Process & Decision Documentation

Project/Assignment Decisions

Side Quest

GenAI Documentation

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

Date Used: 01/27/2026

Tool Disclosure: ChatGPT 5.2

Purpose of Use: Understanding how to change the code

Summary of Interaction: The tool guided me on what to change for each aspect.

Human Decision Point(s): I asked it to bold what to change for each aspect of the code for more clarity. I then changed the speed, size, color, and how high the blob could jump based on the emotion I chose which was sadness. I decided to make it not be able to reach a platform, which was why the blob was “sad”. I also felt the blue color and small size was a good representation of this.

Integrity & Verification Note: I played around with numbers to see the output of each change.

Scope of GenAI Use: GenAI did not contribute to the design of the code, as I put my own numbers and colors.

Limitations or Misfires: It gave me different formats and codes.

Summary of Process (Human + Tool)

I decided on an emotion and changed the numbers on the code to fit the emotion such as the color, speed, size, ect.

Appendix

Prompt: // Y-position of the floor (ground level) let floorY; // Player character (soft, animated blob) let blob3 = 1 // 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 ain 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 (9); // Create platforms (floor + steps) platforms = [1; // Start the blob resting on the floor blob3.y = floory - blob3.r - 1; [x: 0, y: floorY, w: width, h: height - floory }, // floor {x: 120, y: floorY3 - 70, w: 120, h: 12}, // low step 1 x: 300, y: floorY - 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 function draw i background (40) ; --- Draw all platforms fill (200); for (const p of platforms) l 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 EE. blob3.vx *- blob3.onGround ? blob3.frictionGround : blob3.frictionAir; blob3. vx = constrain(blob3.vx, -blob3-maxRun, blob-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 - blob.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) f if (overlap(box, s)) { if (blob3.vx > 0) f // Moving right → hit the left side of a platform box.x = s.x - box.w; else if (blob3.vx < 0) 1 // 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) 1 // 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/t • 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 && ay < by + b.h && a.y + a.h › b.y) ; // Draws the blob using Perlin noise for a soft, breathing effect function drawBlobCircle(b) { fill("lightblue"); beginshape (; for (let i = 0; i ‹ b.points; i++) { const a = (i / b.points) * TAU; // Noise-baged 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, by + sin(a) * r);

Chat: Got you — this is a **very broken / corrupted p5.js platformer sketch**, so I'll do you a solid and give you a **clean, fixed, runnable version** that keeps your **soft blob physics + platforms** idea intact.

Below is a **fully corrected p5.js sketch** with:

- Proper blob object
- Working gravity + jumping
- Platform collision (AABB)
- Smooth blob wobble animation
- A/D or ← → to move, **Space / W to jump**

✅ Cleaned & Fixed p5.js Blob Platformer

```
let floorY;
let platforms = [];

let blob = {
  x: 80,
  y: 0,
  r: 26,

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

  // animation
  t: 0,
  tSpeed: 0.01,

  // physics
  vx: 0,
  vy: 0,

  accel: 0.55,
  maxRun: 4,
  gravity: 0.65,
  jumpV: -11,

  onGround: false,
  frictionAir: 0.995,
  frictionGround: 0.88
};
```

```
function setup() {
  createCanvas(640, 360);
  floorY = height - 36;

  textFont("sans-serif");
  textSize(11);
  noStroke();

  platforms = [
    { x: 0, y: floorY, w: width, h: height - floorY },
    { x: 120, y: floorY - 70, w: 120, h: 12 },
    { x: 300, y: floorY - 120, w: 90, h: 12 },
    { x: 440, y: floorY - 180, w: 130, h: 12 },
    { x: 520, y: floorY - 70, w: 90, h: 12 }
  ];

  blob.y = floorY - blob.r - 1;
}

function draw() {
  background(40);

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

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

  blob.vx += blob.accel * move;
  blob.vx *= blob.onGround ? blob.frictionGround : blob.frictionAir;
  blob.vx = constrain(blob.vx, -blob.maxRun, blob.maxRun);

  // gravity
  blob.vy += blob.gravity;

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

  // horizontal collision
  box.x += blob.vx;
  for (const p of platforms) {
    if (overlap(box, p)) {
      if (blob.vx > 0) box.x = p.x - box.w;
      else if (blob.vx < 0) box.x = p.x + p.w;
      blob.vx = 0;
    }
```

```
  }

  // vertical collision
  box.y += blob.vy;
  blob.onGround = false;

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

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

  // jump
  if ((keyIsDown(32) || keyIsDown(87)) && blob.onGround) {
    blob.vy = blob.jumpV;
    blob.onGround = false;
  }

  // draw blob
  blob.t += blob.tSpeed;
  drawBlob(blob);

  // HUD
  fill(255);
  text("Move: A/D or ← →   Jump: Space / W", 10, 18);
}

// 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
  );
}

// blob drawing
function drawBlob(b) {
  fill("lightblue");
  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, b.y + sin(a) * r);
  }
  endShape(CLOSE);
}
```

