

Entry Header

Name: Alexandra Dillon

Goal of Work Session

Brainstorming ideas for SideQuest 3, coding, and debugging.

GenAI Documentation

Everyone must complete this section. If not GenAI was used, write, “No GenAI used for this task.”

When GenAI is not used, process evidence should

still demonstrate iteration, revision, or development over time.

Because GenAI can closely mimic human-created work, instructors or TAs may occasionally request additional process evidence to confirm non-use.

This may include original working files (e.g., an illustrator file), intermediate drafts, or a brief check-in with a TA to walk through your process.

These requests are not an assumption of misconduct. They are part of ensuring academic integrity in an environment where distinguishing between human-created and AI-generated work is increasingly difficult.

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

Date Used

: Mon Feb 2nd

Tool Disclosure

: ChatGPT 5.2

Purpose of Use

: Brainstorming, Clarification/questions, and Debugging

Summary of Interaction

: GenAI gave me a couple of ideas to choose from based on the instruction prompt I provided and helped clarify some questions I had on integrating the

different files into index.html, as well as how to use \n. It also helped me debug my code when I ran into problems.

Human Decision Point(s)

: I only chose one of the ideas it brainstormed for me, though I modified it to include the stress point factor. I mostly followed its instructions when it came to debugging the code because I didn't know where to start, but it did take a few tries to get a good answer; I ended up having to give it all the code in the files I was using.

Integrity & Verification Note

: I double checked that the idea I planned to use fit the sidequest instructions, and I kind of knew that the debugging had worked after running the code successfully due to the changes.

Scope of GenAI Use

: I coded the assignment, came up with the text and mapped out the “decision tree.”

Limitations or Misfires

: As noted previously, the tool needed all my code to finally be able to help debug it properly.

Summary of Process (Human + Tool)

I started by asking ChatGPT for ideas if I wanted to create a simple click-and-choice-based game. When it gave me a couple of ideas, I chose the Coffee Shop Simulator and mapped out what decisions and consequences I wanted to add. I also added the stress point calculator for bonus marks. After all the brainstorming was done, I started coding it based on the comments and previous code in VS Code. When I went and ran the code, I noticed it was just pulling up the title of the game and a gray screen. After trying to figure it out myself, I went to ChatGPT and asked it to help me debug it. I had to keep modifying what I was asking until I just decided to give it all the code I had, which is when it was finally able to point out what was wrong with the code. After fixing and running it, I also noticed I made some spelling mistakes, and that I didn't really like the background colour I chose, so I changed that.

Appendix

ME: im in a coding class with a small assignment due where I have to build an interactive story that unfolds through multiple game states and files, branching like a small decision tree. can you give me some simple ideas to get started

Yes — this kind of assignment is perfect for keeping things **simple but clever**. You want a story that naturally supports **choices + multiple outcomes**, without needing complex mechanics.

Here are some easy, beginner-friendly ideas that work really well for branching game states:

1. “Lost on Campus”

You’re lost trying to find a classroom before an exam.

Branches:

- Ask a stranger for help
- Use Google Maps
- Wander randomly

Each choice leads to different:

- people
- locations
- outcomes (late / on time / wrong building)

Very easy to implement:

- each “location” = one game state
 - buttons = decisions
-

2. “Choose Your Internship”

You get multiple internship offers.

Branches:

- Corporate job
- Startup
- Research role

Then sub-branches:

- boss personality
- workload
- culture

Ends with different career paths.

3. “Coffee Shop Simulator”

You’re working a shift at a café.

Choices:

- How to handle rude customer
- What drink to recommend
- Whether to take a break

Outcomes:

- tips

- stress
 - fired / promoted / praised
-

4. “Group Project Chaos” (very relatable)

You’re in a group project.

Branches:

- Take leadership
- Stay quiet
- Confront teammate

Each leads to:

- different group dynamics
- grade outcomes
- stress levels

Super easy to write and realistic.

5. “Escape the Room” (classic)

You’re trapped in a room.

