Sidequest 3

Project/Assignment Decisions

Because the task was to build an interactive story that unfolds through multiple game states I built a game where you go through doors and if you go through too many wrong ones you lose. I used ChatGPT to generate the code for this project. First I asked it to generate the code for the game, then asked it to split it into different files. With that the basic structure of my code was ready. Next I asked for some UX improvements, like adding a hover to the doors and adding hearts so you can visually keep track of your lives. There was some trouble with the visibility of the hearts due to the order of the code, so I had to do some debugging to fix it. Finally I asked AI to add a confetti and rain effect on win and lose for some extra embellishment to provide feedback. The code ChatGPT first gave me broke the game, so I had to do some extra debugging to solve it.

Role-Based Process Evidence

Initial Commit
Name: Cherry
Primary responsibility: Complete sidequest
Goal: Initial commit
I used the examples provided in class as a template

Add files
Name: Cherry
Primary responsibility: Complete sidequest
Goal: Add the different files for the game states
I prompted ChatGPT to build the game and generate the different files

UX Improvements
Name: Cherry
Primary responsibility: Complete sidequest
Goal: Add hover effect, hearts, and color signals
I prompted ChatGPT to improve the UX of the game, like adding more hearts

Hearts

```
function drawHeart(x, y, s) {
  push();
  translate(x, y);
  scale(s / 20);

  noStroke();
  fill(255, 60, 60);

  beginShape();
  vertex(0, -10);
  bezierVertex(-10, -20, -25, -5, 0, 20);
  bezierVertex(25, -5, 10, -20, 0, -10);
  endShape(CLOSE);

  pop();
}

function drawHearts() {
  let startX = width / 2 - (health - 1) * 20;
  let y = 70;
  let size = 20;

  for (let i = 0; i < health; i++) {
    drawHeart(startX + i * 40, y, size);
  }
}
```

Hover effect

```
if (hoverDoor === 0) {
    fill(230);
    stroke(255);
    strokeWeight(4);
  } else {
    fill(180);
    noStroke();
  }
  rect(100, 150, 150, 200);

  if (hoverDoor === 1) {
    fill(230);
    stroke(255);
    strokeWeight(4);
  } else {
    fill(180);
    noStroke();
  }
  rect(350, 150, 150, 200);
```

Lose background

```
function drawLose() {
  background(50, 20, 20);
```

Win background

```
function drawWin() {
  background(20, 120, 60);
```

Win and Lose effect

Name: Cherry
Primary responsibility: Complete sidequest
Goal: Add rain effect on loss and confetti on win
I prompted ChatGPT to add these effects and had to debug the effects on game restart

Rain effect

```
for (let i = 0; i < 5; i++) {
    rain.push(newDrop());
  }

  // Update and display rain
  for (let i = rain.length - 1; i >= 0; i--) {
    rain[i].update();
    rain[i].show();

    // Remove raindrops off the screen
    if (rain[i].y > height) {
      rain.splice(i, 1);
    }
  }
}

function newDrop() {
  return {
    x: random(width),
    y: -10,
    len: random(10, 20),
    speed: random(4, 10),

    update: function () {
      this.y += this.speed;
    },

    show: function () {
      stroke(150, 150, 255); // Light blue rain
      strokeWeight(2);
      line(this.x, this.y, this.x, this.y + this.len);
    },
  };
```

Confetti effect

```
for (let i = 0; i < 5; i++) {
    confetti.push(newConfetti());
  }
  // Update and display confetti
  for (let i = confetti.length - 1; i >= 0; i--) {
    confetti[i].update();
    confetti[i].show();

    // Remove confetti that falls off screen
    if (confetti[i].y > height) {
      confetti.splice(i, 1);
    }
  }
}

function newConfetti() {
  return {
    x: random(width),
    y: -10,
    size: random(5, 12),
    speed: random(2, 6),
    color: color(random(255), random(255), random(255)),

    update: function() {
      this.y += this.speed;
    },

    show: function() {
      fill(this.color);
      noStroke();
      rect(this.x, this.y, this.size, this.size);
    }
  };
}
```

GenAI Documentation
Date used: Feb 2, 2026
Tool used: ChatGPT 5.2
Purpose: To generate code and to debug
Summary of interaction: Helped generate code for the sidequest

Human Decision Points: ChatGPT generated the initial extra game files but I asked ChatGPT for more improvements like the hover effect and the hearts display to improve the game.

