

GBDA 302 – Global Digital Project 2
Jowan Manjooran Jomon jmanjoor 21105035
Side Quest Week 2 - Process & Decision Documentation

1. What I was trying to do

I was trying to redesign my blob character's movement and environment to express the emotion of panic, and to add a small mischief mechanic where the blob can bump into and steal objects in a small map.

2. One small decision or change I made

I decided to make the blob feel harder to control by reducing ground friction, increasing camera shake, and making movement more slippery and jittery as the panic value increases.

3. Evidence of the change

I modified the movement variables and added a panic system in the code that affects camera shake, blob wobble, and how the blob interacts with objects in the environment.

4. GenAI Use

I used GenAI for this Side Quest.

I used GenAI (ChatGPT and Gemini) to help extend my existing p5.js code by adding a panic system, camera shake, and a mischief object interaction mechanic. Gemini was also used to help debug collision boundaries and fix movement issues. In addition, I used ChatGPT to refine my technical writing so that it would be clearer and easier for the user to understand.

5. Gen AI Transcript.

Prompt

```
// 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();
  textAlign("center");
  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
*/

```

Response

```

// --- STEP 1: Move horizontally, then resolve X collisions ---
box.x += blob3.vx;

for (const s of platforms) {
  if (!overlap(box, s)) continue;

  // penetration distances along X
  const pushLeft = (s.x - (box.x + box.w));    // negative/0 if box is inside
  const pushRight = ((s.x + s.w) - box.x);      // positive/0 if box is inside

  // Choose the smallest magnitude push that separates along X
  // If moving right, prefer pushing left; if moving left, prefer pushing right.
  if (blob3.vx > 0) {

```

```

    box.x += pushLeft; // push box left
} else if (blob3.vx < 0) {
    box.x += pushRight; // push box right
} else {
    // If vx is 0, pick the smaller absolute push
    box.x += (abs(pushLeft) < abs(pushRight)) ? pushLeft : pushRight;
}

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)) continue;

    const pushUp = (s.y - (box.y + box.h));
    const pushDown = ((s.y + s.h) - box.y);

    if (blob3.vy > 0) {
        // falling
        box.y += pushUp;
        blob3.vy = 0;
        blob3.onGround = true;
    } else if (blob3.vy < 0) {
        // rising
        box.y += pushDown;
        blob3.vy = 0;
    } else {
        box.y += (abs(pushUp) < abs(pushDown)) ? pushUp : pushDown;
    }
}

```

Prompt

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.

```
// Emotion system: PANIC
let panic = 0; // 0..1
let panicVel = 0;
let cameraShake = 0;

// Jump feel upgrades
let coyoteFrames = 0;
let jumpBufferFrames = 0;

// Mischief objects
let objects = [] // {x,y,w,h,vx,vy,type,stolen}
let held = null; // currently stolen object (reference)

// 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();
  textAlign("center");
  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
*/
can you add 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.

Response

```

// Emotion system: PANIC
let panic = 0; // 0..1
let panicVel = 0;
let cameraShake = 0;

// Jump feel upgrades
let coyoteFrames = 0; // frames of grace after leaving ground
let jumpBufferFrames = 0; // frames of buffered jump input

// Mischief objects
let objects = []; // {x,y,w,h,vx,vy,type,stolen,spin}
let held = null; // currently stolen object (reference)

// 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,
points: 48,
wobble: 7,
wobbleFreq: 0.9,

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

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

// Movement tuning (base values; panic modifies them)
accel: 0.55,
maxRun: 4.0,
gravity: 0.65,
jumpV: -11.0,

// State
onGround: false,

// Friction
frictionAir: 0.995,
frictionGround: 0.88,
};

// Solid platforms
let platforms = [];