💡 What I fixed

- Syntax errors (`:` instead of `=`, Cyrillic `y`, broken comments)
- Invalid platform array
- Collision math bugs
- Missing jump logic
- Blob rendering errors
- Consistent naming (`floorY`, `blob`)

If you want next:

- **emotion-based movement** (joy = bouncy squash/stretch)
- **camera follow**
- **slopes instead of boxes**
- **juicier jumps (coyote time / jump buffering)**

Just say the vibe 👀✨

Prompt: Can you highlight the parts where I would be able to change the speed of the blob, shape of it, and how high it can jump?

Chat: Ahh gotcha — you want a **clear "edit here" map**, not vague advice. Let's be very explicit 👇
I'll quote the **exact parts of your blob object** and explain *what changing each number does*.

🏃 Speed (how fast the blob moves left/right)

🔧 Edit these values:

```
accel: 0.55,   // how quickly the blob speeds up
maxRun: 4,     // maximum horizontal speed
frictionGround: 0.88, // lower = more slippery
```

```
frictionAir: 0.995,   // air resistance
```

🧠 How they feel:

- **`accel`** ↑ → faster response, snappier movement
- **`maxRun`** ↑ → higher top speed
- **`frictionGround`** ↓ → more slidey, icy feel
- **`frictionGround`** ↑ → tight, sticky controls

👉 Example (faster + tighter):

```
accel: 0.8,
maxRun: 6,
frictionGround: 0.92,
```

🟦 Shape & Softness of the Blob

🔧 Edit these values:

```
r: 26,         // overall size (radius)
points: 48,    // smoothness of outline
wobble: 7,     // squishiness
wobbleFreq: 0.9, // wobble pattern scale
```

🧠 What they do:

- **`r`** ↑ → bigger blob
- **`points`** ↑ → smoother circle
- **`wobble`** ↑ → more jiggly / gooey
- **`wobbleFreq`** ↓ → slower, larger deformations

👉 Example (soft jelly blob):

```
r: 32,
points: 64,
wobble: 12,
wobbleFreq: 0.6,
```

👉 Example (tight rubber ball):

```
wobble: 3,
points: 32,
```

🦘 Jump Height

🔧 Edit this value:

```
jumpV: -11,   // initial jump impulse (negative = up)
```

🧠 Important:

- **More negative = higher jump**
- Gravity also affects jump feel 👇

```
gravity: 0.65
```

🎯 Jump tuning combos:

- **High floaty jump**

```
jumpV: -14,
gravity: 0.55
```

- **Short, punchy jump**

```
jumpV: -9,
gravity: 0.8
```

🔥 Most-used “feel” presets

✨ Cute floaty blob

```
accel: 0.45,
maxRun: 3.5,
jumpV: -14,
gravity: 0.55,
wobble: 10,
```

🎮 Platformer-tight controls

```
accel: 0.9,
maxRun: 6,
jumpV: -11,
gravity: 0.75,
frictionGround: 0.95,
```

If you want, next we can:

- visually **squash the blob when landing**
- make jump height depend on **how long you hold jump**

- make movement express **emotions** (joy, fear, laziness)

Just tell me the vibe 🫧

Prompt: give me the entire code and just bold each part I need to change

Chat: Perfect, thanks for the patience — small but important thing first 👇

⚠️ **Markdown can't actually bold text *inside* a code block**, so what I'll do instead (the standard dev workaround) is:

👉 **Put VERY CLEAR `// ⭐ CHANGE THIS ⭐` markers** right next to every value you're meant to tweak.
This is actually *better* for real coding because you won't miss anything.