Integrity & Verification Note: I had to playtest the code after adding or changing code based on ChatGPT's output to make sure it works

Scope of GenAI Use: I used ChatGPT to write code and debug. I told ChatGPT what code to generate or what to add to the project.

Limitations or Misfires: Sometimes the code snippets ChatGPT generates ChatGPT doesn't mention the order that you copy paste it in the file matters, which results in the code not working. This happened while I was adding the hearts to the game so I had to debug and the issue was the order I put the hearts snippet into my file.

Appendix
Prompt: p5.js coding make a game where you choose between 2 doors 3 times. if u go through the wrong door u lose health if u go through the right one u dont. if u die u lose if u survive u win structure it similarly to this: https://k2cochra.github.io/GBDA302_W3_Example01/

Response: Here's a **p5.js game** that follows the structure of the example you linked — you choose between **2 doors three times**, lose health if you pick the wrong door, and win if you survive all three picks.

📦 Project Overview

- You have **3 room attempts**
- Each room has **2 doors**: left and right
- One door is correct, the other damages health
- Starting health: **3**
- If health hits **0** → **Lose**
- If you survive all 3 rooms → **Win**

🧠 How It Works

1. The game shows **two doors** side-by-side.
2. You click left or right.
3. The game checks if the choice is correct or wrong.

4. Wrong choices → health–1.
5. After 3 choices either **Win** or **Lose** screen.

🧑‍💻 Full p5.js Code

```
let health = 3;
let room = 1;
let gameState = "playing"; // "playing", "win", "lose"
let correctDoor; // 0 = left, 1 = right

function setup() {
 createCanvas(600, 400);
 textAlign(CENTER, CENTER);
 newRoom();
}

function draw() {
 background(30);

 if (gameState === "playing") {
   drawDoors();
   drawStatus();
 } else if (gameState === "win") {
   textSize(48);
   fill(0, 255, 100);
   text("YOU SURVIVED!", width / 2, height / 2);
 } else if (gameState === "lose") {
   textSize(48);
   fill(255, 0, 0);
   text("YOU LOST!", width / 2, height / 2);
 }
}

function drawDoors() {
 // left door
 fill(200);
 rect(100, 150, 150, 200);
 // right door
 fill(200);
 rect(350, 150, 150, 200);
 textSize(24);
 fill(0);
 text("LEFT", 175, 300);
 text("RIGHT", 425, 300);
}

function drawStatus() {
 textSize(20);
 fill(255);
 text("Health: " + health, width / 2, 50);
 text("Room: " + room + " / 3", width / 2, 80);
}

// Choose the correct door randomly
function newRoom() {
 correctDoor = floor(random(2)); // 0 or 1
}

function mousePressed() {
 if (gameState !== "playing") return;

 // check left door
 if (
   mouseX > 100 && mouseX < 250 &&
   mouseY > 150 && mouseY < 350
 ) {
   handleChoice(0);
 }
 // check right door
 if (
   mouseX > 350 && mouseX < 500 &&
   mouseY > 150 && mouseY < 350
 ) {
   handleChoice(1);
 }
}

function handleChoice(choice) {
 if (choice !== correctDoor) {
   health--;
 }

 room++;
 if (health <= 0) {
   gameState = "lose";
 } else if (room > 3) {
   gameState = "win";
 } else {
   newRoom();
 }
}
```

🕹️ Controls

- **Click left or right door** to choose
- The game automatically moves to the next room
- Incorrect choice reduces health by 1

🧩 Customize It

You can easily expand this by:

✅ Adding images for doors
✅ Showing animations on win/lose
✅ Adding sound effects
✅ Different health or more rooms

If you want **buttons instead of clicking squares**, or **sprites** for doors, let me know! 🎮✨

Prompt: I need to have different files for win lose instructions and etc
Response: Got it — you want this set up **properly for a GBDA / p5.js project**, with **separate files** for **instructions, gameplay, win, and lose states**, like the example site 👍
Below is a **clean, structured multi-file setup** that mirrors that style.