// Simple map bounds (walls)
let walls = [] // AABBs

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

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

```

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

// Small “map”: floor + steps + a couple ledges
platforms = [
  { x: 0, y: floorY3, w: width, h: height - floorY3 }, // floor
  { x: 120, y: floorY3 - 70, w: 120, h: 12 },
  { x: 300, y: floorY3 - 120, w: 90, h: 12 },
  { x: 440, y: floorY3 - 180, w: 130, h: 12 },
  { x: 520, y: floorY3 - 70, w: 90, h: 12 },
  { x: 40, y: floorY3 - 140, w: 70, h: 12 },
];
}

// Map walls to make it feel like a contained little room
walls = [
  { x: -20, y: 0, w: 20, h: height },      // left wall
  { x: width, y: 0, w: 20, h: height },     // right wall
  { x: 0, y: -20, w: width, h: 20 },        // ceiling-ish (keeps shake from showing
void)
];

// Spawn some “mischief” objects
objects = [
  makeObj(180, floorY3 - 18, 18, 18, "cube"),
  makeObj(240, floorY3 - 18, 14, 22, "can"),
  makeObj(340, floorY3 - 138, 20, 14, "book"),
  makeObj(500, floorY3 - 88, 16, 16, "cube"),
  makeObj(470, floorY3 - 198, 22, 12, "bar"),
];
}

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

function draw() {
  // --- PANIC “heartbeat” background ---
  // Panic adds subtle strobing + vignette, plus camera shake.
  updatePanic();
}

```

```
background(240);

// Camera shake based on panic
const shake = cameraShake * (0.5 + panic);
const sx = random(-shake, shake);
const sy = random(-shake, shake);

push();
translate(sx, sy);

// --- Draw environment ---
drawEnvironment();

// --- Update physics + controls ---
updateMovementAndCollisions();

// --- Mischief objects physics + interactions ---
updateObjects();

// --- Draw the animated blob ---
blob3.t += blob3.tSpeed * (1 + panic * 0.8);
drawBlobCircle(blob3);

// Draw held object if any
drawHeld();

pop();

// --- HUD (not shaken) ---
drawHUD();
}

// ----- PANIC SYSTEM -----

function updatePanic() {
  // Target panic rises if you're holding something (mischief = guilty)
  // and if you're moving fast (out of control feel).
  const speed = abs(blob3.vx) + abs(blob3.vy) * 0.15;
  const target = constrain(
```

```

(held ? 0.55 : 0.15) + speed * 0.05,
0,
1
);

// Smooth panic (spring-ish)
panicVel += (target - panic) * 0.05;
panicVel *= 0.86;
panic += panicVel;
panic = constrain(panic, 0, 1);

// Camera shake grows with panic
cameraShake = lerp(cameraShake, 0.8 + panic * 6.0, 0.12);
}

// ----- ENVIRONMENT -----

function drawEnvironment() {
  // Platforms pulse slightly with panic (visual tension)
  for (const p of platforms) {
    const wob = (noise(p.x * 0.01, p.y * 0.01, frameCount * 0.02) - 0.5) * 6 * panic;
    fill(200 + wob, 200 - wob * 0.6, 200 - wob * 0.6);
    rect(p.x, p.y, p.w, p.h);
  }

  // Walls (invisible collisions, but draw faint boundaries)
  fill(230);
  rect(0, 0, width, 10); // top trim
  rect(0, 0, 10, height); // left trim
  rect(width - 10, 0, 10, height); // right trim

  // Panic vignette overlay (very light)
  // (Do it here so it is affected by camera shake — feels “in your head”.)
  push();
  noStroke();
  const v = panic * 90;
  fill(0, v);
  rect(-50, -50, width + 100, 50);      // top
  rect(-50, height, width + 100, 50);    // bottom
  rect(-50, -50, 50, height + 100);      // left
}

```

```

rect(width, -50, 50, height + 100);      // right
pop();
}

// ----- MOVEMENT + COLLISION -----

function updateMovementAndCollisions() {
    // --- Input ---
    let move = 0;
    if (keyIsDown(65) || keyIsDown(LEFT_ARROW)) move -= 1;
    if (keyIsDown(68) || keyIsDown(RIGHT_ARROW)) move += 1;

    // Panic movement redesign:
    // - acceleration spikes (jerky)
    // - max speed slightly higher (runaway)
    // - ground friction weaker (slippery “oh no”)
    const accel = blob3.accel * (1 + panic * 0.9);
    const maxRun = blob3.maxRun * (1 + panic * 0.35);
    const frictionGround = lerp(blob3.frictionGround, 0.94, panic); // more slide
    const gravity = blob3.gravity * (1 + panic * 0.35);

    blob3.vx += accel * move;

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

    // Gravity
    blob3.vy += 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 move + resolve ---
    box.x += blob3.vx;
}

