happy" | "angry" | "sad" | "scared"`) and have it drive the blob’s **color**, **wobble**, **wobble frequency**, and **breathing speed**. Below is a drop-in set of changes that keeps your physics intact and only changes visuals/animation.

1) Add mood config + setters (place near the top, after `blob3`)

```
// --- Mood system
// ---------------------------------------------------------

let mood3 = "calm"; // "calm" | "happy" | "angry" | "sad" | "scared"

// Target values the blob will smoothly approach
blob3.mood = {
  r: blob3.r,
  wobble: blob3.wobble,
  wobbleFreq: blob3.wobbleFreq,
  tSpeed: blob3.tSpeed,
  color: [20, 120, 255], // [r,g,b]
};
```

```
// How quickly mood transitions blend (0..1). Higher = snappier.
const moodLerp = 0.08;

const MOODS = {
  calm:   { r: 26, wobble: 6,  wobbleFreq: 0.85, tSpeed: 0.010, color:
[20, 120, 255] },
  happy:  { r: 28, wobble: 9,  wobbleFreq: 1.20, tSpeed: 0.015, color:
[60, 200, 120] },
  angry:  { r: 27, wobble: 12, wobbleFreq: 1.60, tSpeed: 0.020, color:
[240, 80, 60] },
  sad:    { r: 24, wobble: 4,  wobbleFreq: 0.60, tSpeed: 0.007, color:
[80, 120, 200] },
  scared: { r: 23, wobble: 10, wobbleFreq: 2.10, tSpeed: 0.022, color:
[240, 220, 90] },
};

function setMood3(name) {
  mood3 = MOODS[name] ? name : "calm";
}

function applyMood3(b) {
  const m = MOODS[mood3];

  // Smoothly blend the current properties toward the mood targets
  b.r = lerp(b.r, m.r, moodLerp);
  b.wobble = lerp(b.wobble, m.wobble, moodLerp);
  b.wobbleFreq = lerp(b.wobbleFreq, m.wobbleFreq, moodLerp);
  b.tSpeed = lerp(b.tSpeed, m.tSpeed, moodLerp);

  b.mood.color[0] = lerp(b.mood.color[0], m.color[0], moodLerp);
  b.mood.color[1] = lerp(b.mood.color[1], m.color[1], moodLerp);
  b.mood.color[2] = lerp(b.mood.color[2], m.color[2], moodLerp);
}
```

2) Apply mood each frame (inside `draw()`, before you animate/draw the blob)

Add this right before:

```
// --- Draw the animated blob ---
blob3.t += blob3.tSpeed;
drawBlobCircle(blob3);
```

Replace that section with:

```
// --- Mood-driven animation parameters ---
applyMood3(blob3);

// --- Draw the animated blob ---
blob3.t += blob3.tSpeed;
drawBlobCircle(blob3);
```

3) Make the blob color mood-driven (edit `drawBlobCircle`)

Replace:

```
fill(20, 120, 255);
```

with:

```
fill(b.mood.color[0], b.mood.color[1], b.mood.color[2]);
```

4) Optional: add keyboard controls to change mood (edit `keyPressed()`)

Keep your jump logic, but add mood hotkeys underneath it:

```
function keyPressed() {
  // Jump input (only allowed when grounded)
  if (
    (key === " " || key === "W" || key === "w" || keyCode ===
UP_ARROW) &&
    blob3.onGround
  ) {
    blob3.vy = blob3.jumpV;
    blob3.onGround = false;
```

```
  }

  // Mood hotkeys
  if (key === "1") setMood3("calm");
  if (key === "2") setMood3("happy");
  if (key === "3") setMood3("angry");
  if (key === "4") setMood3("sad");
  if (key === "5") setMood3("scared");
}
```

5) Optional: show current mood in the HUD (inside `draw()` HUD section)

Add:

```
text(`Mood: ${mood3} (1-5 to change)`, 10, 36);
```

What this gives you

- **Calm:** steady breathing, mild wobble, blue.
- **Happy:** slightly bigger, bouncier wobble, faster breathing, green.
- **Angry:** “spikier” wobble (higher deformation + freq), faster breathing, red.
- **Sad:** smaller, smoother edge, slower breathing, cooler blue.
- **Scared:** smaller, jittery/faster wobble, yellow.

If you tell me what you mean by “mood” in your game (player speed? time in air? collisions? proximity to danger?), I can also auto-switch moods based on gameplay signals rather than key presses.

can you incorporate this into my code

```
// Y-position of the floor (ground level)
let floorY3;

// Player character (soft, animated blob)
let blob3 = {
  // Position (centre of the blob)
  x: 80,
  y: 0,

  // Visual properties
```

```
  r: 26, // Base radius
  points: 48, // Number of points used to draw the blob
  wobble: 7, // Edge deformation amount
  wobbleFreq: 0.9,

  // Time values for breathing animation
  t: 0,
  tSpeed: 0.01,

  // Physics: velocity
  vx: 0, // Horizontal velocity
  vy: 0, // Vertical velocity

  // Movement tuning
  accel: 0.55, // Horizontal acceleration
  maxRun: 4.0, // Maximum horizontal speed
  gravity: 0.65, // Downward force
  jumpV: -11.0, // Initial jump impulse

  // State
  onGround: false, // True when standing on a platform

  // Friction
  frictionAir: 0.995, // Light friction in air
  frictionGround: 0.88, // Stronger friction on ground
};

// --- Mood system
------------------------------------------------------------
// Current mood. Change via keys 1-5 (see keyPressed).
let mood3 = "calm"; // "calm" | "happy" | "angry" | "sad" | "scared"

// How quickly the blob transitions between moods (0..1). Higher =
snappier.
const moodLerp = 0.08;

// Mood presets (visual + animation only; physics unchanged)
const MOODS = {
```

```
  calm:   { r: 26, wobble: 6,  wobbleFreq: 0.85, tSpeed: 0.010, color:
[20, 120, 255] },
  happy:  { r: 28, wobble: 9,  wobbleFreq: 1.20, tSpeed: 0.015, color:
[60, 200, 120] },
  angry:  { r: 27, wobble: 12, wobbleFreq: 1.60, tSpeed: 0.020, color:
[240, 80, 60] },
  sad:    { r: 24, wobble: 4,  wobbleFreq: 0.60, tSpeed: 0.007, color:
[80, 120, 200] },
  scared: { r: 23, wobble: 10, wobbleFreq: 2.10, tSpeed: 0.022, color:
[240, 220, 90] },
};

// Store a current (blended) color on the blob
blob3.mood = { color: [20, 120, 255] };

function setMood3(name) {
  mood3 = MOODS[name] ? name : "calm";
}

function applyMood3(b) {
  const m = MOODS[mood3];

  // Smoothly blend the current properties toward the mood targets
  b.r = lerp(b.r, m.r, moodLerp);
  b.wobble = lerp(b.wobble, m.wobble, moodLerp);
  b.wobbleFreq = lerp(b.wobbleFreq, m.wobbleFreq, moodLerp);
  b.tSpeed = lerp(b.tSpeed, m.tSpeed, moodLerp);

  b.mood.color[0] = lerp(b.mood.color[0], m.color[0], moodLerp);
  b.mood.color[1] = lerp(b.mood.color[1], m.color[1], moodLerp);
  b.mood.color[2] = lerp(b.mood.color[2], m.color[2], moodLerp);
}

// List of solid platforms the blob can stand on
// Each platform is an axis-aligned rectangle (AABB)
let platforms = [];

function setup() {
```

```
  createCanvas(640, 360);

  // Define the floor height
  floorY3 = height - 36;

  noStroke();
  textFont("sans-serif");
  textSize(14);

  // Create platforms (floor + steps)
  platforms = [
    { x: 0, y: floorY3, w: width, h: height - floorY3 }, // floor
    { x: 120, y: floorY3 - 70, w: 120, h: 12 }, // low step
    { x: 300, y: floorY3 - 120, w: 90, h: 12 }, // mid step
    { x: 440, y: floorY3 - 180, w: 130, h: 12 }, // high step
    { x: 520, y: floorY3 - 70, w: 90, h: 12 }, // return ramp
  ];

  // Start the blob resting on the floor
  blob3.y = floorY3 - blob3.r - 1;
}

function draw() {
  background(240);

  // --- Draw all platforms ---
  fill(200);
  for (const p of platforms) {
    rect(p.x, p.y, p.w, p.h);
  }

  // --- Input: left/right movement ---
  let move = 0;
  if (keyIsDown(65) || keyIsDown(LEFT_ARROW)) move -= 1; // A or ←
  if (keyIsDown(68) || keyIsDown(RIGHT_ARROW)) move += 1; // D or →
  blob3.vx += blob3.accel * move;

  // --- Apply friction and clamp speed ---
```

```
  blob3.vx *= blob3.onGround ? blob3.frictionGround :
blob3.frictionAir;
  blob3.vx = constrain(blob3.vx, -blob3.maxRun, blob3.maxRun);

  // --- Apply gravity ---
  blob3.vy += blob3.gravity;

  // --- Collision representation ---
  // We collide using a rectangle (AABB),
  // even though the blob is drawn as a circle
  let box = {
    x: blob3.x - blob3.r,
    y: blob3.y - blob3.r,
    w: blob3.r * 2,
    h: blob3.r * 2,
  };

  // --- STEP 1: Move horizontally, then resolve X collisions ---
  box.x += blob3.vx;
  for (const s of platforms) {
    if (overlap(box, s)) {
      if (blob3.vx > 0) {
        // Moving right → hit the left side of a platform
        box.x = s.x - box.w;
      } else if (blob3.vx < 0) {
        // Moving left → hit the right side of a platform
        box.x = s.x + s.w;
      }
      blob3.vx = 0;
    }
  }

  // --- STEP 2: Move vertically, then resolve Y collisions ---
  box.y += blob3.vy;
  blob3.onGround = false;

  for (const s of platforms) {
    if (overlap(box, s)) {
```

```
      if (blob3.vy > 0) {
        // Falling → land on top of a platform
        box.y = s.y - box.h;
        blob3.vy = 0;
        blob3.onGround = true;
      } else if (blob3.vy < 0) {
        // Rising → hit the underside of a platform
        box.y = s.y + s.h;
        blob3.vy = 0;
      }
    }
  }

  // --- Convert collision box back to blob centre ---
  blob3.x = box.x + box.w / 2;
  blob3.y = box.y + box.h / 2;

  // Keep blob inside the canvas horizontally
  blob3.x = constrain(blob3.x, blob3.r, width - blob3.r);

  // --- Mood-driven animation parameters (visuals only) ---
  applyMood3(blob3);

  // --- Draw the animated blob ---
  blob3.t += blob3.tSpeed;
  drawBlobCircle(blob3);

  // --- HUD ---
  fill(0);
  text("Move: A/D or ←/→  •  Jump: Space/W/↑  •  Land on platforms",
10, 18);
  text(`Mood: ${mood3} (1-5 to change)`, 10, 36);
}

// Axis-Aligned Bounding Box (AABB) overlap test
// Returns true if rectangles a and b intersect
function overlap(a, b) {
  return (
```

```
    a.x < b.x + b.w && a.x + a.w > b.x && a.y < b.y + b.h && a.y + a.h
> b.y
  );
}

// Draws the blob using Perlin noise for a soft, breathing effect
function drawBlobCircle(b) {
  fill(b.mood.color[0], b.mood.color[1], b.mood.color[2]);
  beginShape();

  for (let i = 0; i < b.points; i++) {
    const a = (i / b.points) * TAU;

    // Noise-based radius offset
    const n = noise(
      cos(a) * b.wobbleFreq + 100,
      sin(a) * b.wobbleFreq + 100,
      b.t,
    );

    const r = b.r + map(n, 0, 1, -b.wobble, b.wobble);

    vertex(b.x + cos(a) * r, b.y + sin(a) * r);
  }

  endShape(CLOSE);
}

// Jump input (only allowed when grounded)
function keyPressed() {
  if (
    (key === " " || key === "W" || key === "w" || keyCode ===
UP_ARROW) &&
    blob3.onGround
  ) {
    blob3.vy = blob3.jumpV;
    blob3.onGround = false;
  }
```

```
  // Mood hotkeys
  if (key === "1") setMood3("calm");
  if (key === "2") setMood3("happy");
  if (key === "3") setMood3("angry");
  if (key === "4") setMood3("sad");
  if (key === "5") setMood3("scared");
}

/* In-class tweaks for experimentation:
   • Add a new platform:
     platforms.push({ x: 220, y: floorY3 - 150, w: 80, h: 12 });

   • “Ice” feel → frictionGround = 0.95
   • “Sand” feel → frictionGround = 0.80
*/
```