📁 File Structure

project-folder/

```
|
├── index.html
├── sketch.js       // main controller
├── instructions.js
├── game.js
├── win.js
├── lose.js
└── style.css       // optional
```

🌐 `index.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8" />
 <title>Door Game</title>

 <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.9.0/p5.js"></script>

 <script src="instructions.js"></script>
 <script src="game.js"></script>
 <script src="win.js"></script>
 <script src="lose.js"></script>
 <script src="sketch.js"></script>

 <link rel="stylesheet" href="style.css">
</head>
<body>
</body>
</html>
```

> ⚠️ **Script order matters** — state files first, `sketch.js` last.

🎮 `sketch.js` (State Controller)

```
let state = "instructions";

function setup() {
 createCanvas(600, 400);
 textAlign(CENTER, CENTER);
}
```

```
function draw() {
 background(30);

 if (state === "instructions") {
   instructions();
 } else if (state === "game") {
   game();
 } else if (state === "win") {
   win();
 } else if (state === "lose") {
   lose();
 }
}

function mousePressed() {
 if (state === "instructions") {
   state = "game";
   startGame();
 } else if (state === "game") {
   gameMousePressed();
 }
}
```

📖 instructions.js

```
function instructions() {
 fill(255);
 textSize(36);
 text("Door Game", width / 2, 100);

 textSize(18);
 text(
   "Choose between two doors.\n" +
   "Wrong door = lose health.\n" +
   "Survive 3 rooms to win.\n\n" +
   "Click to begin.",
   width / 2,
   220
 );
}
```

🚪 game.js

```
let health;
let room;
let correctDoor;

function startGame() {
 health = 3;
 room = 1;
 newRoom();
}

function game() {
 drawDoors();
 drawHUD();
}

function drawDoors() {
 fill(200);
 rect(100, 150, 150, 200);
 rect(350, 150, 150, 200);

 fill(0);
 textSize(24);
 text("LEFT", 175, 300);
 text("RIGHT", 425, 300);
}

function drawHUD() {
 fill(255);
 textSize(18);
 text("Health: " + health, width / 2, 40);
 text("Room: " + room + " / 3", width / 2, 70);
}

function newRoom() {
 correctDoor = floor(random(2));
}

function gameMousePressed() {
 // left door
 if (mouseX > 100 && mouseX < 250 && mouseY > 150 && mouseY < 350) {
   handleChoice(0);
 }
 // right door
 if (mouseX > 350 && mouseX < 500 && mouseY > 150 && mouseY < 350) {
   handleChoice(1);
 }
}

function handleChoice(choice) {
 if (choice !== correctDoor) {
   health--;
 }

 room++;

 if (health <= 0) {
   state = "lose";
 } else if (room > 3) {
   state = "win";
 } else {
   newRoom();
 }
}
```

🏆 `win.js`

```
function win() {
 background(20, 120, 60);
 fill(255);
 textSize(48);
 text("YOU WIN!", width / 2, height / 2);
}
```

💀 `lose.js`

```
function lose() {
 background(120, 20, 20);
 fill(255);
 textSize(48);
 text("YOU LOST", width / 2, height / 2);
}
```

🎨 (Optional) `style.css`

```
body {
 margin: 0;
 background-color: black;
}
```

✅ Why This Matches the Example

✔ Uses **state-based structure**
✔ Each screen is its **own file**
✔ `sketch.js` acts as the **controller**
✔ Easy to expand with sound, images, transitions

If you want:

- **Keyboard input instead of mouse**
- **Images instead of rectangles**
- **Restart button**
- **Fade transitions between states**

Just tell me — this is already set up perfectly for it 👌

Prompt: what should index.html look like

Here's a **clean, correct `index.html`** that matches the **GBDA p5.js multi-state structure** (like the example you referenced) and works with your **separate files for instructions, game, win, and lose**.

You can copy-paste this exactly.

✅ `index.html`

```
<!DOCTYPE html>
<html lang="en">
```

```
  <head>
    <meta charset="utf-8" />

    <title>Three Doors Game</title>

    <!-- p5.js library -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.9.0/p5.js"></script>

    <!-- Game state files -->
    <script src="instructions.js"></script>
    <script src="game.js"></script>
    <script src="win.js"></script>
    <script src="lose.js"></script>

    <!-- Main sketch (controller) -->
    <script src="sketch.js"></script>

    <style>
      body {
        margin: 0;
        padding: 0;
        background-color: black;
      }
    </style>
  </head>

