Process & Decision Documentation

Project/Assignment Decisions

A key decision in this project was choosing what element to represent the player decision-making and stat tracking. Because the assignment bonus encouraged tracking a player stat, I decided to center the game around a familiar health (HP) mechanic. Health was chosen because it is intuitive and easy to understand, making it well suited for a small interactive game. This decision allowed player choices to feel meaningful, as each action directly affected survival and determined the final outcome.

Another important decision was setting the game within a night market food scenario. I chose this theme because it provided strong visual imagery and a playful, enjoyable atmosphere. The night market context also connects closely to the health mechanic through the familiar idea of “HP" as well, reframed as stomach health. This allowed food choices to feel intuitive and meaningful, while keeping the project’s scope manageable.

Role-Based Process Evidence

Name: Annora Zhu

Role(s): Designer, Developer

Primary responsibility for this work: Concept development, narrative design, interaction logic, state management, UI layout, and p5.js implementation.

Goal of Work Session

The goal of this work session was to create a short interactive game where the player makes choices that affect their health and lead to different endings. I focused on designing clear scenes, simple choices, and making sure the player’s decisions had visible consequences throughout the game.

Tools, Resources, or Inputs Used

- p5.js (JavaScript creative coding framework)
- Week 3 course template (multi-screen state-based structure)
- Lecture examples on game states and input routing
- ChatGPT (GenAI) for debugging, structural planning, and documentation refinement

GenAI Documentation

Date Used: Feb 1, 2026

Tool Disclosure: ChatGPT 5.2

Purpose of Use: GenAI was used to assist with planning the branching structure, debugging screen-routing logic, refining conditional health checks, and improving clarity and grammar in documentation and generating README text.

Summary of Interaction: The tool provided suggestions for organizing multiple scene files, safely routing mouse input across states, and structuring health-based endings. It was also used to help rewrite explanations in a more academic and concise tone.

Human Decision Point(s): I independently chose the game concept (*Night Market Adventure*), the health-based mechanic, and the narrative tone and design details. All final decisions regarding scene flow, health values, and endings were made by me. I adjusted or rejected suggestions that added unnecessary complexity or randomness in order to keep the interaction clear and player-driven.

Integrity & Verification Note: All GenAI-supported suggestions were manually implemented and tested in p5.js. Game flow, health transitions, and endings were verified through repeated playtesting to ensure that player choices consistently produced expected outcomes.

Scope of GenAI Use: GenAI did not determine the game concept, narrative theme, or final implementation. It served only as a support tool for refinement, debugging, and documentation clarity.

Limitations or Misfires: Some early suggestions relied on random outcomes, which conflicted with my goal of player agency. These were replaced with deterministic, choice-based logic tied directly to the health stat.

Summary of Process (Human + Tool)

I began with the provided Week 3 multi-screen template and analyzed its structure for managing game states and routing input. I then removed the random win/lose mechanic and replaced it with a branching narrative built around food choices at a night market.

Throughout development, I iteratively tested screen transitions, health updates, and endings to ensure the logic remained clear and consistent. Visual simplicity was prioritized so the focus stayed on decision-making rather than complex graphics.

Decision Points & Trade-offs

Decision: Replace random outcomes with choice-based logic

- **Why:** Random results reduced player agency and narrative clarity
- **Trade-off:** Required more explicit condition checks and additional scene files

Decision: Use a single tracked stat (health)

- **Why:** Health is intuitive and easy to communicate through food choices
  Trade-off: Limited mechanical complexity in favor of clarity and scope control

Decision: Create multiple endings based on health

- **Why:** Reinforces consequences of player decisions
  Trade-off: Required additional ending states and routing logic

Verification & Judgement

I evaluated the project through repeated playtesting, checking whether choices felt meaningful and whether health changes were understandable. I also assessed the project against the assignment's focus on state management, interaction clarity, and branching structure.

Appendix

Dispatch / Dixit Build an interactive story that unfolds through multiple game states and files, branching like a small decision tree. Bonus: Track a player stat (e.g., trust, health, karma) across scenes and unlock endings based on it this is my week 3 side quest prompt, the attached files is the example, so give me some inspiration of what game i can do, im thinking of to track the stat of health

Ooo this is a fun prompt 👀
You're basically being asked to turn that **win/lose button demo** into a **tiny narrative game with choices + states**, and adding **health** is a perfect move.

I'll give you **concept ideas**, then break one down into **how it maps cleanly onto your existing files** so it doesn't feel scary or over-engineered.

Big picture (what prof wants, lowkey)

Your example already has:

- multiple screens (`start`, `game`, `win`, `lose`)
- state routing via `currentScreen`

You just need to:

1. **Add story meaning** to each screen
2. **Branch based on choices**, not just random
3. **Track a stat (health)** across screens
4. **End differently based on health**

That's it. No AAA game required 😌

Game Concept Ideas (health-focused)

1️⃣"Last Day Before Midterms" (Relatable™)

You're a student trying to survive one chaotic day.

