GBDA 302 – Global Digital Project 2
Jowan Manjooran Jomon
jmanjoor **21105035**

Side Quest Week 3 - Process & Decision Documentation

1. What I was trying to do

For Week 3, I aimed to use the provided multi-screen p5.js starter structure to build a short branching interactive story. Instead of relying on random outcomes, my goal was to have player choices determine which scenes and ending they reached, inspired by the game *Dispatch*.

2. One small decision or change I made

I replaced the original random win/lose outcome in the game screen with a two-step decision structure, where the player makes an initial choice, encounters a follow-up scenario, and then reaches an ending based on those decisions.

3. Evidence of the change

- Random win/lose logic was removed from game.js
- Player choices are tracked using simple variables
- Different endings are reached based on decisions rather than chance

4. GenAI Use

I used GenAI (ChatGPT) to help explain the starter code, generate draft code for a branching decision structure, and refine on-screen instructional and narrative text. I reviewed, tested, and modified the generated code to ensure it aligned with the course template and worked correctly within the existing state-based structure.

Appendix: GenAI Transcript

Tool Used: ChatGPT

Prompt

```
 startt,js file // NOTE: Do NOT add setup() or draw() in this file

// setup() and draw() live in main.js

// This file only defines:

// 1) drawStart() → what the start/menu screen looks like

// 2) input handlers → what happens on click / key press on this screen

// 3) a helper function to draw menu buttons



// ------------------------------------------------------------

// Start screen visuals

// ------------------------------------------------------------

// drawStart() is called from main.js only when:

// currentScreen === "start"

function drawStart() {

  // Background colour for the start screen

  background(180, 225, 220); // soft teal background
```

```
// ---- Title text ----
fill(30, 50, 60);
textSize(46);
textAlign(CENTER, CENTER);
text("Win or Lose", width / 2, 180);

// ---- Buttons (data only) ----
// These objects store the position/size/label for each button.
// Using objects makes it easy to pass them into drawButton()
// and also reuse the same information for hover checks.
const startBtn = {
  x: width / 2,
  y: 320,
  w: 240,
  h: 80,
  label: "START",
};

const instrBtn = {
  x: width / 2,
  y: 430,
    w: 240,
    h: 80,
    label: "INSTRUCTIONS",
  };

  // Draw both buttons
  drawButton(startBtn);
  drawButton(instrBtn);

  // ---- Cursor feedback ----
  // If the mouse is over either button, show a hand cursor
  // so the player knows it is clickable.
  const over = isHover(startBtn) || isHover(instrBtn);
  cursor(over ? HAND : ARROW);
}

// ------------------------------------------------------------
// Mouse input for the start screen
// ------------------------------------------------------------
// Called from main.js only when currentScreen === "start"
function startMousePressed() {
  // For input checks, we only need x,y,w,h (label is optional)

  const startBtn = { x: width / 2, y: 320, w: 240, h: 80 };

  const instrBtn = { x: width / 2, y: 430, w: 240, h: 80 };


  // If START is clicked, go to the game screen

  if (isHover(startBtn)) {

    currentScreen = "game";

  }

  // If INSTRUCTIONS is clicked, go to the instructions screen

  else if (isHover(instrBtn)) {

    currentScreen = "instr";

  }

}


// ------------------------------------------------------------

// Keyboard input for the start screen

// ------------------------------------------------------------

// Provides keyboard shortcuts:

// - ENTER starts the game

// - I opens instructions

function startKeyPressed() {
  if (keyCode === ENTER) {
    currentScreen = "game";
  }

  if (key === "i" || key === "I") {
    currentScreen = "instr";
  }
}

// ------------------------------------------------------------
// Helper: drawButton()
// ------------------------------------------------------------
// This function draws a button and changes its appearance on hover.
// It does NOT decide what happens when you click the button.
// That logic lives in startMousePressed() above.
//
// Keeping drawing separate from input/logic makes code easier to read.
function drawButton({ x, y, w, h, label }) {
  rectMode(CENTER);

  // Check if the mouse is over the button rectangle
```

```
const hover = isHover({ x, y, w, h });


noStroke();


// ---- Visual feedback (hover vs not hover) ----
// This is a common UI idea:
// - normal state is calmer
// - hover state is brighter + more “active”
//
// We also add a shadow using drawingContext (p5 lets you access the
// underlying canvas context for effects like shadows).
if (hover) {
  fill(255, 200, 150, 220); // warm coral on hover


  // Shadow settings (only when hovered)
  drawingContext.shadowBlur = 20;
  drawingContext.shadowColor = color(255, 180, 120);
} else {
  fill(255, 240, 210, 210); // soft cream base


  // Softer shadow when not hovered
```

```
    drawingContext.shadowBlur = 8;
    drawingContext.shadowColor = color(220, 220, 220);
  }

  // Draw the rounded rectangle button
  rect(x, y, w, h, 14);

  // Important: reset shadow so it does not affect other drawings
  drawingContext.shadowBlur = 0;

  // Draw the label text on top of the button
  fill(40, 60, 70);
  textSize(28);
  textAlign(CENTER, CENTER);
  text(label, x, y);
}
```