  <body>
    <!-- p5.js canvas will be injected here -->
  </body>
</html>
```

🔑 Important Notes (GBDA-Style)

1️⃣ Script Order Matters

- Load **p5.js first**
- Load **state files next**
- Load **`sketch.js` last**
  → `sketch.js` controls which state is shown

2️⃣ No Canvas in HTML

- p5.js automatically creates the canvas using `createCanvas()`
- Don’t manually add `<canvas>`

3️⃣ Keep HTML Simple

GBDA examples expect:

- Minimal HTML
- Logic handled entirely in JavaScript
- Visuals handled in p5

📁 Final File Checklist

index.html ✅
sketch.js ✅
instructions.js
game.js
win.js
lose.js

If you want, I can:

- Compare this **line-by-line with the GBDA example**
- Add a **restart button**
- Add **preload() for images**
- Add **sound files** the “correct” p5 way

Just say the word 👌

Prompt: add a hover effect so you know which door your choosing and hearts on the bottom corner so you can see if you lose a heart or not

Response: Perfect additions 👍 — both are **very GBDA-appropriate UX improvements**.

Below I'll show you **exactly what to add/change**, keeping your **multi-file structure** intact and clean.

1️⃣ Hover Effect on Doors

We'll:

- Detect which door the mouse is over
- Change the door color + outline on hover

✨ What it looks like

- Normal door → gray
- Hovered door → lighter + outlined
- Clear visual feedback before clicking

✏️ Update `game.js`

🔹 Add this near the top (globals)

```
let hoverDoor = -1; // -1 = none, 0 = left, 1 = right
```

🔹 Replace `drawDoors()` with this version

```
function drawDoors() {
 hoverDoor = -1;

 // LEFT DOOR
 if (mouseX > 100 && mouseX < 250 && mouseY > 150 && mouseY < 350) {
  hoverDoor = 0;
 }

 // RIGHT DOOR
 if (mouseX > 350 && mouseX < 500 && mouseY > 150 && mouseY < 350) {
   hoverDoor = 1;
 }

 // Left door visuals
 if (hoverDoor === 0) {
   fill(230);
   stroke(255);
   strokeWeight(4);
 } else {
   fill(180);
   noStroke();
 }
 rect(100, 150, 150, 200);

 // Right door visuals
 if (hoverDoor === 1) {
   fill(230);
   stroke(255);
   strokeWeight(4);
 } else {
   fill(180);
   noStroke();
 }
 rect(350, 150, 150, 200);

 // Labels
 noStroke();
 fill(0);
 textSize(24);
 text("LEFT", 175, 300);
 text("RIGHT", 425, 300);
}
```

✅ Now the player **knows exactly which door they're choosing**

2️⃣ Hearts for Health (Bottom Corner)

We’ll:

- Draw ❤️ icons instead of text
- Remove a heart when health is lost
- Display them in the **bottom-left corner**

❤️ Add this function to `game.js`

```
function drawHearts() {
 textSize(28);
 textAlign(LEFT, CENTER);

 for (let i = 0; i < health; i++) {
  fill(255, 0, 0);
  text("♥", 20 + i * 30, height - 30);
 }

 textAlign(CENTER, CENTER); // reset
}
```

🔁 Update `drawHUD()`

Replace it with:

```
function drawHUD() {
 fill(255);
 textSize(18);
 text("Room: " + room + " / 3", width / 2, 40);
 drawHearts();
}
```

❌ You can now **remove the text-based health display**
✔ Visual, intuitive, and readable at a glance

✅ Resulting UX Improvements

✔ Clear hover affordance (perceived affordance win)
✔ Immediate feedback before committing
✔ Hearts = emotional + visual health indicator
✔ Supports accessibility & clarity

This is **exactly the kind of thing instructors like to see** in GBDA projects.

If you want next:

- 💔 Heart loss animation
- 🔊 Sound on wrong door
- 🚪 Door shake on damage
- 🔁 Restart button on win/lose screens

Just tell me — this setup is already perfect for extending.

Prompt: draw the hearts out

Response: Perfect — we'll **draw actual heart shapes** using p5.js geometry instead of the `"♥"` character.

Below is a **clean, reusable heart-drawing solution** that fits right into your existing `game.js`.

❤️1️⃣Add a Heart Shape Function

Put this **anywhere in `game.js`** (below your other functions is fine):