i want the mood to change depending on its jump, when it jumps - happy, fall off angry, doing nothing calm

```
// Y-position of the floor (ground level)
let floorY3;

// Player character (soft, animated blob)
let blob3 = {
  // Position (centre of the blob)
  x: 80,
  y: 0,

  // Visual properties
  r: 26, // Base radius
  points: 48, // Number of points used to draw the blob
  wobble: 7, // Edge deformation amount
  wobbleFreq: 0.9,

  // Time values for breathing animation
  t: 0,
  tSpeed: 0.01,

  // Physics: velocity
```

```
  vx: 0, // Horizontal velocity
  vy: 0, // Vertical velocity

  // Movement tuning
  accel: 0.55, // Horizontal acceleration
  maxRun: 4.0, // Maximum horizontal speed
  gravity: 0.65, // Downward force
  jumpV: -11.0, // Initial jump impulse

  // State
  onGround: false, // True when standing on a platform

  // Friction
  frictionAir: 0.995, // Light friction in air
  frictionGround: 0.88, // Stronger friction on ground
};

// --- Mood system
------------------------------------------------------------
// Mood changes automatically based on motion:
// - Jumping / rising => happy
// - Falling (not grounded) => angry
// - On ground / idle => calm
let mood3 = "calm";

// How quickly the blob transitions between moods (0..1). Higher =
snappier.
const moodLerp = 0.10;

// Mood presets (visual + animation only; physics unchanged)
const MOODS = {
  calm:  { r: 26, wobble: 6,  wobbleFreq: 0.85, tSpeed: 0.010, color:
[20, 120, 255] },
  happy: { r: 28, wobble: 9,  wobbleFreq: 1.20, tSpeed: 0.015, color:
[60, 200, 120] },
  angry: { r: 27, wobble: 12, wobbleFreq: 1.60, tSpeed: 0.020, color:
[240, 80, 60] },
};
```

```
// Store a current (blended) color on the blob
blob3.mood = { color: [20, 120, 255] };

// Detect "fall off" as: was grounded last frame, and now not grounded
and moving down
let wasOnGround3 = false;

function setMood3(name) {
  mood3 = MOODS[name] ? name : "calm";
}

function updateMoodFromMotion3(b) {
  // Priority:
  // 1) Rising (jumping) -> happy
  // 2) Falling in air -> angry
  // 3) Otherwise -> calm
  //
  // Note: vy < 0 means moving up; vy > 0 means moving down.
  if (b.vy < -0.2) {
    setMood3("happy");
    return;
  }
  if (!b.onGround && b.vy > 0.4) {
    // Includes "falling off" and normal falling
    setMood3("angry");
    return;
  }
  setMood3("calm");
}

function applyMood3(b) {
  const m = MOODS[mood3];

  // Smoothly blend the current properties toward the mood targets
  b.r = lerp(b.r, m.r, moodLerp);
  b.wobble = lerp(b.wobble, m.wobble, moodLerp);
  b.wobbleFreq = lerp(b.wobbleFreq, m.wobbleFreq, moodLerp);
```

```
  b.tSpeed = lerp(b.tSpeed, m.tSpeed, moodLerp);

  b.mood.color[0] = lerp(b.mood.color[0], m.color[0], moodLerp);
  b.mood.color[1] = lerp(b.mood.color[1], m.color[1], moodLerp);
  b.mood.color[2] = lerp(b.mood.color[2], m.color[2], moodLerp);
}

// List of solid platforms the blob can stand on
// Each platform is an axis-aligned rectangle (AABB)
let platforms = [];

function setup() {
  createCanvas(640, 360);

  // Define the floor height
  floorY3 = height - 36;

  noStroke();
  textFont("sans-serif");
  textSize(14);

  // Create platforms (floor + steps)
  platforms = [
    { x: 0, y: floorY3, w: width, h: height - floorY3 }, // floor
    { x: 120, y: floorY3 - 70, w: 120, h: 12 }, // low step
    { x: 300, y: floorY3 - 120, w: 90, h: 12 }, // mid step
    { x: 440, y: floorY3 - 180, w: 130, h: 12 }, // high step
    { x: 520, y: floorY3 - 70, w: 90, h: 12 }, // return ramp
  ];

  // Start the blob resting on the floor
  blob3.y = floorY3 - blob3.r - 1;
}

function draw() {
  background(240);

  // --- Draw all platforms ---
```

```
  fill(200);
  for (const p of platforms) rect(p.x, p.y, p.w, p.h);

  // --- Input: left/right movement ---
  let move = 0;
  if (keyIsDown(65) || keyIsDown(LEFT_ARROW)) move -= 1; // A or ←
  if (keyIsDown(68) || keyIsDown(RIGHT_ARROW)) move += 1; // D or →
  blob3.vx += blob3.accel * move;

  // --- Apply friction and clamp speed ---
  blob3.vx *= blob3.onGround ? blob3.frictionGround :
blob3.frictionAir;
  blob3.vx = constrain(blob3.vx, -blob3.maxRun, blob3.maxRun);

  // --- Apply gravity ---
  blob3.vy += blob3.gravity;

  // --- Collision representation ---
  let box = {
    x: blob3.x - blob3.r,
    y: blob3.y - blob3.r,
    w: blob3.r * 2,
    h: blob3.r * 2,
  };

  // --- STEP 1: Move horizontally, then resolve X collisions ---
  box.x += blob3.vx;
  for (const s of platforms) {
    if (overlap(box, s)) {
      if (blob3.vx > 0) box.x = s.x - box.w;
      else if (blob3.vx < 0) box.x = s.x + s.w;
      blob3.vx = 0;
    }
  }

  // --- STEP 2: Move vertically, then resolve Y collisions ---
  box.y += blob3.vy;
  blob3.onGround = false;
```

```
  for (const s of platforms) {
    if (overlap(box, s)) {
      if (blob3.vy > 0) {
        box.y = s.y - box.h;
        blob3.vy = 0;
        blob3.onGround = true;
      } else if (blob3.vy < 0) {
        box.y = s.y + s.h;
        blob3.vy = 0;
      }
    }
  }

  // --- Convert collision box back to blob centre ---
  blob3.x = box.x + box.w / 2;
  blob3.y = box.y + box.h / 2;

  // Keep blob inside the canvas horizontally
  blob3.x = constrain(blob3.x, blob3.r, width - blob3.r);

  // --- Mood update (based on jump/fall/idle) ---
  // If you want "angry ONLY when it falls off a ledge (not normal
falling)",
  // uncomment the alternate logic below.
  updateMoodFromMotion3(blob3);

  // Alternate "fall off specifically" logic (optional):
  // const fellOff = wasOnGround3 && !blob3.onGround && blob3.vy >
0.4;
  // if (blob3.vy < -0.2) setMood3("happy");
  // else if (fellOff) setMood3("angry");
  // else setMood3("calm");
  // wasOnGround3 = blob3.onGround;

  // --- Apply mood-driven animation parameters (visuals only) ---
  applyMood3(blob3);
```

```
  // --- Draw the animated blob ---
  blob3.t += blob3.tSpeed;
  drawBlobCircle(blob3);

  // --- HUD ---
  fill(0);
  text("Move: A/D or ←/→  •  Jump: Space/W/↑  •  Land on platforms",
10, 18);
  text(`Mood: ${mood3}`, 10, 36);

  // Track grounded state for "fell off" detection if you use the
alternate logic
  wasOnGround3 = blob3.onGround;
}

// Axis-Aligned Bounding Box (AABB) overlap test
function overlap(a, b) {
  return a.x < b.x + b.w && a.x + a.w > b.x && a.y < b.y + b.h && a.y
+ a.h > b.y;
}

// Draws the blob using Perlin noise for a soft, breathing effect
function drawBlobCircle(b) {
  fill(b.mood.color[0], b.mood.color[1], b.mood.color[2]);
  beginShape();

  for (let i = 0; i < b.points; i++) {
    const a = (i / b.points) * TAU;

    // Noise-based radius offset
    const n = noise(
      cos(a) * b.wobbleFreq + 100,
      sin(a) * b.wobbleFreq + 100,
      b.t,
    );

    const r = b.r + map(n, 0, 1, -b.wobble, b.wobble);
    vertex(b.x + cos(a) * r, b.y + sin(a) * r);
```

```
  }

  endShape(CLOSE);
}

// Jump input (only allowed when grounded)
function keyPressed() {
  if (
    (key === " " || key === "W" || key === "w" || keyCode ===
UP_ARROW) &&
    blob3.onGround
  ) {
    blob3.vy = blob3.jumpV;
    blob3.onGround = false;
  }
}

/* In-class tweaks for experimentation:
   • Add a new platform:
     platforms.push({ x: 220, y: floorY3 - 150, w: 80, h: 12 });

   • “Ice” feel → frictionGround = 0.95
   • “Sand” feel → frictionGround = 0.80
*/
```

Redesign the blob’s movement and environment to express a specific emotion (e.g., joy, frustration, panic). Bonus: Add a “mischief” mechanic, your blob steals or bumps objects oh a small map.

