# Process & Decision Documentation

## Entry Header

Name: Jolin Li

Role(s): Side Quest 2

Primary responsibility for this work: Redesigning the code and incorporating a mood

Goal of Work Session: Have the blob demonstrate panic

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

assignment.

I was trying to get the screen to flash red when the blob moves and add a platform design.

GenAI Documentation

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

Date Used
January 22, 2026

Tool Disclosure
ChatGPT 5.2

Purpose of Use
Generate ideas

Summary of Interaction

It contributed to the platform design, camera shake, and blob deformation.

Human Decision Point(s)

I added the flashing red screen and panic level meter.

Integrity & Verification Note
I checked with assignment criteria and course concepts.

Scope of GenAI Use
GenAI did not contribute to the flashing red screen effect when moving the blob around.

Summary of Process (Human + Tool)

Testing different codes

Troubleshooting

Describe one or two key decisions you made:

I decided to express the emotion “panic”. Doing so, I incorporated a flashing red screen that demonstrates emergency and a sense of fear. I also decided to include the platform design for a visual effect that further contributed to the emotion.

Verification & Judgement

Playtesting
Comparison with course concepts
Re-reading assignment criteria

## Appendix

Using example 3 as a starting code add a platform design of yellow warning stripes and add a panic level when blob moves.

```
// PANIC BLOB
// Emotion expressed via frantic motion, camera shake, blob color shift, and
environment
```

```
let floorY3;
let platforms = [];

// Panic FX (internal only)
let panic = 0;
let shakeAmt = 0;

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

  r: 26,
  points: 52,
  wobble: 7,
  wobbleFreq: 1.05,

  t: 0,
  tSpeed: 0.012,

  vx: 0,
  vy: 0,

  // PANIC tuning
  accel: 0.75,
  maxRun: 4.8,
  gravity: 0.7,
  jumpV: -11.5,

  onGround: false,

  frictionAir: 0.992,
  frictionGround: 0.84,
};

function setup() {
  createCanvas(640, 360);
  floorY3 = height - 36;
  noStroke();
  textFont("sans-serif");
  textSize(14);

  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 },
    { x: 520, y: floorY3 - 70, w: 90, h: 12 },
  ];

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

function draw() {

  const speed = abs(blob3.vx) + abs(blob3.vy) * 0.15;
  panic = lerp(panic, constrain(speed / 6.0, 0, 1), 0.08);

  background(240);

  // Camera shake
  shakeAmt = lerp(shakeAmt, panic * 6, 0.1);
  const sx = random(-shakeAmt, shakeAmt);
  const sy = random(-shakeAmt, shakeAmt);

  push();
  translate(sx, sy);

  drawPlatforms(panic);

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

  // Panic twitch
  const twitch = (noise(frameCount * 0.05) - 0.5) * panic * 0.6;
  blob3.vx += blob3.accel * (move + twitch);
```

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

blob3.vy += blob3.gravity;

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

// 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;
    shakeAmt += 1.5 * panic;
  }
}

// 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;
      if (blob3.vy > 5) shakeAmt += 3;
      blob3.vy = 0;
      blob3.onGround = true;
    } else if (blob3.vy < 0) {
      box.y = s.y + s.h;
      blob3.vy = 0;
      shakeAmt += 1.2 * panic;
      }
    }
  }

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

  // Visual response to panic
  blob3.tSpeed = 0.012 + panic * 0.028;
  blob3.wobble = 7 + panic * 9;
  blob3.wobbleFreq = 1.05 + panic * 0.9;

  blob3.t += blob3.tSpeed;
  drawBlobCircle(blob3);

  pop();

  // HUD
  fill(0);
  text("Move: A/D or ←/→   Jump: Space/W/↑", 10, 18);
}

function keyPressed() {
  if (
    (key === " " || key === "W" || key === "w" || keyCode === UP_ARROW) &&
    blob3.onGround
  ) {
    blob3.vy = blob3.jumpV;
    blob3.onGround = false;
    shakeAmt += 2;
  }
}

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 drawPlatforms(p) {
  fill(190);
  for (const plat of platforms) rect(plat.x, plat.y, plat.w, plat.h);

  for (const plat of platforms) {
    if (plat.h > 20) continue;
    for (let x = plat.x; x < plat.x + plat.w; x += 14) {
      fill(30, 30, 30, 140);
      rect(x, plat.y, 7, 6);

      fill(250, 210 - p * 70, 40 + p * 30, 170);
      rect(x + 7, plat.y, 7, 6);
    }
  }

  fill(255, 80 + p * 80, 80 + p * 80, 110);
  rect(0, floorY3 - 2, width, 2);
}

function drawBlobCircle(b) {
  const rr = lerp(20, 240, panic);
  const gg = lerp(120, 60, panic);
  const bb = lerp(255, 80, panic);
  fill(rr, gg, bb);

  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 jitter =
      (noise(i * 0.2, frameCount * 0.06) - 0.5) * panic * 2.2;
    const r = b.r + map(n, 0, 1, -b.wobble, b.wobble) + jitter;
    vertex(b.x + cos(a) * r, b.y + sin(a) * r);
  }
  endShape(CLOSE);

  fill(255, 255, 255, 120);
  ellipse(b.x - b.r * 0.25, b.y - b.r * 0.25, b.r * 0.5);
}
```