```
function drawHeart(x, y, size) {
 fill(220, 50, 50);
 noStroke();

 beginShape();
 vertex(x, y);
 bezierVertex(x - size / 2, y - size / 2, x - size, y + size / 3, x, y + size);
 bezierVertex(x + size, y + size / 3, x + size / 2, y - size / 2, x, y);
 endShape(CLOSE);
}
```

This draws a **smooth, recognizable heart** using Bézier curves.

❤️2️⃣ Replace `drawHearts()` with This Version

```
function drawHearts() {
 let startX = 30;
 let y = height - 40;
 let size = 20;

 for (let i = 0; i < health; i++) {
   drawHeart(startX + i * 35, y, size);
 }
}
```

📍 Hearts appear in the **bottom-left corner**
📉 Losing health = one heart disappears
🎮 Clean visual feedback

🧠 Why This Is Better (Design-wise)

- No reliance on fonts or emojis
- Consistent across browsers
- Counts as **custom UI elements**
- Strong perceived affordance + clarity

This is **exactly the level of polish** expected in a GBDA interactive prototype.

If you want next:

- 💔 Broken heart animation when damaged

- ❤️ Heart pulse on hover
- 🎨 Stylized pixel hearts
- 🔊 Sound synced to heart loss

Just say the word 👍

Prompt: the hearts aren't appearing

Response: This is a **very common p5.js issue** — nothing is “wrong” conceptually, it’s just **draw order + coordinates + shape math**.

Let’s fix it **cleanly and reliably**.

✅ Why the hearts aren’t appearing

There are **3 likely causes**:

1️⃣ The heart shape is being drawn *off-canvas*

Your `drawHeart()` function uses the heart’s **top point as the origin**, but you’re positioning it near the bottom of the canvas → most of the heart is below the screen.

2️⃣ The heart color blends into the background

Dark red on dark background = invisible.

3️⃣ `drawHearts()` isn’t being called last

If doors draw after hearts, they can cover them.

🛠️ Guaranteed Working Fix

🔴 Step 1: Replace `drawHeart()` with this version

This version draws the heart **centered**, not from the tip.

```
function drawHeart(x, y, s) {
 push();
 translate(x, y);
 scale(s / 20);

 noStroke();
 fill(255, 60, 60);

 beginShape();
 vertex(0, -10);
 bezierVertex(-10, -20, -25, -5, 0, 20);
 bezierVertex(25, -5, 10, -20, 0, -10);
 endShape(CLOSE);

 pop();
}
```

🔴 Step 2: Replace `drawHearts()` with this

```
function drawHearts() {
 let startX = 40;
 let y = height - 40;

 for (let i = 0; i < health; i++) {
   drawHeart(startX + i * 35, y, 20);
 }
}
```

🔴 Step 3: Make sure `drawHearts()` is called after doors

Your `game()` function should look like this:

```
function game() {
 drawDoors();
 drawHUD();
 drawHearts();
}
```

(Or `drawHearts()` inside `drawHUD()` — either is fine.)

🔍 Quick Debug Check (Optional)

If you want to *verify the function is running*, temporarily add:

console.log("Health:", health);

inside `drawHearts()` — you should see logs updating when health changes.

✅ Result

✔ Hearts appear clearly in bottom-left
✔ Hearts disappear when health drops
✔ Consistent across browsers
✔ No emoji/font issues

This is now **production-safe p5.js drawing code**.

If you want:

- 💔 A heart briefly flashes when lost
- ❤️ Hearts shake on damage
- 🎨 Pixel-style hearts
- 🔁 Restart button with heart reset

Say the word — you're *very* close to a polished GBDA submission.

Prompt: show me the final game.js file

Response: Here is the **complete, final `game.js` file** with:

✅ 3 rooms
✅ Health system
✅ **Hover effect on doors**
✅ **Properly drawn hearts (working + visible)**
✅ Clean state-based logic that plugs into your existing `sketch.js`

You can **replace your entire `game.js` with this**.

✅ FINAL `game.js`

```
// --------------------
// Game Variables
// --------------------
let health;
let room;
let correctDoor;
let hoverDoor = -1; // -1 = none, 0 = left, 1 = right