Health = energy / mental health

Scenes:

- Wake up late
- Decide whether to eat breakfast
- Decide whether to skip class or run

- Decide whether to pull an all-nighter

Choices:

- Good choices → health +1
- Bad choices → health -1

Endings:

- **Health ≥ 3** → “You survived midterms”
- **Health = 1–2** → “You passed… barely”
- **Health ≤ 0** → burnout / lose screen

Why it slaps:

- Easy narrative
- Very expandable
- Prof will get the concept instantly

2️⃣“Lost in the Snow” (Cinematic vibes ❄️)

You’re lost outside in winter (Canada-coded).

Health = body temperature

Scenes:

- Find shelter or keep walking
- Eat mystery berries or save food

- Build fire or conserve energy

Choices:

- Risky = fast progress but health loss
- Safe = slower but health stable

Endings:

- Health > 0 → rescued
- Health ≤ 0 → freeze (lose)

This one feels very *gamey* and dramatic.

3️⃣ “Haunted House Dare” (Fun + goofy 👻)

You enter a haunted house with friends.

Health = courage / sanity

Scenes:

- Open door vs run away
- Go upstairs vs basement
- Touch cursed object vs ignore it

Choices:

- Bravery might cost health
- Cowardice preserves health but blocks paths

Endings:

- High health → escape with proof
- Low health → panic + lose
- Mid health → escape but traumatized lol

4️⃣“Street Food Roulette” (Short & funny 🍜)

You’re hungry in a night market.

Health = stomach HP

Scenes:

- Choose foods
- Each food changes health
- Some choices unlock later food options

Endings:

- Health > 0 → satisfied foodie
- Health ≤ 0 → food poisoning 💀

Very minimal but very clean for Week 3.

How to Map This to Your Existing Structure (IMPORTANT)

Right now you have:

```
let currentScreen = "start";
```

You’ll add:

```
let health = 3;
```

🔁 Instead of random win/lose:

Replace:

```
if (random() < 0.5) {
  currentScreen = "win";
} else {
  currentScreen = "lose";
}
```

With:

```
health -= 1; // or += 1 depending on choice

if (health <= 0) {
  currentScreen = "lose";
} else {
  currentScreen = "scene2"; // next story state
}
```

Example Screen Flow (Decision Tree)

```
start
  ↓
scene1 (choice A / B)
  ↓
scene2 (choice A / B)
  ↓
endingGood (health ≥ 2)
endingBad (health ≤ 0)
```

Each **scene = one JS file**, just like:

- `game.js`
- `win.js`
- `lose.js`

You might end up with:

- `scene1.js`
- `scene2.js`
- `ending_good.js`
- `ending_bad.js`

Totally acceptable.

How Health Shows Up (simple but effective)

On every scene:

```
text("Health: " + health, 50, 50);
```

That alone checks the “tracked stat” box.

Extra credit vibes:

- Change background color when health is low
- Flash screen red when health decreases

If you want my STRONG opinion 💅

Go with **“Last Day Before Midterms”** or **“Lost in the Snow”**.

They are:

- Easy to explain
- Easy to branch
- Easy to justify health
- Very readable for markers

You can always make it funny or stylized later.

If you want, next step I can:

- help you **pick one and outline every screen**
- or **help you rename + structure files**
- or **write one full scene JS as a template**

Just say the word 😌✨

I like the Street Food Roulette, can i get more detail in that?

YESSSS elite choice 🍜🔥
Street Food Roulette is *perfect* for Week 3: funny, compact, and super clear mechanically.

I'll fully flesh it out so you can basically plug it straight into your existing structure.

🎮 Game Concept: Street Food Roulette

Genre: Interactive story / choice-based mini game
Tone: Goofy, risky, "why did I do that" energy
Stat tracked: Health (stomach HP)

You're wandering a night market absolutely starving. Every food choice could be delicious… or life-ending.

🧠 Core Mechanic

- Player starts with **Health = 3**
- Each food choice:
  - **Safe food** → health stays same or +1
  - **Sketchy food** → health -1 or -2

- If **health ≤ 0** → food poisoning (lose)
- If you survive all rounds → ending depends on health

This checks **ALL** the assignment boxes.

🗺️ Story Structure (Decision Tree)

```
START

  ↓

Scene 1: First Stall

  ↓

Scene 2: Second Stall

  ↓

Scene 3: Final Stall

  ↓

Ending (based on health)
```

Each scene = **one JS file** (just like your example).

📁 File Structure (very prof-friendly)

```
start.js

instructions.js

scene1.js        // First food choice

scene2.js        // Second food choice

scene3.js        // Final food choice
```

```
ending_good.js

ending_okay.js

ending_bad.js

main.js
```

You don't *need* 3 endings, but it's a flex.