States:

- Desk
- Door

- Window
- Closet

Each gives choices:

- inspect
- use item
- combine items

Ends:

- escape
 - stuck forever
 - secret ending
-

How to keep it *coding-simple*

Design it like this:

- 1 state = 1 screen
- Each screen has:
 - text
 - 2–3 buttons
- Buttons just change `gameState`

Example structure:

```
if (state === "start") {  
    text = "You're lost on campus."  
    buttonA = "Ask for help"  
    buttonB = "Use map"  
}
```

No inventory.
No animations.
No scoring.
Just **story + decisions**.

Pro tip (for marks)

Markers love when:

- choices feel meaningful
- endings are different
- tone is consistent (funny, dramatic, cozy, etc.)

They do *not* care about:

- fancy graphics
- long code
- complex logic

A clean, well-written **3–5 minute story** usually scores higher than a messy “big idea”.

Easiest high-grade combo:

Group Project Chaos or Lost on Campus

- relatable
- minimal assets
- natural branching
- easy to write good dialogue

Perfect for small interactive storytelling assignments.

ME: how would i get started with the coffee shop simulator especially since i dont have any visuals

This is actually *perfect* — no visuals makes the coffee shop simulator **even easier**, because you can lean fully into **text-based storytelling**, which is exactly what your assignment is about.

Think of it like a mini *interactive fiction game*.

Step 1: Define your core states (on paper first)

You only need **5–8 states total** for a solid project.

Step 2: Write the story in plain text first

Literally in a doc:

Start screen:

It's your first day working at a coffee shop. The manager tells you to "just vibe and do your best."

Choices:

- Take it seriously
 - Try to be chill
-

Rude customer screen:

A customer complains their latte “tastes wrong.”

Choices:

- Apologize and remake
 - Argue
 - Ignore
-

Do this for all states before coding.

This saves you *hours*.

Step 3: Map it like a decision tree

Example:

```
start
  — serious → calmMorning
    |      — rushHour → managerHappy → promoted
  — chill → rudeCustomer
    — apologize → goodDay
    — argue → fired
```

If you can draw it, you can code it.

Step 4: Implement the bare minimum UI

All you need:

- One text box
- Two or three buttons

That's it.

No images.
No sound.
No animations.

Example pseudo-structure:

```
if (state === "rudeCustomer") {  
    text = "A customer says their drink is wrong."  
    button1 = "Apologize"  
    button2 = "Argue"  
}
```

Step 5: Make it fun with writing, not tech

Since you don't have visuals, use:

- tone
- humor
- relatable dialogue

Example:

The line is getting longer. Someone is ordering a “half-caf oat milk extra foam no foam” latte.

That alone carries the experience.

Easy Endings (you only need 3–4)

- **Promoted** – manager impressed
- **Fired** – too many bad choices
- **Burnt out** – survived but exhausted

- **Great day** – tips + compliments
-

Why this works so well for your class

It:

- clearly uses **multiple game states**
- has **branching paths**
- is **fully interactive**
- doesn't depend on assets
- is easy to debug

And most importantly:

the *logic* is simple, so you can focus on **story + decisions**, which is exactly what profs want.

Honestly?

A clean text-based coffee shop story with 6–8 states will usually get a **higher mark** than a half-broken visual game.

It shows:

- planning
- narrative structure
- interaction design
- technical control

Without unnecessary complexity.

ME: they want files: <!-- your existing files (order matters) --> <script src="start.js"></script> <script src="instructions.js"></script> <script src="game.js"></script> <script src="win.js"></script> <script src="lose.js"></script> <script src="[main.js](#)"></script> - can you explain this more to me (easy)

Perfect — this actually makes your life easier. You'll just **split the same game into multiple files**, where each file handles one “screen/state”.

ME: what do i put in the index.html file then

Good question — this is actually the only part people usually mess up.

Your [index.html](#) can be **very minimal**. It just needs to load p5 and your files in the right order.

