

Process & Decision Documentation

Project/Assignment Decisions

I chose the emotion of fear. To represent this I decided to add a ghost into the blob's world, when interacting with the ghost, the blob's noise increases so the blob appears startled. During this decision I wanted to also add a black layer on top of everything with only the portion around the blob to be visible to add to the fear. However, I ended up removing it as the erase function was not working well as a moving object. Another decision I made was to adjust one of the platforms. I found the gap between two of the highest platforms was too small for the blob and it would get stuck. I also added another platform for more space coverage.

GenAI Documentation

Date Used: Saturday January 24th

Tool Disclosure: ChatGPT

Purpose of Use: I used Chatgpt to assist with creating and troubleshooting a feature for the side quest

Summary of Interaction: I asked Chatgpt how I would go about creating a code so when the blob interacts with the ghost the ghost spawns to another place. When it didn't work I asked for clarification on if the implementation is correct. After doing some edits of my own I came to a road block and asked for more assistance.

Human Decision Point(s): I choose what part of the code is beneficial to my desired result, as well as edit them to fit my vision. Examples include removing the "choose to follow ghost" cutscene, implementing randomization code, adjusting the startled blob frequency to look more startled, changing background and text colour for mood, and adjusting the steps.

Integrity & Verification Note: Through implementation and testing I would find areas of error or misinterpretation from the AI. An example being the ghost not appearing in random areas and always generating in the same spot.

Scope of GenAI Use: Gen AI was used to create the base for the ghost teleporting, and startled blob effect. It also assisted with troubleshooting twice further down the line.

Limitations or Misfires: It didn't seem to understand what I was aiming for and added in extra unnecessary features. Additionally, it didn't explain how to properly implement the new code which left me with some confusion.

Summary of Process (Human + Tool) The process involved getting a basic concept of the ghost code from ChatGPT. From there I would edit and add to the code to fit my desired results. When I came across a problem I couldn't resolve, I would ask chatGPT for assistance. I would make sure the advice worked with my code, making adjustments and edits myself. Once the error was passed I continued coding myself, only going to chatGPT if I was stuck or unsure.

Verification & Judgement When I first generated the response instead of making the ghost generate randomly when the blob hits it, it would cause a blank screen to appear saying "you followed the ghost". This was something the AI added on its own which I removed myself. After this first generation I realized it didn't code the ghost randomizing in different locations, just the blob interacting. I added in the randomizing code however I came across multiple errors or unwanted results. After rounds of failed troubleshooting I used chatgpt and learned where I was causing problems. I found through this process that gen AI is more effective at pointing out logic errors.

Appendix

ME: I have this code. I wish for a ghost to spawn on the screen, and when the blob interacts with the ghost it appears in a new section. How would I implement it (please add comments to your iterations)

```
// 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
};
```

```

let ghost = {
  //body
  rectW: 50,
  rectH: 30,
  x: random(width - rectW),
  y: random(height - rectH),

  rect: (x, y, rectW, rectH),

  //face
  circle: (x + 10, y - 20, 10),
  circle: (x + 30, y - 20, 10),
};

// 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);
}

//draw ghost
ghost;

// --- 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("white");
```

```
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
*/

```

Chatgpt: You're actually very close already — the main things missing are:

1. **A proper ghost object structure**
2. **A way to draw the ghost**
3. **Collision detection between blob and ghost**
4. **Some notion of “sections” (state change when touched)**

Below I'll show a **clean, minimal way** to layer this on top of what you already have, with comments explaining *why* each step exists.

1 Define a ghost properly (spawn + state)

Right now `ghost` isn't doing anything because `rect:` and `circle:` don't draw by themselves.

Let's define the ghost as a simple rectangle hitbox + a flag to track if it's been collected.

```
let ghost = {  
  // Position + size (AABB like platforms)  
  x: 420,  
  y: floorY3 - 60,  
  w: 40,  
  h: 50,  
  
  active: true, // Is the ghost still in the world?  
};
```



Why this matters:

Using the same rectangle logic as platforms makes collision detection easy and consistent.

2 Draw the ghost

Add this **inside** `draw()`, after platforms are drawn:

```
// --- Draw ghost ---  
if (ghost.active) {  
  drawGhost(ghost);  
}
```

Now add a helper function:

```
function drawGhost(g) {  
  fill(255, 255, 255, 200);  
  
  // Body  
  rect(g.x, g.y, g.w, g.h, 12);  
  
  // Eyes  
  fill(0);  
  ellipse(g.x + 12, g.y + 18, 6);  
  ellipse(g.x + 28, g.y + 18, 6);  
}
```

🤖 This keeps visuals separate from logic (good practice).