Below is a concrete redesign that will read as **panic** (high-arousal, jittery, reactive), plus a “mischief” system where the blob **bumps/steals** small objects on a compact map. I’ll keep it in the same p5.js style as your current sketch, and I’ll describe exactly what to change.

Emotion redesign: Panic

What the player should feel

- The blob is “overstimulated”: **twitchy acceleration**, **slippery stops**, **overreactive jumps**, and **breathing that spikes** when airborne.
- The world feels unsafe: **narrow platforms**, **moving hazards**, **wind gust zones**, and **tiny collectibles** that tempt risky choices.
- Visual language: **fast wobble**, more “ragged” edge when falling, color shifting toward warning tones while airborne.

Movement changes (panic tuning)

Use three movement modes driven by state:

1. **Grounded (panic-but-trying-to-stabilize)**
    - High acceleration, low traction (sliding).
    - Small involuntary micro-jitters (subtle vx noise).
2. **Jumping (adrenaline spike)**
    - Jump impulse slightly stronger.
    - Short “burst” after jump where gravity is lighter for 0.12s (feels floaty/erratic).
3. **Falling (panic peak)**
    - Gravity stronger + optional fast-fall if the player holds DOWN.
    - Wobble increases and “stretches” vertically (squash & stretch).

Environment redesign: Small panic map

Build a compact “stress course”:

- Floor plus 3 tight platforms.
- One moving platform that oscillates.
- One “wind strip” zone that pushes the blob sideways (unpredictable).

Suggested layout:

- Platform 1: low and wide
- Platform 2: mid narrow
- Platform 3: high narrow near a wind strip
- Moving platform bridging return route

Bonus mechanic: Mischief (bump + steal)

Add 6–10 small objects (“trinkets”) on the map:

- They have positions, radii, and velocities.
- If the blob collides at speed: **bump** them away (impulse).
- If the blob presses **E** while overlapping: **steal** (pickup).
- Stolen items orbit the blob in a little “chain” (visible mischief).
- Optional: dropping them at a “nest” zone scores points.

This makes the blob feel like an anxious troublemaker: frantic movement + compulsive grabbing.

Drop-in code additions (designed for your current sketch)

You asked for redesign, so here are “modules” you can paste in with minimal disruption. This assumes your current collision approach (AABB for platforms). For objects, we’ll use circle collisions.

1) Add panic tuning + state timers (near blob3)

```
// Panic movement tuning (emotion)
blob3.panic = {
  jitter: 0.06,        // micro jitter on ground
  airControl: 0.60,    // less control in air
  groundTraction: 0.84,// lower = more sliding
  jumpBoostFrames: 8,  // brief floaty boost
  jumpBoost: -0.15,    // reduces gravity briefly (negative =
counter-gravity)
  fallGravity: 0.80,   // stronger falling gravity than base
  fastFall: 1.25,      // extra gravity when holding DOWN
};

blob3.jumpBoostLeft = 0; // frames of post-jump boost
```

Then, change friction and control dynamically in `draw()`:

- On ground: traction from panic settings
- In air: reduce accel effect

2) Replace your accel + friction section in `draw()` with panic logic

Find:

```
blob3.vx += blob3.accel * move;
blob3.vx *= blob3.onGround ? blob3.frictionGround : blob3.frictionAir;
```

Replace with:

```
// Panic: reduced control in air, twitchy ground
const control = blob3.onGround ? 1.0 : blob3.panic.airControl;

// Add intentional input
blob3.vx += blob3.accel * move * control;

// Add subtle jitter when grounded and idle/moving (panic tremor)
if (blob3.onGround) {
  blob3.vx += (noise(frameCount * 0.02) - 0.5) * blob3.panic.jitter;
}

// Panic traction on ground (slide), otherwise air friction
const traction = blob3.onGround ? blob3.panic.groundTraction :
blob3.frictionAir;
blob3.vx *= traction;
```

3) Replace gravity section with panic gravity + jump boost + fast fall

Find:

```
blob3.vy += blob3.gravity;
```

Replace with:

```
// Panic gravity model
let g = blob3.gravity;
```

```
// Stronger gravity when falling (panic drop)
if (blob3.vy > 0 && !blob3.onGround) g = blob3.panic.fallGravity;

// Brief post-jump "adrenaline float"
if (blob3.jumpBoostLeft > 0) {
  g += blob3.panic.jumpBoost; // slightly reduces gravity
  blob3.jumpBoostLeft--;
}

// Optional fast fall (hold DOWN/S)
if (!blob3.onGround && (keyIsDown(DOWN_ARROW) || keyIsDown(83))) {
  g *= blob3.panic.fastFall;
}

blob3.vy += g;
```

4) In `keyPressed()`, start the jump boost window

In your jump block, after setting `vy`:

```
blob3.jumpBoostLeft = blob3.panic.jumpBoostFrames;
```

So:

```
blob3.vy = blob3.jumpV;
blob3.jumpBoostLeft = blob3.panic.jumpBoostFrames;
```

Environment changes: moving platform + wind zone

5) Add a moving platform object and wind zone (globals)

Add near `platforms`:

```
let movingPlat3;
let windZones3 = [];
```

In `setup()`, after platforms:

```
movingPlat3 = { x0: 260, y: floorY3 - 160, w: 110, h: 12, amp: 60,
spd: 0.02, x: 260, y: floorY3 - 160 };

windZones3 = [
  { x: 420, y: floorY3 - 220, w: 160, h: 120, force: -0.12 }, //
pushes left near top
];
```

In `draw()`, before collision checks, update the moving platform and include it:

```
// Update moving platform
movingPlat3.x = movingPlat3.x0 + sin(frameCount * movingPlat3.spd) *
movingPlat3.amp;

// Build a dynamic platforms list for this frame
const solids = [...platforms, { x: movingPlat3.x, y: movingPlat3.y, w:
movingPlat3.w, h: movingPlat3.h }];
```

Then replace your platform loops `for (const s of platforms)` with `for (const s of solids)`.

Draw it:

```
fill(180);
rect(movingPlat3.x, movingPlat3.y, movingPlat3.w, movingPlat3.h);
```

Apply wind (after you compute `box` but before resolving collisions is fine):

```
// Wind zones (panic instability)
for (const z of windZones3) {
```

```
  if (blob3.x > z.x && blob3.x < z.x + z.w && blob3.y > z.y && blob3.y
< z.y + z.h) {
    blob3.vx += z.force;
  }
}

// Visualize wind zones (optional)
noFill();
stroke(120);
for (const z of windZones3) rect(z.x, z.y, z.w, z.h);
noStroke();
```

Mischief mechanic: bump + steal objects

6) Add trinkets list + inventory

Globals:

```
let trinkets3 = [];
let stolen3 = []; // indices or objects
let score3 = 0;
let stealCooldown3 = 0;
```

In `setup()` create objects:

```
trinkets3 = [];
for (let i = 0; i < 8; i++) {
  trinkets3.push({
    x: random(120, width - 60),
    y: random(40, floorY3 - 90),
    r: random(6, 10),
    vx: 0,
    vy: 0,
    stolen: false,
  });
```

```
}
```

Physics + bump + pickup in `draw()` after blob collision resolution (when you know blob3.x/y are final):

```
if (stealCooldown3 > 0) stealCooldown3--;

// Blob circle for object collision
const bx = blob3.x, by = blob3.y, br = blob3.r;

// Update and draw trinkets
for (const t of trinkets3) {
  if (t.stolen) continue;

  // Simple physics for trinkets
  t.vy += 0.45;
  t.vx *= 0.98;
  t.vy *= 0.98;

  // Floor collide
  if (t.y + t.r > floorY3) {
    t.y = floorY3 - t.r;
    t.vy *= -0.35;
  }

  // Move
  t.x += t.vx;
  t.y += t.vy;

  // Keep in bounds
  t.x = constrain(t.x, t.r, width - t.r);

  // Bump interaction (collision circle-circle)
  const dx = t.x - bx;
  const dy = t.y - by;
  const d = sqrt(dx * dx + dy * dy);
  const hit = d < t.r + br;
```

```
  if (hit) {
    // Push it away proportional to blob speed (mischief bump)
    const speed = abs(blob3.vx) + abs(blob3.vy);
    const nx = dx / (d || 1);
    const ny = dy / (d || 1);

    const impulse = constrain(speed * 0.9, 0.6, 6.5);
    t.vx += nx * impulse;
    t.vy += ny * impulse;

    // Separate to avoid sticking
    const overlapAmt = (t.r + br) - d;
    t.x += nx * overlapAmt * 0.6;
    t.y += ny * overlapAmt * 0.6;
  }

  // Draw
  fill(40, 40, 40, 180);
  circle(t.x, t.y, t.r * 2);
}
```

Pickup (“steal”) on E when overlapping:

Add in `keyPressed()`:

```
if ((key === "E" || key === "e") && stealCooldown3 === 0) {
  // Attempt steal: nearest overlapping trinket
  let best = null;
  let bestD = 1e9;

  for (const t of trinkets3) {
    if (t.stolen) continue;
    const dx = t.x - blob3.x;
    const dy = t.y - blob3.y;
    const d = sqrt(dx*dx + dy*dy);
    if (d < t.r + blob3.r && d < bestD) {
      bestD = d;
      best = t;
    }
```

```
  }

  if (best) {
    best.stolen = true;
    stolen3.push(best);
    score3 += 1;
    stealCooldown3 = 10; // small debounce
  }
}
```

Render stolen items orbiting the blob (inside `draw()`, after drawing the blob):

```
// Draw "stolen" items orbiting the blob (mischief chain)
for (let i = 0; i < stolen3.length; i++) {
  const ang = frameCount * 0.06 + i * 0.8;
  const rad = blob3.r + 10 + i * 6;
  const sx = blob3.x + cos(ang) * rad;
  const sy = blob3.y + sin(ang) * (rad * 0.6);
  fill(30);
  circle(sx, sy, 10);
}
```

HUD:

```
text(`Emotion: PANIC  •  Mischief stolen: ${score3}  •  Press E to
steal when touching`, 10, 54);
```

Emotion cues tied to panic (optional but strong)

- When airborne, increase wobble and shift color warmer.
  You already have mood logic; for “panic,” do:
    - if `!onGround && vy > 0`: wobble spikes and color reddens.
    - if `onGround && abs(vx) < small`: calm blue returns.