Here's a clean, safe version you can copy:

```
<!DOCTYPE html>

<html>

  <head>

    <meta charset="utf-8">

    <title>Coffee Shop Simulator</title>

  </head>

  <body>

    <!-- p5 library -->

    <script
src="https://cdn.jsdelivr.net/npm/p5@1.9.0/lib/p5.js"></script>

    <!-- your existing files (order matters) -->

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

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

```
<script src="game.js"></script>  
<script src="win.js"></script>  
<script src="lose.js"></script>  
<script src="main.js"></script>  
  
</body>  
</html>
```

That's it. No canvas tag, no divs — p5 creates the canvas for you in `setup()`.

Why this works

- p5 must be loaded **first**
 - your screen files come next
 - `main.js` last because it uses everything
-

Super common mistake (don't do this)

Don't put `main.js` at the top.

If you do, you'll get errors like "startScreen is not defined".

ME: it says theres an error in this file: function winScreen() { text("You apologized and calmed the customer!\n\nYou survived your first shift!\nStress" + stress = "\nClick to restart.", width / 2, height / 2); } function handleClicks() { if (state === "win") { stress = 0; state = "start"; } }

Yep — there *is* an error, and it's this line:

```
text("You apologized and calmed the customer!\n\nYou survived your first shift!\nStress" + stress = "\nClick to restart.", width / 2, height / 2);
```

You accidentally used `=` inside a string concatenation. That turns it into an assignment, which breaks the whole line.

The fix (correct version)

```
function winScreen() {  
    text(  
        "You apologized and calmed the customer!\n\n" +  
        "You survived your first shift!\n" +  
        "Stress: " + stress + "\n\n" +  
        "Click to restart.",  
        width / 2,  
        height / 2  
    );  
}
```

Why it broke

This part is invalid:

```
stress = "\nClick to restart."
```

You were:

- assigning a string to `stress`
- inside a `text()` call
- which JavaScript does not allow in that context

You meant to **concatenate**, not assign.

Extra small improvement (optional but nice)

Before drawing text, make sure alignment is set (usually in `main.js`):

```
textAlign(CENTER, CENTER);  
textSize(20);
```

Otherwise your text might appear in a weird spot.

This is a super common beginner bug btw — you were 99% right, just one character off.

ME: its not working - i followed what you said to put in the individual files. this is whats in my index.html file:

```
<!DOCTYPE html> <html lang="en"> <head> <meta charset="utf-8" /> <meta name="viewport" content="width=device-width, initial-scale=1" /> <meta name="theme-color" content="#1e3a3a" /> <title>GBDA302 — Week 3 Demo</title> <link rel="preconnect" href="https://cdn.jsdelivr.net" /> <link rel="stylesheet" href="style.css" /> </head> <body> <header class="site-header"> <div class="wrap"> <h1 class="title">Coffee Shop Simulator</h1> <p class="subtitle">GBDA302 • Week 3 — Game States, UI, Menus</p> </div> </header> </p> </section> </main> <script src="https://cdn.jsdelivr.net/npm/p5@1/lib/p5.min.js"></script> <!-- your existing files (order matters) --> <script src="start.js"></script> <script src="instructions.js"></script> <script src="game.js"></script> <script src="win.js"></script> <script src="lose.js"></script> <script src="main.js"></script> </body> </html>
```

Okay, this helps a lot — the issue is **not p5**, it's your HTML structure.

Your `<body>` has **broken / unclosed tags**:

```
</p>  
</section>  
</main>
```

...but there is **no opening `<main>` or `<section>`** in the file you showed.

That makes the browser DOM invalid and p5 can fail silently.

The real fix (keep it simple)

For this assignment, you do **not need any layout HTML at all**.
p5 draws everything.

Use this **minimal version** so nothing interferes:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Coffee Shop Simulator</title>
  </head>
  <body>

    <!-- p5 -->
    <script
src="https://cdn.jsdelivr.net/npm/p5@1/lib/p5.min.js"></script>

    <!-- game files (order matters) -->
    <script src="start.js"></script>
    <script src="instructions.js"></script>
    <script src="game.js"></script>
    <script src="win.js"></script>
    <script src="lose.js"></script>
    <script src="main.js"></script>

  </body>
</html>
```

Delete:

- <header>
- <main>
- <section>
- <link rel="stylesheet">

They're not helping and they *can* break things.