3 Detect blob–ghost interaction

Right after you calculate the blob’s collision box:

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

Add:

```
// --- Blob ↔ Ghost interaction ---  
if (ghost.active && overlap(box, ghost)) {  
  ghost.active = false; // Remove ghost  
  currentSection = 1; // Move to a new section  
}
```

4 Add “sections” (simple game state)

At the top of your file:

```
let currentSection = 0; // 0 = main area, 1 = new area
```

Now update `draw()` so the world changes depending on section:

```
function draw() {  
  background(240);  
  
  if (currentSection === 0) {  
    drawMainSection();  
  } else if (currentSection === 1) {  
    drawGhostSection();  
  }  
}
```

5 Split your existing logic into sections

Main section (your current gameplay)

Move **most of your existing draw logic** into:

```
function drawMainSection() {  
  // platforms  
  fill(200);  
  for (const p of platforms) rect(p.x, p.y, p.w, p.h);  
  
  if (ghost.active) drawGhost(ghost);  
  
  // ALL blob movement + physics code stays here  
  // (literally paste it from draw())  
}
```

New section (after ghost interaction)

```
function drawGhostSection() {  
  background(30, 20, 50);  
  
  fill(255);  
  textAlign(CENTER);  
  textSize(24);
```

```
text("You followed the ghost...", width / 2, height / 2);

// Optional: draw blob here too
drawBlobCircle(blob3);
}
```



Conceptually:

This is a **state machine** — touching the ghost flips the game into a new “mode.”

6 Why this approach works well (and scales)

- ✓ Uses the **same collision system** you already wrote
- ✓ Ghosts can be reused (just push into an array later)
- ✓ Sections can become:
 - levels
 - cutscenes
 - puzzles
 - story moments

If you want next steps, I can help you:

- Turn ghosts into **multiple collectibles**
- Add a **fade / transition**
- Teleport the blob instead of switching scenes
- Make the ghost **move or flee**

Just tell me the vibe 🧐✨

Me: did I implement the code properly? The sketch is completely blank

```
// 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
```

```
};
```

```
let ghost = {
```

```
  // Position + size (AABB like platforms)
```

```
  x: 420,
```

```
  y: floorY3 - 60,
```

```
  w: 40,
```

```
  h: 50,
```

```
  active: true, // Is the ghost still in the world?
```

```
};
```

```
let currentSection = 0; // 0 = main area, 1 = new area
```

```
// 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);
```

```

if (currentSection === 0) {
    drawMainSection();
} else if (currentSection === 1) {
    drawGhostSection();
}

// Draw ghost
if (ghost.active) {
    drawGhost(ghost);
}

// --- HUD ---
fill("white");
text("Move: A/D or ←/→ • Jump: Space/W/↑ • Land on platforms", 10, 18);
}

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

```

```
if (ghost.active) drawGhost(ghost);

// ALL blob movement + physics code stays here

// --- 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;

}

}

}

```

```

//ghost collison

```

```

let box2 = {

    x: blob3.x - blob3.r,

    y: blob3.y - blob3.r,

    w: blob3.r * 2,

    h: blob3.r * 2,

};

// --- Blob ↔ Ghost interaction ---

if (ghost.active && overlap(box2, ghost)) {

    ghost.active = false; // Remove ghost

    currentSection = 1; // Move to a new section

}

```

```
// --- 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);
}


function drawGhostSection() {

  background(30, 20, 50);


  fill(255);

  textAlign(CENTER);

  textSize(24);

  text("You followed the ghost...", width / 2, height / 2);


  // Optional: draw blob here too

  drawBlobCircle(blob3);
}
```

```
// 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;
```

```
  }
```

```
}
```

```
//drawing ghost function
```

```
function drawGhost(g) {
```

```
  fill(255, 255, 255, 200);
```

```
// Body
```

```
rect(g.x, g.y, g.w, g.h, 12);
```

```
// Eyes
```

```
fill(0);
```

```
ellipse(g.x + 12, g.y + 18, 6);
```

```
ellipse(g.x + 28, g.y + 18, 6);
```

```
}
```

```
/* 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