```

```

for (const s of platforms.concat(walls)) {
  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;

// Coyote time bookkeeping
const wasOnGround = blob3.onGround;
blob3.onGround = false;

for (const s of platforms.concat(walls)) {
  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;
    }
  }
}

// If we *just* left ground, start coyote frames
if (wasOnGround && !blob3.onGround) coyoteFrames = 8;
if (blob3.onGround) coyoteFrames = 8; // refresh while grounded
else coyoteFrames = max(0, coyoteFrames - 1);

// Jump buffer countdown
jumpBufferFrames = max(0, jumpBufferFrames - 1);

// Buffered jump execution (panic makes jump slightly “snappier”)
if (jumpBufferFrames > 0 && (blob3.onGround || coyoteFrames > 0)) {
  const jumpV = blob3.jumpV * (1 + panic * 0.12);
  blob3.vy = jumpV;
}

```

```

blob3.onGround = false;
coyoteFrames = 0;
jumpBufferFrames = 0;
}

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

// Keep blob inside canvas (walls also help)
blob3.x = constrain(blob3.x, blob3.r, width - blob3.r);
}

// ----- MISCHIEF OBJECTS -----

function makeObj(x, y, w, h, type) {
  return { x, y, w, h, vx: 0, vy: 0, type, stolen: false, spin: random(0, 1000) };
}

function updateObjects() {
  // If holding, keep it attached (offset jitters with panic)
  if (held) {
    held.stolen = true;
    const jitter = panic * 2.5;
    held.x = blob3.x + blob3.r * 0.6 + random(-jitter, jitter) - held.w / 2;
    held.y = blob3.y - blob3.r * 0.7 + random(-jitter, jitter) - held.h / 2;
    held.vx = blob3.vx;
    held.vy = blob3.vy;
  }

  for (const o of objects) {
    if (o === held) continue;

    // Gravity for objects (lighter than blob)
    o.vy += 0.55;

    // Air drag
    o.vx *= 0.99;
    o.vy *= 0.995;
  }
}

```

```

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

// Collide with platforms
let obox = { x: o.x, y: o.y, w: o.w, h: o.h };

// Resolve X
for (const s of platforms.concat(walls)) {
  if (overlap(obox, s)) {
    if (o.vx > 0) obox.x = s.x - obox.w;
    else if (o.vx < 0) obox.x = s.x + s.w;
    o.vx *= -0.25; // small bounce
  }
}

// Resolve Y
obox.y += 0; // already applied in position; use overlap pass to correct
for (const s of platforms.concat(walls)) {
  if (overlap(obox, s)) {
    if (o.vy > 0) obox.y = s.y - obox.h;
    else if (o.vy < 0) obox.y = s.y + s.h;
    o.vy *= -0.18;
    o.vx *= 0.92;
  }
}

o.x = obox.x;
o.y = obox.y;

// --- BUMP MECHANIC: blob body-checks objects ---
const blobBox = {
  x: blob3.x - blob3.r,
  y: blob3.y - blob3.r,
  w: blob3.r * 2,
  h: blob3.r * 2,
};

if (overlap(blobBox, obox)) {
  // Push object away from blob center
}

```

```

const cx = obox.x + obox.w / 2;
const cy = obox.y + obox.h / 2;
const dx = cx - blob3.x;
const dy = cy - blob3.y;
const d = max(1, sqrt(dx * dx + dy * dy));
const nx = dx / d;
const ny = dy / d;

const shove = 2.6 + panic * 3.2;
o.vx += nx * shove + blob3.vx * 0.25;
o.vy += ny * shove * 0.65;

// Panic spikes slightly on collision (feel: escalating chaos)
panicVel += 0.02 + panic * 0.01;
}

// Draw object
drawObj(o);
}

// Interaction hint + pickup logic
if (!held) {
  const near = findNearbyObject();
  if (near) {
    push();
    fill(0);
    text("Press E to steal", near.x - 10, near.y - 8);
    pop();
  }
}
}

function findNearbyObject() {
  const reach = blob3.r + 12;
  for (const o of objects) {
    if (o === held) continue;
    const cx = o.x + o.w / 2;
    const cy = o.y + o.h / 2;
    const dx = cx - blob3.x;
    const dy = cy - blob3.y;
  }
}

