

GenAI Documentation

Name: Emerson Sanchez

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

Date Used: February 8th

Tool Disclosure: ChatGPT 5.2

Purpose of Use: To set a different position of the blob as well as add parts of code to make the blob bounce when it hits a wall.

Summary of Interaction: Provided code for the origin position of the blob and code for when the blob hits the wall.

Human Decision Point(s): I decided to take the generated AI code, but changed the values to fit with the theme of the side quest.

Integrity & Verification Note: Provided GenAI the code and added context to ensure that the output is accurate.

Scope of GenAI Use: GenAI contributed to the creation of two parts of the code that are integral to the theme of the side quest.

Limitations or Misfires: The tool inaccurately made some of the values of the code too low to significantly impact the theme.

Summary of Process (Human + Tool)

- Used the code provided by GenAI; however changed a few of the values to match the theme of the side quest.

Decision Points & Trade-offs

Describe one or two key decisions you made:

- Took the code directly from GenAI.

Appendix

<https://chatgpt.com/share/698803e9-eb60-8004-8dde-d70bb0c76ec7>

GenAI Code

```
// Start the blob resting on the top
blob3.y = blob3.r + 2;
blob3.vy = 0;
blob3.onGround = false;
```

```
// Keep blob inside the canvas horizontally
// --- Bounce off left/right canvas walls ---
if (blob3.x - blob3.r < 0) {
    blob3.x = blob3.r;
    blob3.vx *= -50; // bounce + energy loss
}

if (blob3.x + blob3.r > width) {
    blob3.x = width - blob3.r;
    blob3.vx *= -50; // bounce + energy loss
}
```

Chat history

You said:

```
// 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: 0.1,
```

```
  tSpeed: 0.004,
```

```
  // 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: 10, y: floorY3 - 80, w: 120, h: 12 }, // low step

  { x: 220, y: floorY3 - 100, w: 90, h: 7 }, // mid step

  { x: 440, y: floorY3 - 180, w: 80, h: 12 }, // high step

  { x: 520, y: floorY3 - 65, w: 90, h: 12 }, // return ramp

  { x: 600, y: floorY3 - 130, w: 1, h: 1 }, // a really small hidden ramp

];


// Start the blob resting on the floor

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

}


function draw() {

  background(69, 82, 102);


  // --- Draw all platforms ---

```

```

fill(29, 30, 31);

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 -= 0.4; // A or ←
if (keyIsDown(68) || keyIsDown(RIGHT_ARROW)) move += 0.4; // 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

*/

How to start the blob at the top?

ChatGPT said:

Easy tweak 👍

Right now you're explicitly spawning the blob on the floor in setup():

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

✅ Start the blob at the top

Replace that line with this:

```
blob3.y = blob3.r + 2; // just below the top edge
```

```
blob3.vy = 0;
```

```
blob3.onGround = false;
```

That's it. Gravity will immediately take over and the blob will fall naturally onto the first platform it hits.

Optional extras (nice polish)

If you want it centered horizontally too:

```
blob3.x = width / 2;
```

If you want a drop-in feeling (slight downward motion):

```
blob3.vy = 1.5;
```

Why this works

blob3.y is the center of the blob

So the top-safe position is radius + a tiny buffer

Collision + gravity logic already handles the rest

If you want the blob to start on a specific platform instead (like the highest one), say the word and I'll show you a clean helper function for that 🙄

You said:

```
// 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: 0.1,
```

```
  tSpeed: 0.004,
```

```
  // 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: 10, y: floorY3 - 180, w: 120, h: 12 }, // low step

  { x: 220, y: floorY3 - 100, w: 90, h: 7 }, // mid step

  { x: 440, y: floorY3 - 180, w: 80, h: 12 }, // high step

  { x: 520, y: floorY3 - 65, w: 90, h: 12 }, // return ramp

  { x: 600, y: floorY3 - 130, w: 10, h: 12 }, // small ramp

];


// Start the blob resting on the top

blob3.y = blob3.r + 2;

blob3.vy = 0;

blob3.onGround = false;

}
```

```
function draw() {  
  background(69, 82, 102);  
  
  // --- Draw all platforms ---  
  fill(29, 30, 31);  
  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 -= 0.4; // A or ←  
  if (keyIsDown(68) || keyIsDown(RIGHT_ARROW)) move += 0.4; // 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(42, 59, 144);
```

```
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

*/

Is there a way to make the blob bounce when it hits the canvas width on the left and right?

ChatGPT said:

Yep — totally doable, and it fits nicely into your existing physics 👍

Right now you're clamping the blob to the canvas edges:

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

That stops it. To make it bounce, you want to:

Detect when it hits the left/right wall

Push it back inside

Reverse its horizontal velocity (vx)

✅ Simple wall bounce (recommended)

Replace this part at the end of draw():

```
// Keep blob inside the canvas horizontally
```

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

⬇ With this instead:

```
// --- Bounce off left/right canvas walls ---
```

```
if (blob3.x - blob3.r < 0) {
```

```
    blob3.x = blob3.r;
```

```
    blob3.vx *= -0.8; // bounce + energy loss
```

```
}
```

```
if (blob3.x + blob3.r > width) {
```

```
    blob3.x = width - blob3.r;
```

```
    blob3.vx *= -0.8; // bounce + energy loss
```

}

🎯 What this does

* -1 → reverses direction

0.8 → keeps it from bouncing forever (feels more natural)

Blob stays fully inside the canvas

Works cleanly with your existing friction + acceleration

🔧 Tuning the feel

Want different vibes?

Bouncy / rubbery

```
blob3.vx *= -1.0;
```

Soft / squishy

```
blob3.vx *= -0.5;
```

Chaotic pinball

```
blob3.vx *= -1.1;
```

✨ Extra polish idea (optional)

If you want the blob to visually squash when it hits the wall, you can momentarily increase wobble or reduce r for a few frames — your blob system is already perfect for that.

If you want, I can show you:

wall-specific squash/stretch

angle-based bounces

sound-triggered bounces

camera shake on impact

Just say the word 👁👁