```
*/
```

Chatgpt: Short answer: **you were very close, but there *is* a real bug that explains the totally blank sketch** — and it’s a classic p5/JS gotcha, not a logic failure on your part.

Let’s walk through it calmly 🙌



The main reason your sketch is blank

✗ This line runs before `setup()`

```
let ghost = {
```

```
  x: 420,
```

```
  y: floorY3 - 60, // ← floorY3 is UNDEFINED here
```

```
w: 40,  
h: 50,  
active: true,  
};
```

At the time this code runs:

- `floorY3` has not been assigned yet
- So `floorY3` is `undefined`
- `undefined - 60` → `NaN`
- `rect(NaN, NaN, ...)` silently breaks rendering
- p5 doesn't crash loudly → **blank canvas**

This is *the* critical issue.

✅ **Fix #1 (best practice): create the ghost inside `setup()`**

Step 1: declare the variable only

```
let ghost;
```

Step 2: initialize it after `floorY3` exists

```
function setup() {  
  createCanvas(640, 360);  
  
  floorY3 = height - 36;  
  
  ghost = {  
    x: 420,
```

```
y: floorY3 - 60,  
w: 40,  
h: 50,  
active: true,  
};  
  
blob3.y = floorY3 - blob3.r - 1;  
}
```

💡 Rule of thumb:

Anything that depends on **width**, **height**, or **floorY3** must be created after **createCanvas()**

⚠️ Secondary issues (not fatal, but worth fixing)

1 You're drawing the ghost twice

You draw it in:

- **draw()**
- **drawMainSection()**

👉 Pick **one**. Best place is **drawMainSection()** only.

Remove this from **draw()**

```
if (ghost.active) {  
  drawGhost(ghost);  
}
```

2 HUD text is white on a very light background

This makes it *look* like nothing is there.

Change:

```
fill(0);
```

instead of:

```
fill("white");
```

3 Duplicate blob collision box (not wrong, just redundant)

You don't need `box2`. You already computed `box`.

Replace:

```
let box2 = { ... };
```

```
if (ghost.active && overlap(box2, ghost)) {
```

With:

```
if (ghost.active && overlap(box, ghost)) {
```

Cleaner + safer.

✓ Minimal corrected version (key fixes only)

```
let ghost; // declare only
```

```
function setup() {
```

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

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

```
  ghost = {
```

```
    x: 420,
```

```
    y: floorY3 - 60,
```

```
    w: 40,
```

```
    h: 50,
```

```
    active: true,
```

```
  };
```

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

```
}
```

Everything else you wrote is structurally **correct** 👍

Big picture reassurance

You:

- split logic into sections correctly ✓
- used state (`currentSection`) properly ✓
- reused AABB collision correctly ✓
- separated drawing from logic ✓

This wasn't a "you don't understand" bug — it was an **order-of-execution** bug. Everyone hits these in p5.

If you want next:

- smooth fade between sections
- ghost movement / floating animation
- multiple ghosts
- teleport instead of scene switch

I've got you 🤖✨

Me: For my code I have the ghost appearing in random positions. However, once the blob interacts with the ghost it does not change location and stays where it is. How may I fix it? Additionally how would I add a feature where when the blob interacts with the ghost the noise frequency gets more aggressive for just a moment?

```
// 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

};

let ghost;
```

```
// 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
  ];

  //ghost
```

```
ghost = {  
  x: random(width),  
  y: random(height),  
  w: 40,  
  h: 50,  
  active: true,  
  hit: false,  
};
```

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

```
function draw() {  
  background(0);  
  
  drawMainSection();  
  
  // --- HUD ---  
  fill("white");  
  text("Move: A/D or ←/→ • Jump: Space/W/↑ • Land on platforms", 10, 18);  
}
```

```
function drawMainSection() {
```

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

```
fill(200);
```

```
for (const p of platforms) {
```

```
  rect(p.x, p.y, p.w, p.h);
```

```
}
```

```
if (ghost.active) drawGhost(ghost);
```

```
if ((ghost.hit = true)) {
```

```
  drawGhost(ghost);
```

```
  ghost.hit = false;
```

```
}
```

```
// ALL blob movement + physics code stays here
```

```
// --- 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;
```

```
    }
```

```
  }
```

```
}
```

```
// --- Blob ↔ Ghost interaction ---
```

```
if (ghost.active && overlap(box, ghost)) {
```

```

    ghost.hit = true; // move ghost

    ghost.active = false; //remove first ghost
}

// --- 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);
}

// 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;  
  }  
}
```

//drawing ghost function