// --------------------
// Game Setup
// --------------------
function startGame() {
 health = 3;
 room = 1;
 newRoom();
}

// --------------------
// Main Game Loop
// --------------------
function game() {
 background(30);

 drawDoors();
 drawHUD();
 drawHearts();
}

// --------------------
// Doors + Hover Effect
// --------------------
function drawDoors() {
 hoverDoor = -1;

 // Detect hover
 if (mouseX > 100 && mouseX < 250 && mouseY > 150 && mouseY < 350) {
   hoverDoor = 0;
 }
 if (mouseX > 350 && mouseX < 500 && mouseY > 150 && mouseY < 350) {
   hoverDoor = 1;
 }

 // LEFT DOOR
 if (hoverDoor === 0) {
   fill(230);
   stroke(255);
   strokeWeight(4);
 } else {
   fill(180);
   noStroke();
 }
 rect(100, 150, 150, 200);

 // RIGHT DOOR
 if (hoverDoor === 1) {
   fill(230);
   stroke(255);
   strokeWeight(4);
 } else {
   fill(180);
   noStroke();
 }
 rect(350, 150, 150, 200);

 // Labels
 noStroke();
 fill(0);
 textSize(24);
 text("LEFT", 175, 300);
 text("RIGHT", 425, 300);
}

// --------------------
// HUD (Room Counter)
// --------------------
function drawHUD() {
 fill(255);
 textSize(18);
 textAlign(CENTER, CENTER);
 text("Room: " + room + " / 3", width / 2, 40);
}
```

```
// --------------------
// Hearts (Health)
// --------------------
function drawHearts() {
 let startX = 40;
 let y = height - 40;

 for (let i = 0; i < health; i++) {
   drawHeart(startX + i * 35, y, 20);
 }
}

// Draws a centered heart shape
function drawHeart(x, y, s) {
 push();
 translate(x, y);
 scale(s / 20);

 noStroke();
 fill(255, 60, 60);

 beginShape();
 vertex(0, -10);
 bezierVertex(-10, -20, -25, -5, 0, 20);
 bezierVertex(25, -5, 10, -20, 0, -10);
 endShape(CLOSE);

 pop();
}

// --------------------
// Game Logic
// --------------------
function newRoom() {
 correctDoor = floor(random(2)); // 0 = left, 1 = right
}

function gameMousePressed() {
 // Left door click
 if (mouseX > 100 && mouseX < 250 && mouseY > 150 && mouseY < 350) {
   handleChoice(0);
 }

 // Right door click
 if (mouseX > 350 && mouseX < 500 && mouseY > 150 && mouseY < 350) {
  handleChoice(1);
 }
}

function handleChoice(choice) {
 if (choice !== correctDoor) {
  health--;
 }

 room++;