Second thing to check (99% likely your real bug)

In your files you mixed names:

- In comments: `currentScreen`
- In code: `state`
- In `win.js`: `winScreen()`
- In `main.js`: probably calling `drawWin()`

These **must match exactly**.

Pick ONE system (use this everywhere):

```
let state = "start";
```

And in `main.js`:

```
function draw() {
    background(220);

    if (state === "start") startScreen();
    else if (state === "instructions") instructionsScreen();
    else if (state === "game") gameScreen();
    else if (state === "win") winScreen();
    else if (state === "lose") loseScreen();
}
```

If even one file uses:

- `currentScreen`
- `drawWin()` instead of `winScreen()`

→ the whole game looks “broken”.

How to instantly debug

Open the browser console (right click → Inspect → Console).

If you see:

- `winScreen is not defined`
- `state is not defined`

That tells you **exactly which file is mismatched**.

TL;DR

Your problem is:

1. Broken HTML tags
2. Inconsistent function/state names

Not p5. Not your logic. Just wiring.

ME: Its still not working. this is what i have for main.js: let state = "start"; let stress = 0; function setup() { createCanvas(600, 400); textAlign(CENTER, CENTER); textSize(16); } function draw() { background(240); if (state === "start") startScreen(); if (state === "instructions") instructionsScreen(); if (state === "game") gameScreen(); if (state === "win") winScreen(); if (state === "lose") loseScreen(); } function mousePressed() { handleClicks(); } - this is what i have for instructions.js: function instructionsScreen() { Text("Make choices to survive your first shift!\nClick to continue.", width / 2, height / 2); } function handleClicks() { if (state === "instructions") { state = "game"; } } - this is what i have for game.js: function gameScreen() { Text("A customer complains about their order.\n\nClick LEFT to apologize\nClick RIGHT to argue", width / 2, height / 2); } function handleClicks() { if (state === "game") { if (mouseX < width / 2) { state = "win"; } else { stress += 2; state = "lose"; } } } - this is what i have for win.js: function winScreen() { text("You apologized and calmed the customer!\n\nYou survived your first shift!\nStress" + stress + "\nClick to restart.", width / 2, height / 2); } function handleClicks() { if (state === "win") { stress = 0; state = "start"; } } - this is what i have for lose.js: function loseScreen() { Text("The manager is not impressed.\nYou're fired.\nStress: " + stress + "\nClick to restart.", width / 2, height / 2); } function handleClicks() { if (state === "lose") { stress = 0; state = "start"; } }

Perfect — this shows the exact problem. You actually have **two separate bugs**, both very common:

1. **Text()** is wrong (capital T)

In p5 it must be:

```
text()
```

You used:

```
Text()
```

JavaScript is case-sensitive, so those lines are literally calling a function that doesn't exist → nothing draws.

Fix in three files:

- `instructions.js`
- `game.js`
- `lose.js`

Change **every** `Text()` to `text()`.

2. You redefined `handleClicks()` in every file

This is the bigger logic bug.

JavaScript does **not** merge functions.

Each new one **overwrites the previous one**.

So only the **last loaded file (`lose.js`)** is active.

Meaning:

- clicks only work on the lose screen
 - everything else is dead
-

The correct pattern (one click handler)

You want **one** `handleClicks()` in `main.js`:

`main.js (fix)`

```
function mousePressed() {
  handleClicks();
}

function handleClicks() {
  if (state === "start") state = "instructions";

  else if (state === "instructions") state = "game";

  else if (state === "game") {
    if (mouseX < width / 2) {
      state = "win";
    } else {
      stress += 2;
      state = "lose";
    }
  }

  else if (state === "win" || state === "lose") {
    stress = 0;
    state = "start";
  }
}
```

Then **DELETE** `handleClicks()` from:

- instructions.js
- game.js
- win.js
- lose.js

Those files should **only draw**.

Final file roles (this is what your prof wants)

File	Purpose
main.js	logic + state machine
start.js	draw start screen
instructions.js	draw instructions
game.js	draw game
win.js	draw win
lose.js	draw lose

Only **one brain**, many screens.

Why your game looked "broken"

You had:

- invisible text (Text vs text)
- 5 different click handlers fighting each other

So p5 was running, but doing basically nothing.