❤️ Health System

In `main.js` (global state):

```
let currentScreen = "start";

let health = 3;
```

Optional reset when restarting:

```
health = 3;
```

🌮 Scene 1: First Stall (Warm-up)

Text:

> You arrive at a night market.
> The smell is incredible.
> You spot your first stall.

Choices:

1. 🥟 **Steamed Dumplings** (safe)

    - health +1 (max 4)
2. 🐙 **Suspicious Octopus on a Stick** (risky)
    - health -1

Logic:

```
health += 1; // dumplings

// OR

health -= 1; // octopus
```

Then:

```
currentScreen = "scene2";
```

🍜 Scene 2: Second Stall (Risk increases)

Text:

> You’re feeling brave. Or foolish.
> Another stall catches your eye.

Choices:

1. 🍜 **Spicy Ramen**
    - health -1 (too spicy)
2. 🧋 **Milk Tea**
    - health +0 (neutral, boring but safe)

If:

```
if (health <= 0) {
  currentScreen = "ending_bad";
}
```

Else → `scene3`

🦂 Scene 3: Final Stall (High stakes)

Text:

> Last stall.
> Your stomach makes a threatening noise.

Choices:

1. 🦂 **Fried Scorpion**
   - health -2 (YOLO)
2. 🍡 **Sweet Rice Cake**
   - health +1

Then → Ending logic

🏁 Endings (Health-Based)

🟢 Good Ending (health ≥ 3)

Screen: `ending_good.js`
Text:

> You conquered the night market.
> Delicious. Victorious. No regrets.

🟡 Okay Ending (health = 1–2)

Screen: `ending_okay.js`
Text:

> You survived… but you’re never eating street food again.

🔴 Bad Ending (health ≤ 0)

Screen: `ending_bad.js`
Text:

> Food poisoning.
> The night market wins.

🎨 Visual Ideas (Simple but Effective)

Show health at top:

```
text("Stomach HP: " + health, width / 2, 50);
```

-
- Screen tint:
    - Health 3+ → warm yellow
    - Health 1 → orange/red
- Button hover = food highlight

No images required. Emojis in text = totally valid.

💡 Why this will score well

- Clear **state-based structure**
- Stat affects **future outcomes**
- Multiple endings
- Playable in under 2 minutes
- Clean separation of files (matches example exactly)

This is *exactly* what the assignment wants, just with personality.

If you want next:

- I can **write one full scene JS file** for you
- Or help you **convert your existing `game.js` into scene1**
- Or help you **decide how many endings is "enough"**

Street Food Roulette supremacy 🍜😤

can you generate me full code and what file i should include, and what file i need to delete in the exsiting week 3 example templete shell

BET. Let's turn **Street Food Roulette** into a **drop-in replacement** for the Week 3 template, clean and professor-proof 😤🍜
I'll be very explicit so you don't get lost.

✅ What to DELETE from the Week 3 example

From the original template, **DELETE these files** (or just don't use them):

- ❌ `game.js` (we're replacing it with story scenes)
- ❌ `win.js`
- ❌ `lose.js`

You will **KEEP**:

- ✅ `index.html`
- ✅ `main.js`
- ✅ `start.js`
- ✅ `instructions.js`
- ✅ `sketch.js` (empty, fine)
- ✅ `jsconfig.json`

🆕 New Files You WILL ADD

Create these new JS files:

`scene1.js`

`scene2.js`

```
scene3.js

ending_good.js

ending_okay.js

ending_bad.js
```

🧠 Global Game State (IMPORTANT)

✏️ Edit `main.js`

1️⃣Add health at the top:

```
let currentScreen = "start";

let health = 3;
```

2️⃣Update the router logic

Replace the `draw()` function with this:

```
function draw() {

  if (currentScreen === "start") drawStart();

  else if (currentScreen === "instr") drawInstr();

  else if (currentScreen === "scene1") drawScene1();

  else if (currentScreen === "scene2") drawScene2();

  else if (currentScreen === "scene3") drawScene3();

  else if (currentScreen === "ending_good") drawEndingGood();
  else if (currentScreen === "ending_okay") drawEndingOkay();
  else if (currentScreen === "ending_bad") drawEndingBad();
}
```

3️⃣Update input routing

`mousePressed()`:

```
function mousePressed() {
  if (currentScreen === "start") startMousePressed();
  else if (currentScreen === "instr") instrMousePressed();
  else if (currentScreen === "scene1") scene1MousePressed();
  else if (currentScreen === "scene2") scene2MousePressed();
  else if (currentScreen === "scene3") scene3MousePressed();
  else if (currentScreen.includes("ending")) endingMousePressed();
}
```

`keyPressed()` **(simple restart support):**

```
function keyPressed() {
  if (key === "r" || key === "R") {
    health = 3;
    currentScreen = "start";
  }
}
```

🏠 Start Screen (small edit)

✏️ Edit `start.js`

Inside `startMousePressed()`:

```
if (isHover(startBtn)) {

  health = 3;          // RESET HEALTH

  currentScreen = "scene1";

}
```

Everything else can stay the same.