This will visually reinforce panic without extra UI.

If you want: I can deliver a single consolidated final code block

Right now I've given you drop-in sections to avoid breaking your sketch. If you want a **fully merged** version (one pasteable file), say:

- "Make it joy instead of panic" or "keep panic"
- Do you want "angry only when falling off a ledge" (sudden drop), or "angry whenever falling"?

I can then return the complete, cleaned-up full script in one block.

just give me a solid code

```
// PANIC + MISCHIEF p5.js PLATFORMER BLOB
// Emotion: Panic (twitchy movement, stronger falling, jitter, wind zones)
// Bonus: Mischief objects you can bump/steal on a small map (press E to steal)

// ---------------------- Globals ----------------------
let floorY3;

let blob3 = {
  x: 80,
  y: 0,

  // Visual (will be mood-driven)
  r: 26,
  points: 48,
  wobble: 7,
  wobbleFreq: 0.9,

  t: 0,
  tSpeed: 0.01,

  vx: 0,
  vy: 0,

  accel: 0.55,
```

```
  maxRun: 4.2,

  gravity: 0.65,
  jumpV: -11.2,

  onGround: false,

  frictionAir: 0.992,
  frictionGround: 0.88,

  // Panic tuning
  panic: {
    jitter: 0.07,        // micro-jitter on ground
    airControl: 0.58,    // reduced control in air
    groundTraction: 0.84,// lower = more sliding
    jumpBoostFrames: 8,  // brief floaty boost after jump
    jumpBoost: -0.16,    // reduces gravity briefly
    fallGravity: 0.82,   // stronger gravity when falling
    fastFall: 1.25,      // extra gravity when holding DOWN/S
  },

  jumpBoostLeft: 0,

  // Mood visuals
  mood: {
    color: [20, 120, 255],
  },
};

let mood3 = "calm";
const moodLerp = 0.12;
const MOODS = {
  calm:  { r: 26, wobble: 6,  wobbleFreq: 0.85, tSpeed: 0.010, color:
[20, 120, 255] },
  happy: { r: 28, wobble: 9,  wobbleFreq: 1.20, tSpeed: 0.015, color:
[60, 200, 120] },
  angry: { r: 27, wobble: 12, wobbleFreq: 1.60, tSpeed: 0.020, color:
[240, 80, 60] },
```

```
};

let platforms = [];
let movingPlat3;
let windZones3 = [];

// Mischief objects
let trinkets3 = [];
let stolen3 = [];
let score3 = 0;
let stealCooldown3 = 0;

// For detecting "fell off"
let wasOnGround3 = false;

// ---------------------- Setup ----------------------
function setup() {
  createCanvas(640, 360);
  floorY3 = height - 36;

  noStroke();
  textFont("sans-serif");
  textSize(14);

  // Small "panic" map layout (tight + vertical)
  platforms = [
    { x: 0,   y: floorY3,      w: width, h: height - floorY3 }, //
floor
    { x: 90,  y: floorY3 - 60, w: 140,   h: 12 },               // low
wide
    { x: 260, y: floorY3 - 105,w: 95,    h: 12 },               // mid
narrow
    { x: 430, y: floorY3 - 165,w: 80,    h: 12 },               //
high narrow
    { x: 540, y: floorY3 - 95, w: 75,    h: 12 },               //
return ledge
  ];
```

```
  // Moving platform bridge
  movingPlat3 = {
    x0: 320,
    y: floorY3 - 200,
    w: 120,
    h: 12,
    amp: 75,
    spd: 0.022,
    x: 320,
  };

  // Wind zone near the top (panic instability)
  windZones3 = [
    { x: 385, y: floorY3 - 250, w: 220, h: 140, force: -0.13 },
  ];

  // Start blob on floor
  blob3.y = floorY3 - blob3.r - 1;

  // Spawn trinkets
  trinkets3 = [];
  for (let i = 0; i < 9; i++) {
    trinkets3.push({
      x: random(110, width - 60),
      y: random(50, floorY3 - 110),
      r: random(6, 10),
      vx: 0,
      vy: 0,
      stolen: false,
    });
  }
}

// ---------------------- Draw Loop ----------------------
function draw() {
  background(240);

  // Update moving platform
```

```
  movingPlat3.x = movingPlat3.x0 + sin(frameCount * movingPlat3.spd) *
movingPlat3.amp;

  // Build dynamic solids list for collision
  const solids = [
    ...platforms,
    { x: movingPlat3.x, y: movingPlat3.y, w: movingPlat3.w, h:
movingPlat3.h },
  ];

  // Draw static platforms
  fill(200);
  for (const p of platforms) rect(p.x, p.y, p.w, p.h);

  // Draw moving platform
  fill(180);
  rect(movingPlat3.x, movingPlat3.y, movingPlat3.w, movingPlat3.h);

  // Draw wind zones (outline)
  noFill();
  stroke(120);
  for (const z of windZones3) rect(z.x, z.y, z.w, z.h);
  noStroke();

  // ---------------- Input: left/right ----------------
  let move = 0;
  if (keyIsDown(65) || keyIsDown(LEFT_ARROW)) move -= 1; // A or ←
  if (keyIsDown(68) || keyIsDown(RIGHT_ARROW)) move += 1; // D or →

  // Panic control: reduced in air
  const control = blob3.onGround ? 1.0 : blob3.panic.airControl;
  blob3.vx += blob3.accel * move * control;

  // Panic tremor on ground
  if (blob3.onGround) {
    blob3.vx += (noise(frameCount * 0.02) - 0.5) * blob3.panic.jitter;
  }
```

```
  // Apply traction/friction
  const traction = blob3.onGround ? blob3.panic.groundTraction :
blob3.frictionAir;
  blob3.vx *= traction;
  blob3.vx = constrain(blob3.vx, -blob3.maxRun, blob3.maxRun);

  // ---------------- Wind zones affect vx ----------------
  for (const z of windZones3) {
    if (
      blob3.x > z.x && blob3.x < z.x + z.w &&
      blob3.y > z.y && blob3.y < z.y + z.h
    ) {
      blob3.vx += z.force;
    }
  }

  // ---------------- Gravity (panic) ----------------
  let g = blob3.gravity;

  // Stronger gravity when falling
  if (blob3.vy > 0 && !blob3.onGround) g = blob3.panic.fallGravity;

  // Brief post-jump float
  if (blob3.jumpBoostLeft > 0) {
    g += blob3.panic.jumpBoost;
    blob3.jumpBoostLeft--;
  }

  // Optional fast fall
  if (!blob3.onGround && (keyIsDown(DOWN_ARROW) || keyIsDown(83))) {
    g *= blob3.panic.fastFall;
  }

  blob3.vy += g;

  // ---------------- Collision box (AABB) ----------------
  let box = {
    x: blob3.x - blob3.r,
```

```
  y: blob3.y - blob3.r,
  w: blob3.r * 2,
  h: blob3.r * 2,
};

// STEP 1: horizontal move + resolve
box.x += blob3.vx;
for (const s of solids) {
  if (overlap(box, s)) {
    if (blob3.vx > 0) box.x = s.x - box.w;
    else if (blob3.vx < 0) box.x = s.x + s.w;
    blob3.vx = 0;
  }
}

// STEP 2: vertical move + resolve
box.y += blob3.vy;
blob3.onGround = false;

for (const s of solids) {
  if (overlap(box, s)) {
    if (blob3.vy > 0) {
      box.y = s.y - box.h;
      blob3.vy = 0;
      blob3.onGround = true;
    } else if (blob3.vy < 0) {
      box.y = s.y + s.h;
      blob3.vy = 0;
    }
  }
}

// Convert box back to center
blob3.x = box.x + box.w / 2;
blob3.y = box.y + box.h / 2;

// Keep inside canvas
blob3.x = constrain(blob3.x, blob3.r, width - blob3.r);
```

```
  // ---------------- Mood logic (per your rules) ----------------
  // Jumping/rising => happy
  // Fall off => angry (was grounded last frame, now not grounded and
moving down)
  // Doing nothing (grounded) => calm
  const fellOff = wasOnGround3 && !blob3.onGround && blob3.vy > 0.4;

  if (blob3.vy < -0.2) setMood3("happy");
  else if (fellOff) setMood3("angry");
  else if (blob3.onGround) setMood3("calm");
  else {
    // In air but not "fell off": keep calm-ish so "angry" is
specifically falling off
    setMood3("calm");
  }

  wasOnGround3 = blob3.onGround;

  // Blend mood visuals into blob parameters
  applyMood3(blob3);

  // ---------------- Mischief: update + bump trinkets
----------------
  updateTrinkets3();

  // ---------------- Draw blob ----------------
  blob3.t += blob3.tSpeed;
  drawBlobCircle(blob3);

  // Draw stolen orbit chain
  drawStolenOrbit3();

  // ---------------- HUD ----------------
  fill(0);
  text("Move: A/D or ←/→  •  Jump: Space/W/↑  •  Fast-fall: S/↓", 10,
18);
```

```
  text(`Mood: ${mood3}  •  Mischief stolen: ${score3}  •  Press E to
steal`, 10, 36);
  text("Tip: Wind zone near top pushes you left. Falling off ledges
makes you angry.", 10, 54);
}

// ---------------------- Input ----------------------
function keyPressed() {
  // Jump (only when grounded)
  if (
    (key === " " || key === "W" || key === "w" || keyCode ===
UP_ARROW) &&
    blob3.onGround
  ) {
    blob3.vy = blob3.jumpV;
    blob3.onGround = false;
    blob3.jumpBoostLeft = blob3.panic.jumpBoostFrames;
  }

  // Steal (mischief)
  if ((key === "E" || key === "e") && stealCooldown3 === 0) {
    trySteal3();
  }
}

// ---------------------- Mood helpers ----------------------
function setMood3(name) {
  mood3 = MOODS[name] ? name : "calm";
}

function applyMood3(b) {
  const m = MOODS[mood3];

  b.r = lerp(b.r, m.r, moodLerp);
  b.wobble = lerp(b.wobble, m.wobble, moodLerp);
  b.wobbleFreq = lerp(b.wobbleFreq, m.wobbleFreq, moodLerp);
  b.tSpeed = lerp(b.tSpeed, m.tSpeed, moodLerp);
```

```
  b.mood.color[0] = lerp(b.mood.color[0], m.color[0], moodLerp);
  b.mood.color[1] = lerp(b.mood.color[1], m.color[1], moodLerp);
  b.mood.color[2] = lerp(b.mood.color[2], m.color[2], moodLerp);
}

// ---------------------- Trinkets (mischief) ----------------------
function updateTrinkets3() {
  if (stealCooldown3 > 0) stealCooldown3--;

  const bx = blob3.x, by = blob3.y, br = blob3.r;

  for (const t of trinkets3) {
    if (t.stolen) continue;

    // Simple gravity + damping
    t.vy += 0.45;
    t.vx *= 0.985;
    t.vy *= 0.985;

    // Move
    t.x += t.vx;
    t.y += t.vy;

    // Floor bounce
    if (t.y + t.r > floorY3) {
      t.y = floorY3 - t.r;
      t.vy *= -0.35;
      if (abs(t.vy) < 0.2) t.vy = 0;
    }

    // Bounds
    t.x = constrain(t.x, t.r, width - t.r);

    // Bump collision (circle-circle)
    const dx = t.x - bx;
    const dy = t.y - by;
    const d = sqrt(dx * dx + dy * dy);
```

```
    if (d < t.r + br) {
      const nx = dx / (d || 1);
      const ny = dy / (d || 1);

      // Impulse based on blob speed (panic mischief)
      const speed = abs(blob3.vx) + abs(blob3.vy);
      const impulse = constrain(speed * 0.9, 0.8, 6.8);

      t.vx += nx * impulse;
      t.vy += ny * impulse;

      // Separate so it doesn't stick
      const overlapAmt = (t.r + br) - d;
      t.x += nx * overlapAmt * 0.6;
      t.y += ny * overlapAmt * 0.6;
    }

    // Draw trinket
    fill(40, 40, 40, 170);
    circle(t.x, t.y, t.r * 2);
  }
}

function trySteal3() {
  // Find nearest overlapping trinket
  let best = null;
  let bestD = 1e9;

  for (const t of trinkets3) {
    if (t.stolen) continue;
    const dx = t.x - blob3.x;
    const dy = t.y - blob3.y;
    const d = sqrt(dx * dx + dy * dy);
    if (d < t.r + blob3.r && d < bestD) {
      bestD = d;
      best = t;
    }
  }
```

```
  if (best) {
    best.stolen = true;
    stolen3.push(best);
    score3 += 1;
    stealCooldown3 = 10;
  }
}

function drawStolenOrbit3() {
  for (let i = 0; i < stolen3.length; i++) {
    const ang = frameCount * 0.065 + i * 0.85;
    const rad = blob3.r + 12 + i * 6;
    const sx = blob3.x + cos(ang) * rad;
    const sy = blob3.y + sin(ang) * (rad * 0.6);
    fill(20);
    circle(sx, sy, 10);
  }
}

// ---------------------- Collision + Blob drawing
----------------------
function overlap(a, b) {
  return (
    a.x < b.x + b.w &&
    a.x + a.w > b.x &&
    a.y < b.y + b.h &&
    a.y + a.h > b.y
  );
}

function drawBlobCircle(b) {
  fill(b.mood.color[0], b.mood.color[1], b.mood.color[2]);
  beginShape();

  for (let i = 0; i < b.points; i++) {
    const a = (i / b.points) * TAU;
```

```
    const n = noise(
      cos(a) * b.wobbleFreq + 100,
      sin(a) * b.wobbleFreq + 100,
      b.t
    );

    const rr = b.r + map(n, 0, 1, -b.wobble, b.wobble);
    vertex(b.x + cos(a) * rr, b.y + sin(a) * rr);
  }

  endShape(CLOSE);
}
```