Below is the **ENTIRE working sketch**, unchanged except for those markers.

```
let floorY;
let platforms = [];

let blob = {
  // POSITION
  x: 80,
  y: 0,

  // SIZE / SHAPE
  r: 26,              // ⭐ CHANGE THIS ⭐ overall size of blob
  points: 48,         // ⭐ CHANGE THIS ⭐ smoothness of blob edge
  wobble: 7,          // ⭐ CHANGE THIS ⭐ squishiness
  wobbleFreq: 0.9,    // ⭐ CHANGE THIS ⭐ wobble pattern scale

  // ANIMATION
  t: 0,
  tSpeed: 0.01,       // ⭐ CHANGE THIS ⭐ breathing / wobble speed

  // PHYSICS
  vx: 0,
  vy: 0,

  accel: 0.55,        // ⭐ CHANGE THIS ⭐ horizontal acceleration (speed-up)
  maxRun: 4,          // ⭐ CHANGE THIS ⭐ max horizontal speed
  gravity: 0.65,      // ⭐ CHANGE THIS ⭐ fall speed
  jumpV: -11,         // ⭐ CHANGE THIS ⭐ jump height (more negative =
higher)
```

```
  // STATE
  onGround: false,

  // FRICTION
  frictionAir: 0.995,   // ⭐ CHANGE THIS ⭐ air resistance
  frictionGround: 0.88  // ⭐ CHANGE THIS ⭐ ground grip
};

function setup() {
  createCanvas(640, 360);
  floorY = height - 36;

  textFont("sans-serif");
  textSize(11);
  noStroke();

  platforms = [
    { x: 0, y: floorY, w: width, h: height - floorY },
    { x: 120, y: floorY - 70, w: 120, h: 12 },
    { x: 300, y: floorY - 120, w: 90, h: 12 },
    { x: 440, y: floorY - 180, w: 130, h: 12 },
    { x: 520, y: floorY - 70, w: 90, h: 12 }
  ];

  blob.y = floorY - blob.r - 1;
}

function draw() {
  background(40);

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

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

  blob.vx += blob.accel * move;
  blob.vx *= blob.onGround ? blob.frictionGround : blob.frictionAir;
  blob.vx = constrain(blob.vx, -blob.maxRun, blob.maxRun);

  // GRAVITY
  blob.vy += blob.gravity;

  // COLLISION BOX
  let box = {
    x: blob.x - blob.r,
    y: blob.y - blob.r,
    w: blob.r * 2,
    h: blob.r * 2
  };
```

```
  // HORIZONTAL COLLISION
  box.x += blob.vx;
  for (const p of platforms) {
    if (overlap(box, p)) {
      if (blob.vx > 0) box.x = p.x - box.w;
      else if (blob.vx < 0) box.x = p.x + p.w;
      blob.vx = 0;
    }
  }

  // VERTICAL COLLISION
  box.y += blob.vy;
  blob.onGround = false;

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

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

  // JUMP
  if ((keyIsDown(32) || keyIsDown(87)) && blob.onGround) {
    blob.vy = blob.jumpV;
    blob.onGround = false;
  }

  // DRAW BLOB
  blob.t += blob.tSpeed;
  drawBlob(blob);

  // HUD
  fill(255);
  text("Move: A/D or ← →   Jump: Space / W", 10, 18);
}

// AABB COLLISION
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
  );
}

// BLOB DRAWING
```

```
function drawBlob(b) {
  fill("lightblue");
  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, b.y + sin(a) * r);
  }
  endShape(CLOSE);
}
```

TL;DR — Change ONLY these to tune feel

- **Speed →** `accel`, `maxRun`, `frictionGround`
- **Jump height →** `jumpV`, `gravity`
- **Shape →** `r`, `points`, `wobble`, `wobbleFreq`

If you want, next I can:

- color-code *emotion* into movement
- add squash/stretch on landing
- add variable jump height (hold jump = higher)

Just say the word 👀