 if (health <= 0) {
  state = "lose";
 } else if (room > 3) {
  state = "win";
 } else {
  newRoom();
 }
}
```

🧪 Works With These Files

✔ `sketch.js` (state controller)
✔ `instructions.js`
✔ `win.js`
✔ `lose.js`

No changes needed elsewhere.

🏁 What You've Got Now (GBDA-wise)

- Clear **perceived affordance** (hover feedback)
- Visual health system (not text-based)
- Clean separation of concerns
- Extendable for sound, animation, restart

If you want, next I can help you:

- Add a **restart button**
- Add **door animations**
- Write a **design rationale paragraph** for your submission
- Match **Week 3 language (affordance, feedback, constraints)**

Just tell me 👍

Prompt:
let health; let room; let correctDoor; let hoverDoor = -1; // -1 = none, 0 = left, 1 = right function resetGame() { health = 3; room = 1; newRoom(); } function drawGame() { drawDoors(); drawHUD(); } function drawDoors() { hoverDoor = -1; // LEFT DOOR if (mouseX > 100 && mouseX < 250 && mouseY > 150 && mouseY < 350) { hoverDoor = 0; } // RIGHT DOOR if (mouseX > 350 && mouseX < 500 && mouseY > 150 && mouseY < 350) { hoverDoor = 1; } // Left door visuals if (hoverDoor === 0) { fill(230); stroke(255); strokeWeight(4); } else { fill(180); noStroke(); } rect(100, 150, 150, 200); // Right door visuals if (hoverDoor === 1) { fill(230); stroke(255); strokeWeight(4); } else { fill(180); noStroke(); } rect(350, 150, 150, 200); // Labels noStroke(); fill(0); textSize(24); text("LEFT", 175, 300); text("RIGHT", 425, 300); } function drawHeart(x, y, s) { push(); translate(x, y); scale(s / 20); noStroke(); fill(255, 60, 60); beginShape(); vertex(0, -10); bezierVertex(-10, -20, -25, -5, 0, 20); bezierVertex(25, -5, 10, -20, 0, -10); endShape(CLOSE); pop(); } function drawHearts() { let startX = 40; let y = height - 40; for (let i = 0; i < health; i++) { drawHeart(startX + i * 35, y, 20); } } function drawHUD() { fill(255); textSize(18); text("Health: " + health, width / 2, 40); text("Room: " + room + " / 3", width / 2, 70); } function newRoom() { correctDoor = floor(random(2)); // 0 = left, 1 = right } function gameMouse() { // left door if (mouseX > 100 && mouseX < 250 && mouseY > 150 && mouseY < 350) { checkChoice(0); } // right door if (mouseX > 350 && mouseX < 500 && mouseY > 150 && mouseY < 350) { checkChoice(1); } } function checkChoice(choice) { if (choice !== correctDoor) { health--; } room++; if (health <= 0) { state = "lose"; } else if (room > 3) { state = "win"; } else { newRoom(); } } I dont see the hearts just the Health 3 sign

Response: Ah! I see exactly what's happening — in your `drawHUD()` function, you are **still drawing the text** **`"Health: "`**, which will show on screen, but you never call `drawHearts()` anywhere inside `drawGame()` or after `drawHUD()`. That's why you only see the text and not the hearts.

You need to:

1. Remove or comment out the text-based health display in `drawHUD()`.

2. Call `drawHearts()` **after drawing the HUD**, so the hearts render on top of the background and doors.

Here’s a **fixed version** of the relevant parts:

✅ Updated `drawHUD()` and `drawGame()`

```
function drawGame() {
 drawDoors();
 drawHUD();     // Draw room info only
 drawHearts();  // Draw hearts on top
}

function drawHUD() {
 fill(255);
 textSize(18);
 textAlign(CENTER, CENTER);
 // Only show room now; remove text health
 text("Room: " + room + " / 3", width / 2, 40);
}
```

🔹 Important Points

- **`drawHearts()` must be called every frame** so it updates with health changes.
- Your old `text("Health: " + health, ...)` was **blocking attention** from the drawn hearts. Remove it to avoid confusion.
- Make sure `drawGame()` is being called from `draw()` in `sketch.js` while in the `"game"` state.

🔹 Example `draw()` in `sketch.js`

```
function draw() {
 background(30);

 if (state === "instructions") {
   instructions();
 } else if (state === "game") {
   drawGame();  // now draws doors + HUD + hearts
 } else if (state === "win") {
   win();
 } else if (state === "lose") {
   lose();
 }
}
```

✅ After this, you should see **red heart shapes at the bottom-left** and the **room counter at the top**, and the hearts will **disappear as health decreases**.

If you want, I can also **adjust the hearts so they never overlap with doors and look nicely aligned**, like in a professional GBDA prototype.

Do you want me to do that next?

Prompt: move the hearts under the room text

Response: Perfect! Let’s **move the hearts under the room text** so they’re at the **top center**, like a proper HUD. We’ll also make sure they **align nicely**.

Here’s how to fix it:

1️⃣Update `drawHearts()`

```
function drawHearts() {
 let startX = width / 2 - (health - 1) * 20; // center hearts under room text
 let y = 70; // below the room counter
 let size = 20;

 for (let i = 0; i < health; i++) {
   drawHeart(startX + i * 40, y, size);
 }
}
```

Explanation:

- `width / 2` → center horizontally
- `- (health - 1) * 20` → offsets hearts so they are **centered as a group**, no matter how many you have

- `y = 70` → just under the room text (which was at y = 40)
- `i * 40` → spacing between hearts

2️⃣Update `drawHUD()` (room text only)

```
function drawHUD() {
 fill(255);
 textSize(18);
 textAlign(CENTER, CENTER);
 text("Room: " + room + " / 3", width / 2, 40);
}
```

- No need for `"Health: "` anymore because hearts now show health visually

---

3️⃣Make sure `drawHearts()` is called