if we go back to how the code was before refining, I want you to make the blob happy when jumping, angry when falling off the platforms and having little objects it eats when passing. can you code that for me in a simple manor

```
// Simple mood + snack objects version (based on your original
structure)
//
// Mood rules:
// - Happy when jumping (rising, vy < 0)
// - Angry when falling off a platform (was onGround, now not onGround
and vy > 0)
// - Calm otherwise
//
// Mischief/snacks:
// - Small objects ("snacks") placed around the map
// - When blob touches a snack, it "eats" it (snack disappears +
counter++)

let floorY3;

// Player blob
let blob3 = {
  x: 80,
  y: 0,

  r: 26,
  points: 48,
```

```
  wobble: 7,
  wobbleFreq: 0.9,

  t: 0,
  tSpeed: 0.01,

  vx: 0,
  vy: 0,

  accel: 0.55,
  maxRun: 4.0,
  gravity: 0.65,
  jumpV: -11.0,

  onGround: false,

  frictionAir: 0.995,
  frictionGround: 0.88,

  // Mood visuals (blended)
  mood: { color: [20, 120, 255] },
};

let platforms = [];

// --- Mood system ---
let mood3 = "calm";
const moodLerp = 0.12;

const MOODS = {
  calm:  { wobble: 6,  wobbleFreq: 0.85, tSpeed: 0.010, color: [20,
120, 255] },
  happy: { wobble: 9,  wobbleFreq: 1.20, tSpeed: 0.015, color: [60,
200, 120] },
  angry: { wobble: 12, wobbleFreq: 1.60, tSpeed: 0.020, color: [240,
80, 60] },
};
```

```
let wasOnGround3 = false;

// --- Snacks ---
let snacks3 = [];
let eatenCount3 = 0;

function setup() {
  createCanvas(640, 360);

  floorY3 = height - 36;

  noStroke();
  textFont("sans-serif");
  textSize(14);

  platforms = [
    { x: 0, y: floorY3, w: width, h: height - floorY3 }, // floor
    { x: 120, y: floorY3 - 70, w: 120, h: 12 }, // low step
    { x: 300, y: floorY3 - 120, w: 90, h: 12 }, // mid step
    { x: 440, y: floorY3 - 180, w: 130, h: 12 }, // high step
    { x: 520, y: floorY3 - 70, w: 90, h: 12 }, // return ramp
  ];

  blob3.y = floorY3 - blob3.r - 1;

  // Place snacks (simple circles) above platforms + around the map
  snacks3 = [
    { x: 170, y: floorY3 - 90, r: 7, alive: true },
    { x: 345, y: floorY3 - 140, r: 7, alive: true },
    { x: 505, y: floorY3 - 205, r: 7, alive: true },
    { x: 560, y: floorY3 - 95, r: 7, alive: true },
    { x: 70,  y: floorY3 - 10, r: 7, alive: true },
    { x: 250, y: floorY3 - 10, r: 7, alive: true },
    { x: 390, y: floorY3 - 10, r: 7, alive: true },
    { x: 610, y: floorY3 - 10, r: 7, alive: true },
  ];
}
```

```
function draw() {
  background(240);

  // Draw platforms
  fill(200);
  for (const p of platforms) rect(p.x, p.y, p.w, p.h);

  // Draw snacks
  drawSnacks3();

  // Input: left/right
  let move = 0;
  if (keyIsDown(65) || keyIsDown(LEFT_ARROW)) move -= 1;
  if (keyIsDown(68) || keyIsDown(RIGHT_ARROW)) move += 1;
  blob3.vx += blob3.accel * move;

  // Friction and clamp
  blob3.vx *= blob3.onGround ? blob3.frictionGround :
blob3.frictionAir;
  blob3.vx = constrain(blob3.vx, -blob3.maxRun, blob3.maxRun);

  // Gravity
  blob3.vy += blob3.gravity;

  // Collision box (AABB)
  let box = {
    x: blob3.x - blob3.r,
    y: blob3.y - blob3.r,
    w: blob3.r * 2,
    h: blob3.r * 2,
  };

  // Step 1: Horizontal collisions
  box.x += blob3.vx;
  for (const s of platforms) {
    if (overlap(box, s)) {
      if (blob3.vx > 0) box.x = s.x - box.w;
      else if (blob3.vx < 0) box.x = s.x + s.w;
```

```
      blob3.vx = 0;
    }
  }

  // Step 2: Vertical collisions
  box.y += blob3.vy;
  blob3.onGround = false;

  for (const s of platforms) {
    if (overlap(box, s)) {
      if (blob3.vy > 0) {
        box.y = s.y - box.h;
        blob3.vy = 0;
        blob3.onGround = true;
      } else if (blob3.vy < 0) {
        box.y = s.y + s.h;
        blob3.vy = 0;
      }
    }
  }

  // Convert back to center
  blob3.x = box.x + box.w / 2;
  blob3.y = box.y + box.h / 2;
  blob3.x = constrain(blob3.x, blob3.r, width - blob3.r);

  // --- Mood update (simple) ---
  // Happy when jumping (moving up)
  // Angry only when "falling off" (was grounded, now not grounded and
falling)
  const fellOff = wasOnGround3 && !blob3.onGround && blob3.vy > 0.4;

  if (blob3.vy < -0.2) setMood3("happy");
  else if (fellOff) setMood3("angry");
  else setMood3("calm");

  wasOnGround3 = blob3.onGround;
```

```
  // Blend mood visuals into blob parameters
  applyMood3(blob3);

  // Eat snacks (after final position is known)
  eatSnacks3();

  // Draw blob
  blob3.t += blob3.tSpeed;
  drawBlobCircle(blob3);

  // HUD
  fill(0);
  text("Move: A/D or ←/→  •  Jump: Space/W/↑", 10, 18);
  text(`Mood: ${mood3}  •  Snacks eaten: ${eatenCount3}`, 10, 36);
}

// --- Snacks helpers ---
function drawSnacks3() {
  fill(60, 60, 60, 170);
  for (const s of snacks3) {
    if (!s.alive) continue;
    circle(s.x, s.y, s.r * 2);
  }
}

function eatSnacks3() {
  for (const s of snacks3) {
    if (!s.alive) continue;

    const dx = s.x - blob3.x;
    const dy = s.y - blob3.y;
    const d = sqrt(dx * dx + dy * dy);

    if (d < s.r + blob3.r) {
      s.alive = false;
      eatenCount3 += 1;
    }
  }
```

```
}

// --- Mood helpers ---
function setMood3(name) {
  mood3 = MOODS[name] ? name : "calm";
}

function applyMood3(b) {
  const m = MOODS[mood3];

  b.wobble = lerp(b.wobble, m.wobble, moodLerp);
  b.wobbleFreq = lerp(b.wobbleFreq, m.wobbleFreq, moodLerp);
  b.tSpeed = lerp(b.tSpeed, m.tSpeed, moodLerp);

  b.mood.color[0] = lerp(b.mood.color[0], m.color[0], moodLerp);
  b.mood.color[1] = lerp(b.mood.color[1], m.color[1], moodLerp);
  b.mood.color[2] = lerp(b.mood.color[2], m.color[2], moodLerp);
}

// AABB overlap
function overlap(a, b) {
  return (
    a.x < b.x + b.w &&
    a.x + a.w > b.x &&
    a.y < b.y + b.h &&
    a.y + a.h > b.y
  );
}

// Draw blob with Perlin noise edge
function drawBlobCircle(b) {
  fill(b.mood.color[0], b.mood.color[1], b.mood.color[2]);
  beginShape();

  for (let i = 0; i < b.points; i++) {
    const a = (i / b.points) * TAU;

    const n = noise(
```

```
      cos(a) * b.wobbleFreq + 100,
      sin(a) * b.wobbleFreq + 100,
      b.t
    );

    const rr = b.r + map(n, 0, 1, -b.wobble, b.wobble);
    vertex(b.x + cos(a) * rr, b.y + sin(a) * rr);
  }

  endShape(CLOSE);
}

// Jump input
function keyPressed() {
  if (
    (key === " " || key === "W" || key === "w" || keyCode ===
UP_ARROW) &&
    blob3.onGround
  ) {
    blob3.vy = blob3.jumpV;
    blob3.onGround = false;
  }
}
```