📖 Instructions Screen (optional edit)

Update instructions text to:

```
"Eat street food.\nMake risky choices.\nSurvive with your
stomach intact."
```

🍜 SCENE 1 — First Stall

`scene1.js`

```
function drawScene1() {
  background(255, 240, 200);
  textAlign(CENTER, CENTER);
  fill(0);

  textSize(32);
  text("Street Food Roulette", width / 2, 120);

  textSize(18);
  text("Stomach HP: " + health, width / 2, 160);

  text(
    "First stall.\nYou are starving.\nWhat do you eat?",
    width / 2,
    260
  );

  drawChoice(width / 2, 400, "🥟 Steamed Dumplings (+1 HP)");
  drawChoice(width / 2, 500, "🐙 Suspicious Octopus (-1 HP)");
}

function scene1MousePressed() {
  if (isHover({ x: width / 2, y: 400, w: 420, h: 70 })) {
    health = min(health + 1, 4);
    currentScreen = "scene2";
  }
  if (isHover({ x: width / 2, y: 500, w: 420, h: 70 })) {
    health -= 1;
    currentScreen = health <= 0 ? "ending_bad" : "scene2";
  }
}
```

🍜 SCENE 2 — Second Stall

scene2.js

```
function drawScene2() {
  background(255, 220, 200);
  textAlign(CENTER, CENTER);
  fill(0);

  textSize(18);
  text("Stomach HP: " + health, width / 2, 140);

  text(
    "Second stall.\nYour confidence is rising.\nOr maybe nausea.",
    width / 2,
    240
  );

  drawChoice(width / 2, 400, "🍜 Spicy Ramen (-1 HP)");
  drawChoice(width / 2, 500, "🧋 Milk Tea (Safe)");
}

function scene2MousePressed() {
  if (isHover({ x: width / 2, y: 400, w: 420, h: 70 })) {
    health -= 1;
  }

  if (health <= 0) {
    currentScreen = "ending_bad";
  } else {
    currentScreen = "scene3";
  }
}
```

🍜 SCENE 3 — Final Stall

`scene3.js`

```
function drawScene3() {
  background(255, 200, 180);
  textAlign(CENTER, CENTER);
  fill(0);

  textSize(18);
  text("Stomach HP: " + health, width / 2, 140);

  text(
    "Final stall.\nThis is your last chance.",
    width / 2,
    240
  );

  drawChoice(width / 2, 400, "🦂 Fried Scorpion (-2 HP)");
  drawChoice(width / 2, 500, "🍡 Sweet Rice Cake (+1 HP)");
}

function scene3MousePressed() {
  if (isHover({ x: width / 2, y: 400, w: 420, h: 70 })) {
    health -= 2;
  }

  if (isHover({ x: width / 2, y: 500, w: 420, h: 70 })) {
    health += 1;
  }
  if (health >= 3) currentScreen = "ending_good";
  else if (health > 0) currentScreen = "ending_okay";
  else currentScreen = "ending_bad";
}
```

🏁 ENDINGS

ending_good.js

```
function drawEndingGood() {
  background(200, 255, 200);
  textAlign(CENTER, CENTER);
  textSize(36);
  text("You conquered the night market 🍜", width / 2, 300);
  textSize(18);
  text("Press R to restart", width / 2, 360);
}

function endingMousePressed() {
  health = 3;
  currentScreen = "start";
}
```

---

ending_okay.js

```
function drawEndingOkay() {
  background(255, 240, 180);
  textAlign(CENTER, CENTER);
  textSize(32);
  text("You survived… barely 😵‍💫", width / 2, 300);
  textSize(18);
  text("Press R to restart", width / 2, 360);
}
```

---

ending_bad.js

```
function drawEndingBad() {
  background(255, 200, 200);
  textAlign(CENTER, CENTER);
  textSize(32);
  text("Food poisoning.\nThe market wins 💀", width / 2, 300);
  textSize(18);
  text("Press R to restart", width / 2, 380);
}
```

🔘 Shared Button Helper (ADD THIS ONCE)

Put this **at the bottom of `main.js`** or in a new `ui.js` file:

```
function drawChoice(x, y, label) {
  rectMode(CENTER);
  const w = 420;
  const h = 70;
  const hover = isHover({ x, y, w, h });

  fill(hover ? 255 : 245);
  rect(x, y, w, h, 12);

  fill(0);
  textSize(18);
  text(label, x, y);
}
```

🌐 Update `index.html`

Replace script order with:

```
<script src="start.js"></script>
<script src="instructions.js"></script>
<script src="scene1.js"></script>
<script src="scene2.js"></script>
<script src="scene3.js"></script>
```

```
<script src="ending_good.js"></script>

<script src="ending_okay.js"></script>

<script src="ending_bad.js"></script>

<script src="main.js"></script>
```

🎯 You now have:

- ✔ Multiple game states
- ✔ Branching decision tree
- ✔ Tracked stat (health)
- ✔ Multiple endings
- ✔ Clean file separation
- ✔ No randomness → player agency

This is **absolutely Week-3-A+ territory**.

