

Side Quests and A4 (Individual Work)

I chose to keep the mechanic extremely simple by using only two door choices and an additive RGB “mixed world” ending, instead of adding movement, collisions, or extra levels. I made this decision so the project would stay reliable within the existing multi-state template and so the player’s choices would remain the main focus. This change made the game easier to test and polish, and it clearly demonstrated how a small interactive decision can create a meaningful, immediate outcome.

GenAI Documentation

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

Date Used: January 31, 2026

Tool Disclosure: ChatGPT 5.2 (OpenAI)

Purpose of Use:

Brainstorming a simple branching story structure that fits an existing p5.js multi-state template; outlining minimal variables/helpers for door choices and RGB mixing; confirming how to reuse existing states/files without expanding scope.

Summary of Interaction:

The tool suggested a clean decision-tree structure (two choices → two choices → ending) and recommended storing colour choices in an array, generating two random door options per pick, and using additive RGB mixing for the final background. It also proposed keeping the existing script order and repurposing the win screen as the ending.

Human Decision Point(s):

- The creative concept (colour-labeled doors) was my own inspiration from an episode from [The Apothecary Diaries](#)
 - GenAI was used for implementation planning.
- I chose to reuse existing screens (start, instr, game, win) rather than adding new state files, to keep the codebase stable and aligned with the template.
- I modified the suggested structure so it matched my template’s existing function naming and routing pattern.

Integrity & Verification Note:

I verified suggestions by checking that they matched p5.js conventions and my project’s screen-routing approach (single currentScreen router + per-screen draw/input functions). I also tested that the RGB mixing logic produces the expected outcomes (red+blue → purple/magenta, green+blue → cyan, red+green → yellow), and ensured the interaction flow stayed a true two-step decision tree.

Scope of GenAI Use:

GenAI did not write the final integrated project end-to-end. It did not validate my existing file/script setup, perform actual debugging in my environment, or test transitions/click areas. Final code organization, state routing integration, UI layout choices, and playtesting were done by me.

Limitations or Misfires:

Early responses were generic and sometimes assumed different state names or file structures than my template. The tool also could not account for my exact index.html script ordering and existing function naming conventions without my direction, so I had to adapt the plan to fit the template properly.

Summary of Process (Human + Tool)

I started by reviewing the existing template to understand how currentScreen routed between start, instruction, game, and win. I then used GenAI to quickly validate an implementation plan for a simple decision-tree interaction (two door choices, then a result). After that, I implemented the logic by adding shared state (picked colours and current door options), created door UI click zones, and wired the game flow to progress from Door 1 → Door 2 → ending. I iterated by playtesting each path, adjusting button hitboxes, and confirming that restarting correctly resets the stored choices.

Decision Points & Trade-offs

Decision 1: Add new files/states vs. repurpose existing template states

- **Options considered:** Create separate files like door1.js, door2.js, end.js vs. reuse game.js and win.js.
- **What changed:** I repurposed game.js to handle both door picks and repurposed win.js as the ending screen.
- **Why:** This reduced complexity, and matched the example organization.

Decision 2: Random outcomes (win/lose) vs. choice-based deterministic ending

- **Options considered:** Keep random win/lose logic vs. make the ending depend on the two chosen colours.
- **What changed:** I removed randomness and used additive RGB mixing to generate the ending.
- **Why:** It better demonstrates a decision tree and makes player choice directly influence the outcome.

Verification & Judgement

- Playtesting multiple runs (left/right choice combinations across both door picks).
- Confirming the logic always follows the intended flow: Door 1 pick → Door 2 pick → ending → restart.
- Checking visual correctness of colour mixing (expected mixed results appear as backgrounds).
- Re-checking that the implementation fits the intended learning goals: multi-state interaction, minimal complexity, and clear cause-and-effect from player decisions.

Limitations, Dead Ends, or Open Questions

- Random door options can sometimes repeat colours across different runs, which may reduce perceived variety (but keeps the code simple).
 - I intentionally avoided adding extra mechanics (sound, inventory, timers, animations) to keep scope small and aligned with the “simple branching story” goal.
-

Appendix

My Prompt:

I have an existing p5.js project that already uses multiple files and a screen router (currentScreen) with start, instructions, game, win, lose. I want to build an interactive story that branches like a small decision tree but without adding complicated code. Brainstorm a few simple story ideas that fit this structure.