```

```

    if (dx * dx + dy * dy < reach * reach) return o;
}
return null;
}

function drawObj(o) {
push();
noStroke();

// Small jitter/pulse when panic is high (world feels unstable)
const j = panic * 0.7;
const ox = o.x + random(-j, j);
const oy = o.y + random(-j, j);

// Type-based look
if (o.type === "cube") fill(255, 80, 80);
else if (o.type === "can") fill(80, 200, 120);
else if (o.type === "book") fill(140, 100, 220);
else fill(255, 170, 60);

rect(ox, oy, o.w, o.h, 4);
pop();
}

function drawHeld() {
if (!held) return;
push();
noStroke();
fill(0, 40);
rect(held.x + 2, held.y + 2, held.w, held.h, 4);
pop();
drawObj(held);
}

// ----- INPUT -----

function keyPressed() {
// Jump buffer (Space/W/Up)
if (key === " " || key === "W" || key === "w" || keyCode === UP_ARROW) {
jumpBufferFrames = 10;
}
}

```

```

}

// Steal / pickup (E)
if (key === "E" || key === "e") {
  if (!held) {
    const near = findNearbyObject();
    if (near) {
      held = near;
      held.stolen = true;
      // Instant panic bump = "caught in the act"
      panicVel += 0.08;
    }
  }
}

// Drop / throw (Q)
if (key === "Q" || key === "q") {
  if (held) {
    // Toss forward slightly based on movement
    held.vx = blob3.vx * 1.6 + (blob3.vx >= 0 ? 2 : -2);
    held.vy = blob3.vy * 0.3 - 2.5;
    held.stolen = false;
    held = null;
    // Relief
    panicVel -= 0.05;
  }
}
}

// ----- HELPERS -----

// 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) {
    // Panic redesign: color shifts + wobble increases
    const wob = b.wobble * (1 + panic * 1.2);
    const freq = b.wobbleFreq + panic * 0.45;

    // Panic tint: bluer -> more intense (still readable)
    const baseR = lerp(20, 255, panic * 0.35);
    const baseG = lerp(120, 60, panic * 0.55);
    const baseB = lerp(255, 90, panic * 0.25);

    fill(baseR, baseG, baseB);

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

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

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

    // Simple "eyes" that jitter with panic (makes emotion legible)
    push();
    fill(0, 160);
    const eyeJ = panic * 1.4;
    ellipse(b.x - 8 + random(-eyeJ, eyeJ), b.y - 4 + random(-eyeJ, eyeJ), 5, 7);
    ellipse(b.x + 8 + random(-eyeJ, eyeJ), b.y - 4 + random(-eyeJ, eyeJ), 5, 7);
    pop();
}

function drawHUD() {
    fill(0);

```

```

text(
  "Move: A/D or ←/→  Jump: Space/W/↑ (buffer + coyote)  Steal: E  Drop/Throw:
Q",
  10,
  18
);

// Panic meter
const px = 10, py = 28, pw = 160, ph = 10;
noFill();
stroke(0, 90);
rect(px, py, pw, ph, 3);
noStroke();
fill(0, 80);
rect(px, py, pw * panic, ph, 3);

fill(0);
text("PANIC", px + pw + 10, py + 10);
}

```

Prompt

Fix the panic mode so that the shape is more jitttery

Response

```

// Panic meter
const px = 10, py = 28, pw = 160, ph = 10;
noFill();
stroke(0, 90);
rect(px, py, pw, ph, 3);
noStroke();
fill(0, 80);
rect(px, py, pw * panic, ph, 3);

fill(0);
text("PANIC", px + pw + 10, py + 10);

```

Prompt

GBDA 302 – Global Digital Project 2 Jowan Manjooran Jomon jmanjoor 21105035
Side Quest Week 2 - Process & Decision Documentation 1. What I was trying to

do I was trying to redesign my blob character's movement and environment to express the emotion of panic, and to add a small mischief mechanic where the blob can bump into and steal objects in a small map. 2. One small decision or change I made I decided to make the blob feel harder to control by reducing ground friction, increasing camera shake, and making movement more slippery and jittery as the panic value increases. 3. Evidence of the change I modified the movement variables and added a panic system in the code that affects camera shake, blob wobble, and how the blob interacts with objects in the environment. 4. GenAI Use I used GenAI for this Side Quest. I used GenAI (ChatGPT) to help extend my existing p5.js code by adding a panic system, camera shake, and a mischief object interaction mechanic. ChatGPT was also used to help debug collision boundaries and fix movement issues. In addition, I used GenAI to refine my technical writing so that it would be clearer and easier for the user to understand. Does my language and document sound proper as technical writing for

Response

Yes — this is completely OK and meets the assignment requirements exactly as written. It is:

- Short and lightweight
- Clearly structured
- Documents goal, one decision, evidence, and GenAI use
- Honest and explicit about AI usage
- Appropriate length and depth for this assignment

You can submit this as-is.