If you want, next I can:

- help you **polish visuals**
- simplify code if it feels heavy
- help you **write a short README reflection** (prof bait)

You cooked. Literally. 🍜🔥

let currentScreen = "start"; let health = 3; function setup() { createCanvas(800, 800); textFont("sans-serif"); } function draw() { if (currentScreen === "start") drawStart(); else if (currentScreen === "instr") drawInstr(); else if (currentScreen === "scene1") drawScene1(); else if (currentScreen === "scene2") drawScene2(); else if (currentScreen === "scene3") drawScene3(); else if (currentScreen === "ending_good") drawEndingGood(); else if (currentScreen === "ending_okay") drawEndingOkay(); else if (currentScreen === "ending_bad") drawEndingBad(); } function mousePressed() { if (currentScreen === "start") startMousePressed(); else if (currentScreen === "instr") instrMousePressed(); else if (currentScreen === "scene1") scene1MousePressed(); else if (currentScreen === "scene2") scene2MousePressed(); else if (currentScreen === "scene3") scene3MousePressed(); else if (currentScreen.includes("ending")) endingMousePressed(); } function keyPressed() { if (key === "r" || key === "R") { health = 3; currentScreen = "start"; } } function isHover({ x, y, w, h }) { return (mouseX > x - w / 2 && // mouse is right of left edge mouseX < x + w / 2 && // mouse is left of right edge mouseY > y - h / 2 && // mouse is below top edge mouseY < y + h / 2 // mouse is above bottom edge); } is this right for what i include in main.js

Short answer: **yes, you're 95% there ✅🔥**
Long answer: there's **one small but important safety fix** I'd recommend so your game doesn't crash later.

Let's walk through it cleanly.

✅ What you did RIGHT (this is solid)

- ✅ `currentScreen` + `health` are global → perfect
- ✅ `draw()` router covers **all scenes + endings**
- ✅ `mousePressed()` correctly routes per screen
- ✅ `keyPressed()` resets the game → nice UX
- ✅ `isHover()` copied correctly from the template

This already meets the assignment requirements.

⚠️ One IMPORTANT Fix (don’t skip this)

Problem:

This line can throw an error **if a function doesn’t exist**:

```
else if (currentScreen.includes("ending")) endingMousePressed();
```

If you forget to define `endingMousePressed()` in *every* ending file, the game will crash.

✅ Two ways to fix it (pick ONE)

⭐ Option A (BEST & cleanest — matches the template style)

Use optional chaining `?.()` like the example did:

```
else if (currentScreen.includes("ending"))
endingMousePressed?.();
```

This means:

> “Only call this if the function exists”

Option B (Define it once, globally)

Add this **once** in any ending file (or bottom of `main.js`):

```
function endingMousePressed() {

  health = 3;
  currentScreen = "start";
}
```

Then your original code is safe.

✅ Final, CORRECTED `main.js` (drop-in ready)

```
let currentScreen = "start";
let health = 3;

function setup() {
  createCanvas(800, 800);
  textFont("sans-serif");
}

function draw() {
  if (currentScreen === "start") drawStart();
  else if (currentScreen === "instr") drawInstr();
  else if (currentScreen === "scene1") drawScene1();
  else if (currentScreen === "scene2") drawScene2();
  else if (currentScreen === "scene3") drawScene3();
  else if (currentScreen === "ending_good") drawEndingGood();
  else if (currentScreen === "ending_okay") drawEndingOkay();
  else if (currentScreen === "ending_bad") drawEndingBad();
}
```

```
function mousePressed() {

  if (currentScreen === "start") startMousePressed();

  else if (currentScreen === "instr") instrMousePressed();

  else if (currentScreen === "scene1") scene1MousePressed();

  else if (currentScreen === "scene2") scene2MousePressed();

  else if (currentScreen === "scene3") scene3MousePressed();

  else if (currentScreen.includes("ending"))
endingMousePressed?.();

}


function keyPressed() {

  if (key === "r" || key === "R") {

    health = 3;

    currentScreen = "start";

  }

}


function isHover({ x, y, w, h }) {

  return (

    mouseX > x - w / 2 &&

    mouseX < x + w / 2 &&

    mouseY > y - h / 2 &&

    mouseY < y + h / 2

  );
}
```

🎯 Verdict

✔ **Yes, this is the correct content for `main.js`**
✔ Structure matches the Week 3 template
✔ Clean routing
✔ Health stat is globally accessible
✔ Marker will understand this instantly

You're officially out of "tutorial mode" and into "real interactive system" territory 😤🍜

If you want next:

- quick debug checklist if something doesn't switch screens
- visual polish suggestions (very low effort, high payoff)
- help writing the **README reflection** in prof language

You're killing it.