```
function drawGame() {
 drawDoors();
 drawHUD();
 drawHearts();
}
```

✅ Result:

- **Room text** at the top center
- **Hearts** directly below it
- As health decreases, hearts disappear **visually**
- Looks clean and professional

If you want, I can also **make the hearts “pulse” when you lose one** — gives it a nice polished GBDA effect.

Do you want me to add that next?

Prompt:
let health; let room; let correctDoor; let hoverDoor = -1; // -1 = none, 0 = left, 1 = right function resetGame() { health = 3; room = 1; newRoom(); } function drawDoors() { hoverDoor = -1; // Left door if (mouseX > 100 && mouseX < 250 && mouseY > 150 && mouseY < 350) { hoverDoor = 0; } // Right door if (mouseX > 350 && mouseX < 500 && mouseY > 150 && mouseY < 350) { hoverDoor = 1; } // Left door visuals if (hoverDoor === 0) { fill(230); stroke(255); strokeWeight(4); } else { fill(180); noStroke(); } rect(100, 150, 150, 200); // Right door visuals if (hoverDoor === 1) { fill(230); stroke(255); strokeWeight(4); } else { fill(180); noStroke(); } rect(350, 150, 150, 200); // Labels noStroke(); fill(0); textSize(24); text("LEFT", 175, 300); text("RIGHT", 425, 300); } function drawHeart(x, y, s) { push(); translate(x, y); scale(s / 20); noStroke(); fill(255, 60, 60); beginShape(); vertex(0, -10); bezierVertex(-10, -20, -25, -5, 0, 20); bezierVertex(25, -5, 10, -20, 0, -10); endShape(CLOSE); pop(); } function drawHearts() { let startX = width / 2 - (health - 1) * 20; let y = 70; let size = 20; for (let i = 0; i < health; i++) { drawHeart(startX + i * 40, y, size); } } function drawGame() { drawDoors(); drawHUD(); drawHearts(); } function drawHUD() { fill(255); textSize(18); textAlign(CENTER, CENTER); text("Room: " + room + " / 3", width / 2, 40); } function newRoom() { correctDoor = floor(random(2)); } function gameMouse() { // left door if (mouseX > 100 && mouseX < 250 && mouseY > 150 && mouseY < 350) { checkChoice(0); } // right door if (mouseX > 350 && mouseX < 500 && mouseY > 150 && mouseY < 350) { checkChoice(1); } } function checkChoice(choice) { if (choice !== correctDoor) { health--; } room++; if (health <= 0) { state = "lose"; } else if (room > 3) { state = "win"; } else { newRoom(); } } is the correct door random every time?

Response: Yes — in your current code, the correct door **is random every time `newRoom()` is called**. Let me explain exactly why and how it works:

1️⃣ How `newRoom()` works

```
function newRoom() {
 correctDoor = floor(random(2)); // 0 = left, 1 = right
}
```

- `random(2)` → generates a decimal between 0 (inclusive) and 2 (exclusive), e.g., 0.0 ≤ x < 2.0
- `floor()` → rounds down to the nearest integer → results in **0 or 1 randomly**
- So every time `newRoom()` is called, `correctDoor` is either 0 (left) or 1 (right) randomly.

2️⃣ When `newRoom()` is called

```
function resetGame() {
 health = 3;
 room = 1;
 newRoom();
}

function checkChoice(choice) {
 if (choice !== correctDoor) {
   health--;
 }

 room++;

 if (health <= 0) {
   state = "lose";
 } else if (room > 3) {
   state = "win";
 } else {
   newRoom(); // ← correct door changes for the next room
 }
}
```

✅ So:

- When the game **starts**, `resetGame()` calls `newRoom()` → first room has a random correct door
- After each choice (if the game is not over), `checkChoice()` calls `newRoom()` again → the **next room gets a new random correct door**

3️⃣ Important Note

- Each room’s correct door is **independent** from the previous rooms
- That means you **cannot “learn” the pattern** — the player cannot predict the next correct door.
- This matches your intended design for a “3-room guessing game.”

If you want, I can also **modify it so the correct doors are random at the start but fixed per game** — so the player can remember which door is correct in earlier rooms. That sometimes makes gameplay more strategic.

Do you want me to do that?