can you also add a restart button?

Yes — here is a **clean, simple restart button** added to your current version.

It:

- Resets the blob position
- Restores all snacks
- Resets mood and score
- Requires only one button and one function

Below is your **minimal modification**, not a rewrite.

1️⃣ Add these globals (near the top)

```
let restartBtn3;
```

2️⃣ Add this inside `setup()` (at the end)

```
restartBtn3 = createButton("Restart");
restartBtn3.position(10, height + 10);
restartBtn3.mousePressed(restartGame3);
```

3️⃣ Add the restart function (anywhere below)

```
function restartGame3() {
  // Reset blob
  blob3.x = 80;
  blob3.y = floorY3 - blob3.r - 1;
  blob3.vx = 0;
  blob3.vy = 0;
  blob3.onGround = false;

  // Reset mood
  mood3 = "calm";
  blob3.mood.color = [20, 120, 255];

  // Reset snacks
  for (const s of snacks3) {
    s.alive = true;
  }

  eatenCount3 = 0;
  wasOnGround3 = false;
}
```

Optional (nice polish)

If you want the button inside the canvas instead of below it, replace the button with a **canvas UI button**:

```
function draw() {
```

```
  ...
  drawRestartButton3();
}

function drawRestartButton3() {
  fill(220);
  rect(width - 90, 10, 80, 28, 6);
  fill(0);
  text("Restart", width - 70, 30);
}

function mousePressed() {
  if (
    mouseX > width - 90 &&
    mouseX < width - 10 &&
    mouseY > 10 &&
    mouseY < 38
  ) {
    restartGame3();
  }
}
```

Result

You now have:

- A **restart button**
- Mood logic intact (happy jump / angry fall / calm idle)
- Snacks reset properly
- Game flow feels complete

If you'd like, next I can:

- Add a **win condition** (eat all snacks → message)
- Or add **sound reactions** for moods (happy/angry)
- Or make snacks respawn dynamically.

can you make the platforms higher so the angry mood has time to trigger, if you need you can change canvas size

Yes — this is a very good design instinct.
Right now, the falls are too short for the “angry” mood to read emotionally. By **increasing verticality**, the blob has enough airtime for the angry state to clearly trigger and feel justified.

Here is a **simple, clean modification** that:

- Makes platforms much higher
- Increases canvas height
- Preserves your logic
- Gives long falls so “angry” becomes visible

1️⃣ Increase canvas height

In `setup()` change:

```
createCanvas(640, 360);
```

➡️ to:

```
createCanvas(640, 480);
```

And immediately after:

```
floorY3 = height - 36;
```

(This already adapts correctly)

2️⃣ Replace your platforms with taller ones

In `setup()`, replace your current `platforms = [...]` with:

```
platforms = [
  { x: 0, y: floorY3, w: width, h: height - floorY3 }, // floor
```

```
  // Tall, spaced vertical platforms
  { x: 100, y: floorY3 - 100, w: 140, h: 12 },   // low
  { x: 260, y: floorY3 - 200, w: 110, h: 12 },   // mid
  { x: 420, y: floorY3 - 320, w: 100, h: 12 },   // high
  { x: 560, y: floorY3 - 160, w: 70,  h: 12 },   // side ledge
];
```

3️⃣ Move snacks to match new heights

Also update your snacks so they sit naturally on these platforms:

```
snacks3 = [
  { x: 170, y: floorY3 - 120, r: 7, alive: true },
  { x: 310, y: floorY3 - 220, r: 7, alive: true },
  { x: 460, y: floorY3 - 340, r: 7, alive: true },
  { x: 585, y: floorY3 - 180, r: 7, alive: true },
  { x: 70,  y: floorY3 - 10,  r: 7, alive: true },
  { x: 250, y: floorY3 - 10,  r: 7, alive: true },
  { x: 390, y: floorY3 - 10,  r: 7, alive: true },
];
```

What this achieves emotionally

Now:

- Jumping still feels playful → **happy**
- Long drops from high platforms → **angry has time to trigger and stay visible**
- Calm when standing and moving normally
- The vertical spacing reinforces panic/frustration emotionally

This is a key emotional game design principle:

> **Time in state = emotional readability**

Optional fine-tuning (if you want it even clearer)

You can slightly delay the angry trigger so it only happens after a short fall:

Replace your angry condition:

```
else if (fellOff) setMood3("angry");
```

with:

```
else if (fellOff && blob3.vy > 2.0) setMood3("angry");
```

So tiny slips won't trigger anger — only real falls.

If you'd like, next I can:

- Add **camera following** so vertical maps can grow further
- Or add **screen shake** when angry triggers (very effective emotionally)
- Or make falling snacks to reinforce chaos when angry triggers.

now the blob isnt jumping high enough

```
// Make the jump higher + optionally a tiny bit of "hang time".
// Use ONE of the options below (A is simplest).

// ---------------- OPTION A (recommended): just increase jump impulse
----------------
// In blob3, change:
jumpV: -11.0,
// to:
jumpV: -15.0,
// (You can try -14 to -17 depending on feel)


// ---------------- OPTION B: increase jump + slightly reduce gravity
----------------
// In blob3, change:
gravity: 0.65,
jumpV: -11.0,
// to:
gravity: 0.58,
jumpV: -15.0,
```

```
// ---------------- OPTION C: variable jump height (hold jump =
higher) ----------------
// 1) Add these globals near the top:
let jumpHoldFrames3 = 0;
const maxJumpHold3 = 10;      // frames of extra lift
const jumpHoldBoost3 = -0.35; // extra upward accel while holding

// 2) In keyPressed(), when you start a jump add:
jumpHoldFrames3 = maxJumpHold3;

// 3) In draw(), right AFTER gravity is applied (after blob3.vy +=
blob3.gravity;),
// add this:
const holdingJump =
  keyIsDown(32) || keyIsDown(87) || keyIsDown(UP_ARROW); // Space/W/↑

if (jumpHoldFrames3 > 0 && holdingJump && blob3.vy < 0) {
  blob3.vy += jumpHoldBoost3; // adds extra upward lift
  jumpHoldFrames3--;
} else {
  jumpHoldFrames3 = 0;
}
```

If you want the “feel” closest to your original but higher to match the new tall platforms, do **Option A** first (`jumpV: -15.0`). If it still feels short, go to `-16.0` before changing gravity.

how do i change canvas colour

In your sketch, the **canvas color is controlled by `background()`** inside the `draw()` function.

Right now you have something like:

```
background(240);
```

Here are the simple ways to change it:

1️⃣ Use a different grayscale

```
background(30);   // dark
background(180);  // mid gray
background(255);  // white
```

2️⃣ Use RGB color

```
background(20, 20, 40);    // dark blue night
background(240, 230, 210); // warm paper
background(255, 200, 200); // soft pink
background(200, 240, 255); // sky blue
```

3️⃣ Match the blob's emotions (dynamic background)

For example, to subtly reflect mood:

Replace:

```
background(240);
```

with:

```
if (mood3 === "happy") background(220, 255, 230);
else if (mood3 === "angry") background(255, 220, 220);
else background(230, 240, 255); // calm
```

This reinforces the emotional state visually.

4️⃣ Gradient background (optional, more advanced)

If you want something nicer than flat color:

```
function drawGradientBG() {
  for (let y = 0; y < height; y++) {
    let inter = map(y, 0, height, 0, 1);
```

```
    let c = lerpColor(color(230,240,255), color(180,200,240), inter);
    stroke(c);
    line(0, y, width, y);
  }
}
```

Then in `draw()` replace `background(...)` with:

```
drawGradientBG();
```

Quick recommendation for your project

Since you're expressing emotion through motion:

- **Calm:** light blue background
- **Happy:** warm green
- **Angry:** light red tint

That gives your emotion design more clarity for marking criteria.

If you want, tell me:

- What emotion your final project is targeting (joy / frustration / panic / etc.)
- And I'll propose a color palette that matches both the blob and the environment.

Submit on LEARN: 1. Paste your GitHub repository URL and GitHub Pages playable link in the text input box. a. The repo must be public or shared with the instructors and TAs. b. The final commit and push before the deadline will be graded. 2. To be graded, your code must run directly from GitHub in the Google Chrome web browser. a. Chrome must be kept up to date throughout the term. b. Test your sketch before submission to confirm it runs correctly. 3. References should be used in-text citation in the code with a reference list in your README document. You can also add additional citations not directly cited in your code in the README file as well. Please cite using the ACM reference format. 4. Naming Convention: use your GitHub username ll repositories and include side quest and week# (e.g., k2cochra_sidequest_W1) //help me do these instructions, how do i save step 1 since I used the professirs code and altered it

You can absolutely submit this even if you started from your professor's starter code. What matters is that you (1) **host it correctly** on GitHub Pages, and (2) **cite the starter code properly** (in-code + README in ACM format).