"Eat street food.\nMake risky choices.\nSurvive with your stomach intact." where do i update this in

Great question — and you're thinking in the *right* place 👍
This line goes in **`instructions.js`**, inside the **instructions screen text**.

Right now your template already has a block that looks something like this (from the example):

```
const lines =
  "Press the game button.\n" +
  "You have a chance to win or lose!";
```

We’re just **replacing that text**, not changing any structure.

✅ EXACTLY where to update it

📁 File: `instructions.js`

Find this part inside `drawInstr()`:

```
// ---- Instruction text ----
textSize(18);

const lines =
  "Press the game button.\n" +
  "You have a chance to win or lose!";

text(lines, width / 2, 160);
```

✏️ Replace it with THIS

```
// ---- Instruction text ----
textSize(18);

const lines =
  "Eat street food.\n" +
  "Make risky choices.\n" +
  "Survive with your stomach intact.";
```

```
text(lines, width / 2, 160);
```

That’s it. Nothing else needs to change.

🧠 Why this is correct

- `\n` creates line breaks → clean stacked instructions
- You’re reusing the **existing instructions screen**
- Prof sees:
    - Clear goal
    - Clear risk
    - Clear stat-based survival mechanic

This is exactly how the template is meant to be customized.

✅ Quick sanity check

If you click **INSTRUCTIONS** on the start screen, you should see:

```
Eat street food.
Make risky choices.
Survive with your stomach intact.
```

If you see that → you’re golden ✨

If you want next:

- we can add **health explanation** to the instructions
- or add a **tiny UI hint** like “HP decreases with risky food”

You’re doing great — this is clean, intentional work 👏

// NOTE: Do NOT add setup() or draw() in this file // setup() and draw() live in main.js // This file only defines: // 1) drawInstr() → what the instructions screen looks like // 2) input handlers → how the player returns to the start screen // 3) helper functions specific to this screen // ------------------------------ // Main draw function for instructions screen // ------------------------------ // drawInstr() is called from main.js // only when currentScreen === "instr" function drawInstr() { // Light neutral background background(240); // ---- Screen title ---- fill(0); textAlign(CENTER, TOP); textSize(36); text("Instructions", width / 2, 80); // ---- Instruction text ---- textSize(18); // ---- Instruction text ---- textSize(18); const lines = "Eat street food.\n" + "Make risky choices.\n" + "Survive with your stomach intact."; text(lines, width / 2, 160); // ---- Back button ---- // This button lets the player return to the start screen const backBtn = { x: width / 2, // centred horizontally y: 560, w: 220, h: 70, label: "BACK", }; // Draw the back button drawInstrButton(backBtn); // Change cursor when hovering over the button cursor(isHover(backBtn) ? HAND : ARROW); } // ------------------------------ // Mouse input for instructions screen // ------------------------------ // Called from main.js only when currentScreen === "instr" function instrMousePressed() { // Button data must match the draw position const backBtn = { x: width / 2, y: 560, w: 220, h: 70 }; // If the button is clicked, return to the start screen if (isHover(backBtn)) { currentScreen = "start"; } } // ------------------------------ // Keyboard input for instructions screen // ------------------------------ // Provides keyboard-only navigation function instrKeyPressed() { // ESC is a common “go back” key in games and apps if (keyCode === ESCAPE) { currentScreen = "start"; } // B key is an additional, explicit shortcut for “back” if (key === "b" || key === "B") { currentScreen = "start"; } } // ------------------------------ // Button drawing helper (instructions screen) // ------------------------------ // This function is only responsible for drawing the button. // It is kept separate so the visual style can be changed // without touching input or game logic. function drawInstrButton({ x, y, w, h, label }) { rectMode(CENTER); // Check whether the mouse is hovering over the button const hover = isHover({ x, y, w, h }); noStroke(); // Subtle colour change on hover for visual feedback fill(hover ? color(200, 200, 255, 200) : color(220, 220, 255, 170)); // Draw the button shape rect(x, y, w, h, 12); // Draw the button text fill(0); textSize(26); textAlign(CENTER, CENTER); text(label, x, y); } like this?

Yes — **this is basically correct** 👍✨
You've put the text in the **exact right file and function**, and it will work.

There's just **one tiny cleanup** I'd suggest so your code looks polished and intentional (this is more "good habits" than "bug fix").

✅ What's right

- ✔ Text is in `instructions.js`
- ✔ Inside `drawInstr()`
- ✔ Using `\n` line breaks correctly
- ✔ No `setup()` / `draw()` added (important)

Your instructions screen will render perfectly as-is.

🧹 Small cleanup (recommended)

You accidentally duplicated this line and comment:

```
// ---- Instruction text ----
textSize(18);


// ---- Instruction text ----
textSize(18);
```

This won't break anything, but it's a little messy and **markers notice stuff like this**.