ChatGPT-5.2:

Here are several simple interactive story ideas that work well with a multi-state p5.js template and don't require complex mechanics:

1. **Two Doors Dungeon (Decision Tree):**
 - Door 1: choose left or right
 - Door 2: new two-door choice
 - Ending: result depends on your two choicesImplementation: one array to store choices + a few screen states.
2. **Text Message Mystery:**
 - Choice: reply or ignore
 - Second choice: block or meet
 - Ending depends on pathImplementation: buttons or key presses + 2–3 states.

3. **Late for Class:**

- Choice: shortcut or main road
 - Second choice: run or stop for coffee
 - Ending: success/fail/funny outcome
- Implementation: optional “time” variable or purely choice-based.

If you want the simplest possible, the “two doors” structure is the cleanest decision tree: it’s just two choices and then an ending.

My Prompt:

I already have my concept: I want the story to be going through doors with labeled colours. I want to keep it extra simple: each door choice is an RGB colour. Example: Door 1 gives “green” or “blue.” If I choose “blue,” Door 2 gives two random options like “green” or “red.” After the second choice, it ends and the background becomes the mix of the two colours (blue + red = purple). How should I structure this with minimal new code?

ChatGPT-5.2:

You can implement this with minimal new code by using your existing screens and adding only a few shared variables.

Recommended structure (reusing your template):

- `start` → `instructions` → `game` (Door pick #1 and Door pick #2) → `win` (repurposed as ending)

Minimal variables:

- `pickedColors = []` → store two strings like `"blue"`, `"red"`
- `doorChoices = []` → store the two current door options

Minimal helper functions:

1. `pickTwoDoorColors()`
 - Randomly choose 2 distinct colours from `["red", "green", "blue"]`
2. `mixTwoColors(c1, c2)`
 - Add RGB values and clamp at 255
 - red (255,0,0) + blue (0,0,255) = (255,0,255) (purple/magenta)

Flow logic:

- On entering `game`, set `pickedColors = []` and `doorChoices = pickTwoDoorColors()`
- When player picks a door:
 - push colour into `pickedColors`
 - if first pick: reroll `doorChoices = pickTwoDoorColors()`
 - if second pick: go to ending screen (`win`) and draw background with `mixTwoColors(pickedColors[0], pickedColors[1])`

Choice UI:

- Two big rectangles (doors) with fill colour matching the option
- `mousePressed()` checks which rectangle was clicked and selects that colour

This keeps everything p5-only and fits the multi-file state router you already have.

My Prompt:

Implement it using my existing file layout (`start.js`, `instructions.js`, `game.js`, `win.js`, `main.js`). Keep code organized into sections. Use `p5.js` only.

ChatGPT-5.2:

Below is a minimal integration plan that reuses your file names and keeps the project organized.

Plan

- `main.js`: keep the router, add shared variables + helper functions
- `game.js`: replace old logic with door drawing + door pick handling
- `win.js`: replace “win” message with ending screen (mixed background)
- `start.js` and `instructions.js`: update text + transitions

Implementation approach

1. In `main.js`, define:
 - `COLORS = { red:[255,0,0], green:[0,255,0], blue:[0,0,255] }`
 - `pickedColors = []`
 - `doorChoices = []`
 - `pickTwoDoorColors(), mixTwoColors(), resetDoorRun()`
2. In `game.js`, implement:
 - `drawGame()` draws two doors using `doorChoices[0]` and `doorChoices[1]`

- `gameMousePressed()` checks door hitboxes and calls `pickDoorColor(choice)`
- `pickDoorColor()` pushes into `pickedColors`, rerolls if first pick, transitions to `win` if second pick
- 3. In `win.js`, implement:
 - `drawWin()` sets background to mixed colour and displays chosen colours
 - click or `R` restarts → returns to `game` with a fresh run

This fits your template and keeps code changes minimal.

My Prompt:

I also want keyboard support: press 1 for left door and 2 for right door. And press R to restart on the ending screen.

ChatGPT-5.2:

That's simple to add:

- In `gameKeyPressed()`:
 - If `key === "1"` → pick left door colour
 - If `key === "2"` → pick right door colour
- In `winKeyPressed()`:
 - If `key === "r" || key === "R"` → reset and go back to `game`

No extra libraries needed; it fits the existing `keyPressed()` routing pattern.