```
function drawGhost(g) {  
  fill(255, 255, 255, 200);  
  
  // Body  
  rect(g.x, g.y, g.w, g.h, 12);  
  
  // Eyes  
  fill(0);  
  ellipse(g.x + 12, g.y + 18, 6);  
  ellipse(g.x + 28, g.y + 18, 6);  
}
```

/* 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

```
*/
```

ChatGPT: Great questions — you’ve actually uncovered **two classic logic bugs** *and* you’re right at the point where adding juice (the wobble spike) makes a lot of sense. Let’s tackle this cleanly.

I’ll break it into:

1. **Why the ghost doesn’t move**
2. **The exact bug causing that**
3. **How to respawn the ghost properly**
4. **How to add a temporary “aggressive wobble” effect**

1 Why your ghost never changes position

Right now, when the blob hits the ghost, you do this:

```
ghost.hit = true;
```

```
ghost.active = false;
```

But you **never change** **ghost.x** or **ghost.y**, and you never turn **active** back on.

So the ghost:

- gets deactivated
 - never respawns
 - never moves
-

2 The silent killer bug (this one really matters)

This line:

```
if ((ghost.hit = true)) {
```

✗ What this actually does

- `=` is **assignment**, not comparison
- You are setting `ghost.hit` to `true` every frame
- The `if` condition always succeeds

✓ What you *meant* to write

```
if (ghost.hit === true) {
```

...but even with that fixed, this logic still won't move the ghost.

3 Correct way to respawn the ghost in a new random position

Instead of juggling `hit` + `active`, the cleanest pattern is:

👉 On collision, immediately move the ghost somewhere new

✓ Replace your ghost collision block with this:

```
// --- Blob ↔ Ghost interaction ---
```

```
if (overlap(box, ghost)) {  
    respawnGhost();  
  
    triggerWobbleSpike(); // we'll define this below  
}
```

Add this helper function:

```
function respawnGhost() {  
  
    ghost.x = random(40, width - ghost.w - 40);  
  
    ghost.y = random(40, floorY3 - ghost.h - 20);  
  
}
```

💡 Why this works:

- No flags needed
- Ghost stays active
- Ghost instantly jumps to a new spot

You can now **delete**:

ghost.active

ghost.hit

They're no longer needed.

4 Temporary aggressive noise / wobble effect (the fun part)

We'll do this using a **timer-based spike**, which is the standard game-dev approach.

Step 1: add wobble state to the blob

Add these to `blob3`:

wobbleBase: 0.9, // normal noise frequency

wobbleBoost: 2.2, // aggressive spike

wobbleTimer: 0, // how long the spike lasts

Update the existing property:

wobbleFreq: 0.9,

Step 2: trigger the spike on ghost hit

Add this function:

```
function triggerWobbleSpike() {  
    blob3.wobbleFreq = blob3.wobbleBoost;  
    blob3.wobbleTimer = 20; // frames (~0.3 seconds)  
}
```

Step 3: decay the wobble back to normal

Inside `drawMainSection()` before drawing the blob:

```
// --- Wobble cooldown ---  
  
if (blob3.wobbleTimer > 0) {  
    blob3.wobbleTimer--;  
  
    if (blob3.wobbleTimer === 0) {  
        blob3.wobbleFreq = blob3.wobbleBase;  
    }  
}
```

✨ Result:

- Blob freaks out briefly on contact

- Smoothly returns to normal
 - No permanent mutation
-

5 Cleaned-up ghost drawing logic

You no longer need this:

```
if ((ghost.hit = true)) {  
    drawGhost(ghost);  
    ghost.hit = false;  
}
```

Just keep:

```
drawGhost(ghost);
```

Conceptual takeaway (important)

You were *thinking* in terms of “states” (**hit**, **active**) — which is good — but for collectibles / triggers:

It’s often simpler to immediately mutate position instead of toggling visibility.

That keeps your game loop clean and avoids “dead” entities.

If you want next-level polish, we can:

- add easing to the wobble decay
- make the ghost flee instead of teleport
- add a sound-reactive wobble
- tie wobble intensity to collision speed

Just say the word 🤖💥