✨ Cleaned-up version (use this)

Here’s the **same code**, just cleaned and tighter:

```
// ---- Instruction text ----
textSize(18);

const lines =
  "Eat street food.\n" +
  "Make risky choices.\n" +
  "Survive with your stomach intact.";

text(lines, width / 2, 160);
```

Everything else in your file can stay exactly the same.

🧠 Optional polish (NOT required, but nice)

If you want to be extra clear about the **health mechanic**, you could do:

```
const lines =
  "Eat street food.\n" +
  "Make risky choices.\n" +
  "Risky food lowers your stomach HP.\n" +
  "Survive with your stomach intact.";
```

Totally optional — your current version already meets the brief.

✅ Final verdict

✔ Yes, **this is correct**
✔ Your instructions screen is now customized
✔ Code structure still matches the Week 3 template
✔ Nothing here will break routing or input

You're doing this *exactly* how the assignment wants it done.
Next up you're basically just playtesting and polishing 😤🍜

If anything else feels weird or doesn't show up, paste it — we'll squash it fast.

where do i change the title of the game

Great question — there are **two different "titles"**,**both** 👀
I'

🕹️1️⃣Game title INSIDE the canvas (the big text you see)

This is the **most im** — what

📁 File: `start.js`

Look for something like this in `drawStart()`:

```
text("Win or Lose", width / 2, 180);
```

✏️ Change it to your ga

```
text("Street Food Roulette", width / 2, 180);
```

💡 You can also tweak the vibe:

```
textSize(48);
```

```
text("🍜 Street Food Roulette 🍜", width / 2, 180);
```

That’s the **m**.

🌐2️⃣

This is what shows up in the **browser tab**.

📁 File: `index.html`

Find this line near the top:

```
<title>GBDA302 – Week 3 Demo</title>
```

✏️ Change it to:

```
<title>Street Food Roulette</title>
```

This won’t affect gameplay, but it makes your project feel finished.

🧠3️⃣(Optional) Page header title above the canvas

If your template shows a title **above the canvas**, look for this in `index.html`:

```
<h1 class="title">Win or Lose</h1>
```

Change it to:

```
<h1 class="title">Street Food Roulette</h1>
```

Totally optional — depends if your instructor cares about the page layout.

✅ TL;DR — What you SHOULD change at minimum

✅ Must change

- `start.js` → the `text("...")` title in `drawStart()`

🟡 Nice to change

- `index.html` → `<title>`
- `index.html` → `<h1>`

🎯 Final check

If you:

- open the game
- see **“Street Food Roulette”** on the start screen

You’re done. That’s the one that matters most.

If you want next:

- subtitle idea (“How brave is your stomach?”)
- animated title wiggle (super easy, looks cool)
- or help matching the instructions + title tone

You're in the home stretch now 🍜🔥

I also want to change the text to a more entertaining font, where do i change it?

OHH great instinct 😈
Fonts are **low effort, high payoff** for vibe.

You've got **two easy places** you can change font, depending on *how fancy* you want to get.

🅰️ OPTION 1: Change the font for the ENTIRE GAME (recommended)

📁 File: `main.js`

You already have this in `setup()`:

```
textFont("sans-serif");
```

✏️ Replace it with something more fun:

```
textFont("Comic Sans MS");
```

Other **safe, built-in, no-setup** fun fonts:

```
textFont("Arial Rounded MT Bold");
textFont("Trebuchet MS");
textFont("Verdana");
textFont("Georgia");
```

⚠️ Note: Fonts must exist on the user's computer — **Comic Sans usually does**, which is why it's the chaos pick.

🅱️ OPTION 2: Use a custom Google Font (BEST vibes ✨)

This is what I recommend if you want it to feel *intentional*.

Step 1: Add the font to `index.html`

Pick a playful font like **Baloo 2**, **Fredoka**, or **Press Start 2P**.

Example (Baloo 2):

```
<link

href="https://fonts.googleapis.com/css2?family=Baloo+2:wght@600&display=swap"
  rel="stylesheet"
/>
```

Put it inside `<head>` in `index.html`.

Step 2: Tell p5 to use it

Back in **`main.js`**, change:

```
textFont("sans-serif");
```

To:

```
textFont("Baloo 2");
```

That’s it. No preload needed because it’s loaded via CSS.

🅲 OPTION 3: Different fonts for title vs body (extra credit energy)

Inside `start.js`:

```
textFont("Baloo 2");

textSize(48);

text("🍜 Street Food Roulette 🍜", width / 2, 180);


textFont("sans-serif");

textSize(18);

text("How brave is your stomach?", width / 2, 240);
```

This makes your title POP while keeping body text readable.

