

Side quest: Week 3

Name: Min Htet Naing, dmnaing (21008098)

Project/Assignment Decisions

Description

For Side Quest 3, I created a simple Solo Levelling-inspired training story where the player chooses an exercise each day (push-ups, running, or resting). These choices branch the story into different scenes using a multi-file, multi-state decision tree. My goal was to meet the Week 3 requirement by keeping the experience clearly state-driven while still feeling like a short interactive story.

Key Decision / Change

I decided to track health and level across states, so the player must balance training and recovery instead of always selecting the same option. Endings are unlocked based on the player's stats and thresholds, which makes the choices meaningful over multiple days. I also kept visuals lightweight (buttons, HUD, small effects) to avoid heavy assets and keep the focus on clean branching logic across multiple files.

GenAI Documentation (Used)

Date Used: February 2, 2026

Tool Disclosure: ChatGPT (GPT-5.2 Thinking)

Purpose of Use:

ChatGPT was used to help structure the project as a clean state machine across multiple files and to draft a branching decision tree that meets the Week 3 requirements (multiple game states + bonus stat tracking + multiple endings). It also helped with troubleshooting a

blank-screen bug and suggested small UI improvements (HUD bars, simple effects) while keeping the project scope manageable.

Appendix

Prompt 1: “I have uploaded my CSS file, and please improve my UI/CSS so the project looks cleaner (centred layout, readable text, and consistent button style) but keep it lightweight.”

This CSS is already solid (clean dark gradient + nice canvas styling). If you want it to look **a bit cleaner + more “finished”** while staying lightweight, here are **3 small upgrades** you can apply (you can copy/paste the full updated version below).

What to improve (quick)

1. Add **padding** around the canvas so it doesn’t feel “stuck” to the edges on small screens.
2. Make canvas **responsive** (`max-width: 100%`, `height: auto`) so it doesn’t overflow on laptop screens.
3. Optional: add a simple **container card** so the canvas feels framed (still minimal).

If you want to use the `.canvas-wrap` container

In your `index.html`, wrap your canvas container like this (super minimal):

html

 Copy code

```
<body>
  <div class="canvas-wrap"></div>
</body>
```

And in p5, move the canvas into it:

js

 Copy code

```
let c = createCanvas(900, 540);
c.parent("canvas-wrap");
```

Prompt 2: “Show me step-by-step what to add/replace and how to make new files using my existing multi-file project template so I can build a branching state-based story because I want to learn it step-by-step, which file is for which part of the design code.”

To build Week 3 using your existing multi-file template, the safest approach is: **one shared game state object, one router in `draw()`, and one file per screen**. Here's the exact step-by-step that matches your setup (`start/instructions/hub/scenes/boss/win/lose + UI helpers`).

- 1. Keep shared variables in one place (`scenes.js`)**
Put your global `game` object and `STATE` constants here (health, level, day, maxDays, current state). This makes stats persist across all screens.
- 2. Create/keep one “router” (`main.js`)**
In `draw()`, call only ONE screen function based on `game.state`, like:
 - `drawStartScreen()`
 - `drawInstructionsScreen()`
 - `drawHubScreen()`
 - `drawPushupsScreen() / drawRunScreen() / drawRestScreen()`
 - `drawBossScreen()`
 - `drawWinScreen() / drawLoseScreen()`
- 3. Split screens into files (what to add)**
Use your existing file pattern:
 - `start.js` → `drawStartScreen()`
 - `instructions.js` → `drawInstructionsScreen()`
 - `game.js` → hub + exercise scenes + boss gate ↓
`drawHubScreen()`, `drawPushupsScreen()`,
`drawRunScreen()`, `drawRestScreen()`, `drawBossScreen()`
- 4. Replace “loose drawing code” with functions**
Anything that draws (text, shapes, push/pop, buttons) must live inside a `draw____Screen()` function. If drawing code sits outside a function, it can run at load time and crash the sketch.
- 5. Use one choice handler (what to add/replace in logic)**
Make a single function like `applyChoice("pushups" | "run" | "rest")` that updates:
 - `game.health`
 - `game.level`
 - `game.day` (or sets next state)Then switch state to the correct scene.
- 6. Add a “next step” function to control branching**
After each scene, call something like `nextDayOrBoss()` that decides:
 - if `game.day < game.maxDays` → go back to `STATE.HUB`
 - else → go to `STATE.BOSS` (gate)This keeps branching logic in one place and prevents bugs.
- 7. Make endings stat-based**
Create `computeEnding()` that checks thresholds (health/level) and sets:
 - `STATE.WIN` or `STATE.LOSE` (or multiple endings ↓. you have them)
- 8. Check `index.html` script order (critical)**
Load files so definitions exist before use:
`ui.js` → `fx.js` → `scenes.js` → `start.js` → `instructions.js` → `game.js` → `win.js` → `lose.js` → `main.js`
(Important rule: `main.js` LAST.)
- 9. Debug blank screens using the console**
If it goes blank after switching states, open DevTools Console. The error will usually be: missing function name, missing file load, or unknown state value. Fix by ensuring every `STATE.X` has a matching draw function.
That's the workflow: **define states → create one screen function per state → route from `main.js` → keep logic centralized (`applyChoice`, `nextDayOrBoss`, `computeEnding`) → verify script order.**

Prompt 3

“Where should I put my resetGame() logic, so it reliably resets Health/Level/Day and sends the player back to the start screen without breaking state transitions?”

Answer (short):

Put resetGame () in your shared logic file (the one that holds game + STATE, like scenes.js) so every screen can call it. Reset the game. Health, game. Level and game. Day, and set game. state = STATE.START, then trigger it from your Win/Lose “Play Again” button.