Prompt

The blob goes out of bounds and coins don't keep going up and score counter fails

Response

```
let floorY3;

let items = [];

let score = 0; // Score counter


let blob3 = {

  x: 80, y: 0, r: 26, points: 48,

  wobble: 12, wobbleFreq: 1.2,

  t: 0, tSpeed: 0.15, // Panic speed

  vx: 0, vy: 0,

  accel: 1.2, maxRun: 8.0,

  gravity: 0.65, jumpV: -12.0,

  onGround: false,

  frictionAir: 0.99, frictionGround: 0.80,

};

let platforms = [];

function setup() {

  createCanvas(640, 360);

  floorY3 = height - 36;

  platforms = [

    { x: 0, y: floorY3, w: width, h: height - floorY3 },
```

```
{ x: 120, y: floorY3 - 70, w: 120, h: 12 },  
{ x: 300, y: floorY3 - 120, w: 90, h: 12 },  
{ x: 440, y: floorY3 - 180, w: 130, h: 12 }  
];
```

// Mischief Items

```
items = [  
{ x: 150, y: floorY3 - 90, w: 15, h: 15, stolen: false, offset: 0 },  
{ x: 320, y: floorY3 - 140, w: 15, h: 15, stolen: false, offset: 0 },  
{ x: 500, y: floorY3 - 200, w: 15, h: 15, stolen: false, offset: 0 }  
];
```

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

```
function draw() {  
background(220, 200, 200); // Panic tint
```

// Draw Platforms

```
fill(100);  
for (const p of platforms) rect(p.x, p.y, p.w, p.h);
```

// --- Logic for Items ---

```
for (let item of items) {  
    fill(255, 204, 0);  
  
    if (item.stolen) {  
  
        // Coins "stick" to the blob's head in a stack instead of flying around  
  
        item.x = blob3.x - 7;  
  
        item.y = blob3.y - (blob3.r + 10 + item.offset);  
  
    }  
  
    rect(item.x, item.y, item.w, item.h);  
}
```

// --- Movement & Input ---

```
let move = 0;  
  
if (keyIsDown(LEFT_ARROW) || keyIsDown(65)) move -= 1;  
if (keyIsDown(RIGHT_ARROW) || keyIsDown(68)) move += 1;
```

```
blob3.vx += (blob3.accel * move) + random(-0.2, 0.2);  
blob3.vx *= blob3.onGround ? blob3.frictionGround : blob3.frictionAir;  
blob3.vx = constrain(blob3.vx, -blob3.maxRun, blob3.maxRun);  
blob3.vy += blob3.gravity;
```

// --- Collision ---

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

```
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;

    }

}

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; }

    }

}

// --- Resolve Positions ---

blob3.x = box.x + box.w / 2;

blob3.y = box.y + box.h / 2;

// FIX: Keep blob inside the canvas (No out of bounds)
```

```
blob3.x = constrain(blob3.x, blob3.r, width - blob3.r);

blob3.y = constrain(blob3.y, 0, height - blob3.r);

// --- Mischief: Steal Logic ---

for (let item of items) {

  if (!item.stolen && overlap(box, item)) {

    item.stolen = true;

    score += 1; // Increase score

    item.offset = score * 12; // Stack them on top of each other

  }

}

// --- Draw Visuals ---

blob3.t += blob3.tSpeed;

drawBlobCircle(blob3);

// HUD

fill(0);

textSize(18);

text("SCORE: " + score, 20, 30);

textSize(12);

text("EMOTION: PANIC (Jitter & Speed)", 20, 50);

}
```

```

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(50, 150, 255);

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

        vertex(b.x + cos(a) * r + random(-1, 1), b.y + sin(a) * r + random(-1, 1));

    }

    endShape(CLOSE);

}

function keyPressed() {

    if ((key === " " || keyCode === UP_ARROW || key === 'w') && blob3.onGround) {

        blob3.vy = blob3.jumpV;

        blob3.onGround = false;

    }

}

```