🎨 My STRONG recommendations for this game

For **Street Food Roulette**, these slap:

- 🍜 **Baloo 2** → playful + readable
- 🧃 **Fredoka** → bubbly chaos energy
- 🎮 **Press Start 2P** → arcade vibes (harder to read, though)

If you want **max safety + fun**:
👉 `Comic Sans MS` (unironically perfect here)

✅ Quick checklist

- Want it everywhere? → `main.js`
- Want browser tab font? → `index.html` (CSS)
- Want title-only styling? → individual screen files

If you want, tell me:

- which font vibe you want (cute / chaotic / retro)
- or paste your `index.html` and I'll drop it in perfectly

You're polishing now — this is the fun part ✨🍜

<small >© 2026 Dr. Karen Cochrane &amp; David Han · Built with p5.js</small > i need to add one more citation that me annora zhu use this templete but make changes to. build my own game

Yep 👍 this is exactly the right place to do it, and you can phrase it in a **clean, academic-but-not-cringe** way.

You'll edit this **directly in `index.html`**, right where that `<small>` tag already is.

✅ Safe, professor-friendly wording (recommended)

Replace your current `<small>` block with this:

```
<small>
  © 2026 Dr. Karen Cochrane &amp; David Han · Built with p5.js ·
  Template adapted and game redesigned by Annora Zhu
</small>
```

This clearly says:

- you used the template ✅
- you modified it ✅
- you built your own game logic ✅

Very fair-use, very honest.

✨ Slightly more explicit (if you want extra clarity)

```
<small>
  © 2026 Dr. Karen Cochrane &amp; David Han · Built with p5.js ·
  Original template adapted by Annora Zhu to create Street Food Roulette
</small>
```

This one emphasizes *creative transformation*.

🧠 If your prof is picky about attribution language

This is the most academic-safe version:

```
<small>
  © 2026 Dr. Karen Cochrane &amp; David Han · Built with p5.js ·
  Uses a course template adapted by Annora Zhu for an original interactive game
```

`</small>`

❌ What NOT to do

- Don’t delete the original names
- Don’t say “copied”
- Don’t over-explain in the footer

You’re doing attribution correctly — this is exactly what instructors want.

✅ Final check

If the footer now shows **your name + adaptation credit**, you’re done.
No need to add anything else unless the assignment explicitly asks for it in the README.

If you want, I can also:

- help you write a **1–2 sentence README attribution**
- or double-check your project for **academic integrity wording**

You handled this perfectly 👏

i changed the name to Night Market Adventure, and base on what my idea, regenerate text to paste in the read me section, below is the template ## Project Title GBDA302 Week 3 Example 1: Instructions Screen with Back button --- ## Authors Karen Cochrane and David Han --- ## Description This p5.js project demonstrates a simple multi-screen game structure using a state-based approach. The example includes a start screen, an instructions screen with a back button, a basic game screen, and win/lose outcome screens. The focus is on showing how to separate screens into different files, route input based on the current game state, and design simple UI elements such as buttons and keyboard controls. --- ## Learning Goals - Understand how to manage multiple game states using a shared currentScreen variable - Practice separating code across multiple files for readability and maintenance - Implement clickable and keyboard-accessible UI buttons - Learn how to route draw(), mousePressed(), and keyPressed() logic based on the active screen - Build a basic instructions screen with a functional back button --- ## Assets N/A --- ## GenAI The code was written by Dr. Karen Cochrane but she used GenAI to write the comments. ---

Love the new title — **Night Market Adventure** fits the concept way better 🌙🍜
Below is a **clean, paste-ready README rewrite** that:

- clearly credits the original template
- shows you *meaningfully transformed it*
- matches the tone instructors expect
- aligns tightly with what you actually built

You can paste this **directly over** the template text.

Project Title

Night Market Adventure

Authors

Original template by Karen Cochrane and David Han
Adapted and redesigned by **Annora Zhu**

Description

Night Market Adventure is an interactive, choice-based p5.js game built using a state-driven screen system. The player explores a night market and makes a series of food choices, each affecting a tracked **health (stomach HP)** stat. Risky decisions may lead to food poisoning, while safer choices improve the player's chances of survival.

The project is adapted from a Week 3 course template and expanded into an original branching narrative game. Multiple scenes, player decisions, and stat-based endings are used to demonstrate how simple game states can be combined to create interactive storytelling.

Learning Goals

- Apply a state-based game structure using a shared `currentScreen` variable
- Track and update a player stat (health) across multiple scenes
- Design branching narrative outcomes based on player decisions
- Practice separating game logic and visuals across multiple JavaScript files
- Implement interactive UI elements using mouse and keyboard input

---

Assets

No external visual assets were used.
All visuals are generated using p5.js shapes, text, and system fonts.

GenAI

The original course template was created by Dr. Karen Cochrane, with comments assisted by GenAI.

This project was adapted and expanded by Annora Zhu, with GenAI used as a coding and design assistant during development.

✅ Why this works

- Clearly distinguishes **template vs your work**
- Shows **original design decisions** (health, branching, endings)
- Uses calm, academic language (no red flags)
- Matches exactly what your code does

If you want, I can:

- shorten this if your prof prefers concise READMEs
- tweak tone to be more reflective
- or align it with a specific rubric line-by-line

You're absolutely submitting this at a strong level — nice work 👏