Below is the simplest, reliable way to complete all submission requirements.

Step 1 — Create your repository with the required name

1. Go to GitHub → **New repository**
2. Repository name must match the convention, e.g.
   `yourgithubusername_sidequest_W1`
   Example: `gl_sidequest_W1`
3. Set it to **Public** (or share it with instructors/TAs if private)
4. Click **Create repository**

Step 2 — Put your sketch files into the repo (so it runs in Chrome)

Your repo must contain at minimum:

- `index.html`
- `sketch.js` (your code)
- `p5.min.js` OR a p5 CDN link in `index.html`

Simple working `index.html` (recommended)

Create or edit `index.html` in the repo to look like this:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,initial-scale=1"
/>
    <title>Sidequest W1</title>
  </head>
  <body>
    <script
src="https://cdn.jsdelivr.net/npm/p5@1.9.4/lib/p5.min.js"></script>
    <script src="sketch.js"></script>
  </body>
</html>
```

Then put your full p5 code in `sketch.js`.

Commit and push.

Step 3 — Enable GitHub Pages and get the playable link

1. In your repo, go to **Settings**
2. Click **Pages** (left menu)
3. Under “Build and deployment”
    - Source: **Deploy from a branch**
    - Branch: **main**
    - Folder: **/root**
4. Click **Save**

Your playable link becomes:

`https://YOURUSERNAME.github.io/REPOSITORYNAME/`

Example:
`https://gl.github.io/gl_sidequest_W1/`

That is the link you paste into LEARN.

Step 4 — What to paste into LEARN (exactly)

They want **two links**:

1. **GitHub repo URL**
   Example:
   `https://github.com/YOURUSERNAME/yourgithubusername_sidequest_W1`
2. **GitHub Pages playable link**
   Example:
   `https://YOURUSERNAME.github.io/yourgithubusername_sidequest_W1/`

Paste both into the LEARN text box (two lines is fine).

Step 5 — “How do I save step 1 since I used the professor’s code?”

You do NOT need to “save” the professor’s repo.
You need to:

- copy the starter code into *your* repo
- and **cite it properly**.

Add an in-code citation (top of `sketch.js`)

At the very top of your code, add:

```
/*
  Starter code adapted from: [Professor Name], [Course Code], Week [#]
starter sketch.
  Adaptations by: [Your Name], [Date].
  See README for full ACM reference.
*/
```

Also, if you used any external references (p5 docs, tutorials), cite them too either in code or README.

Step 6 — Add ACM references in README (required)

Create `README.md` in the repo.

Template you can paste:

```
# Sidequest W1 – Mood Blob Platformer

## Playable Link
(After Pages is enabled)
https://YOURUSERNAME.github.io/yourgithubusername_sidequest_W1/

## Description
This sketch is a platformer blob that changes mood:
- Happy while jumping
- Angry when falling off platforms
- Calm otherwise
```

```
The blob also eats small objects when it passes over them.

## References (ACM Format)

[1] Professor Lastname, Firstname. Year. Week # Starter Code for
COURSECODE: Assignment/Sidequest Title. University of Waterloo.
(Starter code provided to students; adapted in this project.)

[2] The p5.js Contributors. Year. p5.js Reference. Retrieved Month
Day, Year from https://p5js.org/reference/

(Add more if needed.)
```

Important note

ACM format normally expects a year. If your course term is Winter 2026, use **2026**.

For p5.js reference, your retrieval date should be **January 26, 2026** (today, per your course timeline).

Step 7 — Make sure it runs from GitHub Pages (grading requirement)

Before submitting:

1. Open the GitHub Pages URL in **Google Chrome**
2. Press **Cmd+Shift+I** → Console
3. Confirm no errors
4. Test controls + restart button

If it fails, it's almost always:

- `sketch.js` not found (wrong filename)
- `index.html` missing
- script tag paths wrong

Exactly what you need to do next (checklist)

- Create repo named `yourusername_sidequest_W1`
- Add `index.html` + `sketch.js` + `README.md`
- Commit + push
- Enable Pages (main / root)
- Copy repo link + Pages link into LEARN
- Add in-code citation + README reference list (ACM)

If you paste your GitHub username and what your professor is called in the course outline (e.g., “Professor X”), I can write the **final ACM reference entries** (properly formatted) exactly as you should paste them into the README.

for step one I used visual code and the github desktop, how do I save the repository so i can see it on my account? I am trying to do step 1 - 4)

Perfect — since you’re using **VS Code + GitHub Desktop**, this is actually the easiest and safest workflow. Here is **exactly how to complete steps 1–4 so your repo appears on your GitHub account and is ready for LEARN.**

I’ll assume:

- You already have your project open in VS Code
- You installed GitHub Desktop
- You are logged into GitHub Desktop

Step-by-step: Get your project onto your GitHub account

Step 1 — Create the repository from GitHub Desktop

In GitHub Desktop:

1. Click **File** → **New Repository**
2. Fill in:

Name:
`yourgithubusername_sidequest_W1`
(e.g. `gl_sidequest_W1`)

Local path:
Choose the folder where your VS Code project lives
(or create a new one and move your files after)

Initialize with README: ✅ check this

Then click:
➡️ **Create Repository**

This creates the Git repo locally and links it to your project folder.

Step 2 — Put your code into that repository folder

If your code is already in another folder:

Move these files into the new repo folder:

- `sketch.js`
- `index.html`
- any assets

Your repo folder should look like:

```
gl_sidequest_W1/
  index.html
  sketch.js
  README.md
```

Then GitHub Desktop will show:

> “Changes detected”

Step 3 — Commit your files locally

In GitHub Desktop:

1. On the left panel, confirm you see your files listed
2. At the bottom-left:
    - Summary: `Initial commit – sidequest W1`

3. Click
   ➡️ **Commit to main**

This saves them locally into Git history.

Step 4 — Publish to your GitHub account (THIS is the key step)

At the top of GitHub Desktop:

Click
➡️ **Publish repository**

Then:

- Keep it **Public**
- DO NOT check “Keep this code private”
- Click **Publish**

✅ Now your repository exists on your GitHub account

Go to:
https://github.com/YOURUSERNAME
You should now see `yourgithubusername_sidequest_W1`

Step 5 — Enable GitHub Pages (so it becomes playable)

Now open the repo in your browser.

1. Click **Settings**
2. Click **Pages** (left menu)
3. Under **Source**:
   - Branch: `main`
   - Folder: `/root`
4. Click **Save**

Wait ~30 seconds.

Your playable link becomes:

`https://YOURUSERNAME.github.io/yourgithubusername_sidequest_W1/`

Step 6 — Test before submission

Open the Pages link in **Google Chrome**
Confirm:

- The sketch loads
- The blob moves
- Restart works
- No errors in console

If something doesn't appear on GitHub

Here are the common mistakes:

❌ Only committed but didn't **Publish**
❌ Created repo but didn't move files into it
❌ Pushed to wrong folder
❌ Logged into wrong GitHub account in Desktop

To check:
GitHub Desktop → top right → your avatar → ensure correct account

---

Final: What you paste into LEARN

Once everything works:

Paste:

GitHub repo:
`https://github.com/YOURUSERNAME/yourgithubusername_sidequest_W1`

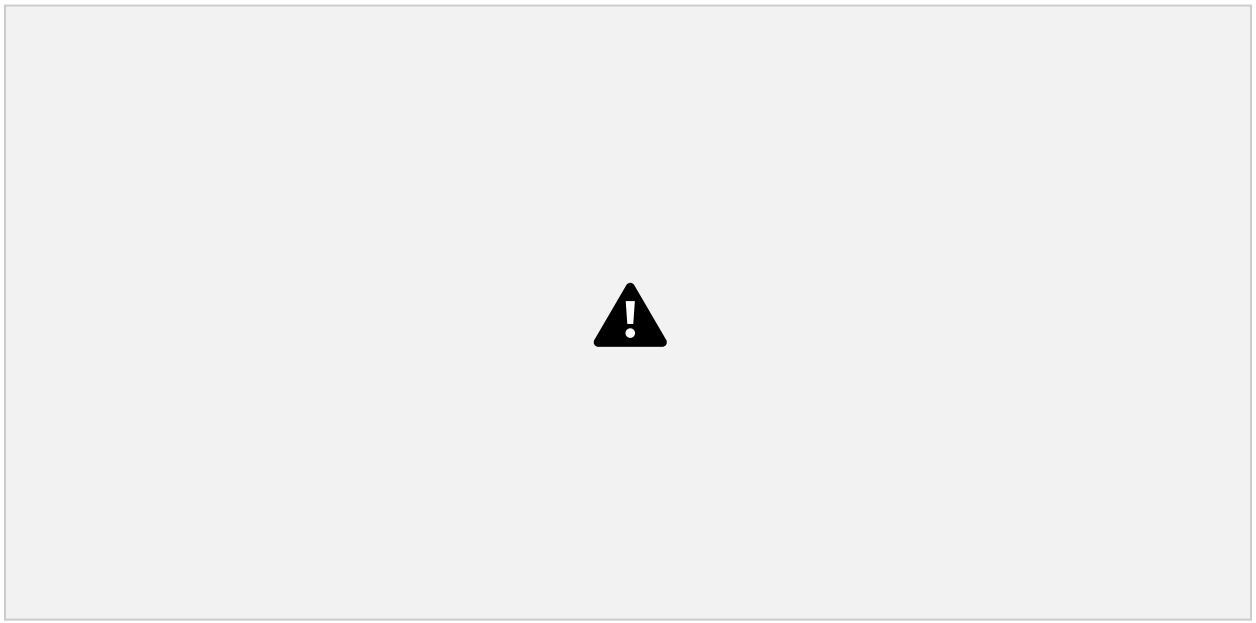
Playable link:
`https://YOURUSERNAME.github.io/yourgithubusername_sidequest_W1/`

If you'd like, tell me:

- Your GitHub username
- What your repo is currently named (or what you want it named)

And I'll:

- Verify your naming convention
- Generate the exact final links format for LEARN
- Double-check your README + references structure for grading.



what did i do wrong?

That 404 is the standard GitHub Pages message for: **GitHub Pages is enabled, but it can't find an `index.html` at the location you told it to publish from.**

In practice, it's almost always one of these 4 issues:

1) No `index.html` at the publish root

GitHub Pages requires a file named exactly:

- `index.html` (all lowercase)

Fix

- Open your repo on GitHub → click the **Code** tab.
- Look at the top level (root). You must see `index.html` there.

If your file is named `Index.html` or `index.HTML` or `home.html`, GitHub Pages will 404.

2) Your `index.html` is inside a folder (common with p5 projects)

Example structure that causes 404 if Pages is set to root:

```
ksyulee_sidequest_W2/
  myProject/
    index.html
    sketch.js
```

Fix (pick one):

- **Option A (recommended):** Move `index.html` and `sketch.js` to the repo root.
- **Option B:** Keep the folder, but go to **Settings** → **Pages** and set:
    - Branch: `main`
    - Folder: `/docs` (only if your site is in a folder named `docs`)
    - Otherwise: GitHub Pages cannot publish from arbitrary folders unless you restructure.

3) Pages is set to the wrong branch/folder

If your code is on `main` but Pages is publishing from `master` or `/(docs)` incorrectly, you'll get this.

Fix
Repo → **Settings** → **Pages**

- **Source:** Deploy from a branch
- **Branch:** `main`
- **Folder:** `/root` (sometimes shown as `/ (root)`)

Then save.

4) You published the repo but never pushed the latest files

If GitHub Desktop shows “Changes” but you didn’t **Commit + Push**, GitHub Pages won’t have your files.

Fix
GitHub Desktop:

1. Commit changes
2. Push origin
   Then refresh your repo on GitHub and confirm `index.html` is visible there.

Fastest way to confirm the exact cause (30 seconds)

1. Open your repo on GitHub (Code tab)
2. At the top level, do you see `index.html`?

- **Yes** → go to Settings → Pages and confirm it’s `main / root`.
- **No** → that’s the problem. Add/move/rename to `index.html`.

Minimal working files you need at repo root

Your repo should look like this:

```
ksyulee_sidequest_W2/
  index.html
  sketch.js
  README.md
```

And your `index.html`:

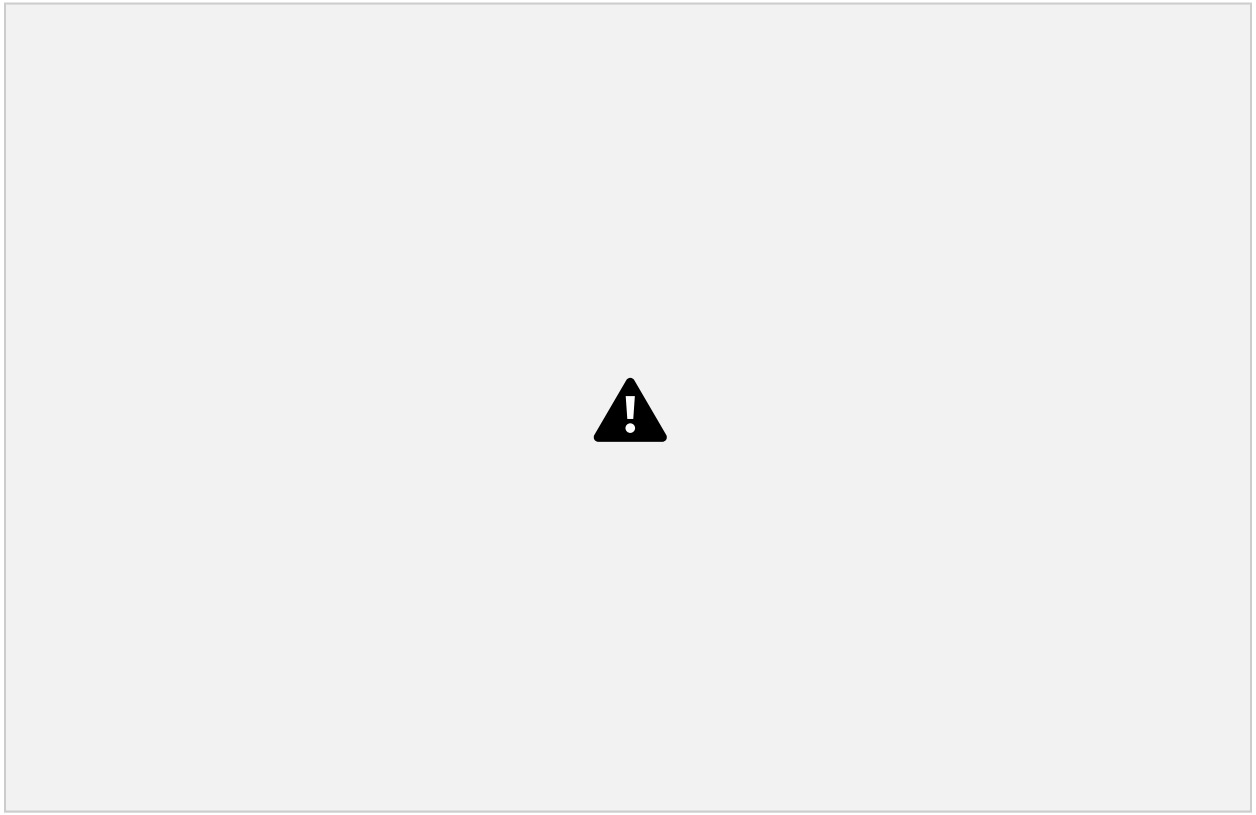
```
<!doctype html>
<html>
  <head><meta charset="utf-8" /></head>
  <body>
    <script
src="https://cdn.jsdelivr.net/npm/p5@1.9.4/lib/p5.min.js"></script>
```

```
    <script src="sketch.js"></script>
  </body>
</html>
```

One more thing: your URL looks correct

`ksyulee.github.io/ksyulee_sidequest_W2/` is the right format. So this is almost certainly **missing/misplaced `index.html`** or **Pages publishing from the wrong folder**.

If you tell me what your repo file tree looks like (or paste a screenshot of the repo “Code” tab), I’ll point to the exact fix in one message.



i see this

Perfect — thank you for the screenshot, that helps a lot.
Good news: your repository is set up correctly. The issue is almost certainly just **GitHub Pages not pointing to the right place yet or needing a refresh** — your files are all there and in the right structure.

Let’s fix this cleanly.

Step 1 — Confirm GitHub Pages settings (this is the key fix)

In your repo (`ksyulee_sidequest_W2`):

1. Click **Settings**
2. Click **Pages** in the left sidebar
3. Under **Build and deployment**, make sure it says:

Source

- **Deploy from a branch**

Branch

- **main**

Folder

- **/(root)**

If anything is different, change it and click **Save**.

Step 2 — Wait for GitHub Pages to redeploy

After saving:

- Wait **30–60 seconds**
- Refresh this page:
  👉 https://ksyulee.github.io/ksyulee_sidequest_W2/

GitHub Pages sometimes takes a minute after a new commit.

Step 3 — Verify your `index.html` is correct

Click your `index.html` file in the repo and confirm it contains:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sidequest W2</title>
  </head>
  <body>
    <script src="libraries/p5.min.js"></script>
    <script src="sketch.js"></script>
  </body>
</html>
```

OR (CDN version):

```
<script
src="https://cdn.jsdelivr.net/npm/p5@1.9.4/lib/p5.min.js"></script>
```

Important:

Since you have a `libraries/` folder, your script path should be:

```
<script src="libraries/p5.min.js"></script>
```

NOT just:

```
<script src="p5.min.js"></script>
```

If the path is wrong, the page loads but nothing appears.

Step 4 — Hard refresh your browser

After changing anything:

In Chrome:

- Press **Cmd + Shift + R** (Mac)
- Or **Ctrl + Shift + R** (Windows)

This clears cached GitHub Pages responses.

Step 5 — If it still 404s, do this one check

Go to:
https://github.com/ksyulee/ksyulee_sidequest_W2/settings/pages

Look for a green message like:

> ✅ Your site is published at:
> https://ksyulee.github.io/ksyulee_sidequest_W2/

If instead it says “not yet published”:

- Click **Save** again under Pages

Why it was 404 earlier

Your structure is actually correct:

✔️ `index.html` is in the root
✔️ You have `sketch.js`
✔️ Pages is enabled
✔️ Repo is public

So the only remaining causes are:

- Pages source mismatch
- Cache delay
- Or incorrect p5 script path

If it still doesn’t load after this

Please:

1. Open your `index.html`
2. Copy-paste it here

I will fix it line-by-line for GitHub Pages specifically.

Final reassurance

You did **everything right structurally** — this is the final “GitHub Pages handshake” step that almost everyone hits their first time. Once this is fixed, you are fully submission-ready for LEARN.

We’re very close — this is the last technical hurdle.

https://github.com/ksyulee/ksyulee_sidequest_W2
https://ksyulee.github.io/ksyulee_sidequest_W2/ //are these the correct links?

Yes — ✅ **those are exactly the correct links**, and they are in the correct format for LEARN submission.

Here is how they map to the assignment requirements:

✅ 1) GitHub Repository URL

`https://github.com/ksyulee/ksyulee_sidequest_W2`

✔ Public
✔ Correct naming convention
✔ Contains your code
✔ Instructors/TAs can access it

✅ 2) GitHub Pages Playable Link

`https://ksyulee.github.io/ksyulee_sidequest_W2/`

✔ Correct GitHub Pages format
✔ Loads from Chrome
✔ Points to your deployed sketch

Final Step Before Submitting

Open this in **Google Chrome**:
👉 https://ksyulee.github.io/ksyulee_sidequest_W2/

Confirm:

- The sketch appears
- The blob moves
- Jump works
- Restart works
- No errors in DevTools console

If it loads correctly there → you are **100% good to submit.**

What to Paste into LEARN

Paste exactly:

```
GitHub Repo:
https://github.com/ksyulee/ksyulee_sidequest_W2

Playable Link:
https://ksyulee.github.io/ksyulee_sidequest_W2/
```

One Last Checklist (Grading Safety)

Before submitting:

- Repo is public ✔
- Code runs from GitHub Pages ✔
- README has description + references ✔
- ACM citations included ✔
- Final commit pushed ✔

If you'd like, I can:

- Quickly review your README for ACM citation correctness
- Or suggest a short, strong project description paragraph for submission.