Response

Good — your `start.js` is already correct structurally. For “Dispatch vibe,” you only need **small text + label changes** (optional: one extra subtitle). You do **not** need to touch the routing logic unless you want different keyboard shortcuts.

Here’s what to change **inside this file** (no new architecture):

✅ Changes to make in `drawStart()`

1) Change the title

Replace:

- `"Win or Lose"`

With something like:

- `"DISPATCH: Night Shift"`
  or
- `"DISPATCH / NIGHT SHIFT"`

Why: instantly sets theme (Design & Clarity points).

2) Add a subtitle line (optional but makes it feel “real”)

Under the title, add **one short line** like:

- `"02:13 AM • Storm warning • Lines unstable"`

This is just a text() call, no logic. Makes it Dispatch.

3) Rename the START button label (optional)

In `startBtn`, change label from:

- `"START"`

To:

- `"BEGIN SHIFT"` (best)
  or
- `"TAKE CALLS"`

Keep the instructions button as-is.

Everything else in `drawStart()` stays.

✅ Changes to make in `startMousePressed()` / `startKeyPressed()` (optional)

Option A (recommended): leave logic exactly the same

Keep:

- Start → `currentScreen = "game"`
- Instructions → `currentScreen = "instr"`
- ENTER starts, I opens instructions

This is fine and consistent with the template.

Option B (if you want Dispatch keyboard feel)

You *can* add shortcuts like:

- `S` for start shift
- `H` for help/instructions

But it's not required. Don't overcomplicate.

✅ What NOT to change

- Don't add `setup()` or `draw()` (main.js owns that).
- Don't change `isHover()` usage (it's shared and correct).
- Don't add new screens here.

Quick "before/after" checklist

After edits, your start screen should read like:

- **DISPATCH: Night Shift**
- “02:13 AM • Storm warning • Lines unstable”
- Button: **BEGIN SHIFT**
- Button: INSTRUCTIONS

That’s it ✅

If you paste your `instructions.js` next, I’ll tell you exactly what text to change there so it matches the branching story (still no code changes beyond text unless you want).

Prompt

```
// NOTE: Do NOT add setup() or draw() in this file
// setup() and draw() live in main.js
// This file only defines:
// 1) drawStart() → what the start/menu screen looks like
// 2) input handlers → what happens on click / key press on this screen
// 3) a helper function to draw menu buttons

// ------------------------------------------------------------
// Start screen visuals
// ------------------------------------------------------------
// drawStart() is called from main.js only when:
// currentScreen === "start"
function drawStart() {
```

```
// Background colour for the start screen
background(180, 225, 220); // soft teal background

// ---- Title text ----
fill(30, 50, 60);
textSize(46);
textAlign(CENTER, CENTER);

text("02:13 AM • Tornado warning • Lines unstable", width / 2, 180);

// ---- Buttons (data only) ----
// These objects store the position/size/label for each button.
// Using objects makes it easy to pass them into drawButton()
// and also reuse the same information for hover checks.
const startBtn = {
  x: width / 2,
  y: 320,
  w: 240,
  h: 80,
  label: "BEGIN SHIFT",
};
```

```
  const instrBtn = {

    x: width / 2,

    y: 430,

    w: 240,

    h: 80,

    label: "INSTRUCTIONS",

  };


  // Draw both buttons

  drawButton(startBtn);

  drawButton(instrBtn);


  // ---- Cursor feedback ----

  // If the mouse is over either button, show a hand cursor

  // so the player knows it is clickable.

  const over = isHover(startBtn) || isHover(instrBtn);

  cursor(over ? HAND : ARROW);

}


// ------------------------------------------------------------
// Mouse input for the start screen
// ------------------------------------------------------------
// Called from main.js only when currentScreen === "start"
function startMousePressed() {
  // For input checks, we only need x,y,w,h (label is optional)
  const startBtn = { x: width / 2, y: 320, w: 240, h: 80 };
  const instrBtn = { x: width / 2, y: 430, w: 240, h: 80 };

  // If START is clicked, go to the game screen
  if (isHover(startBtn)) {
    currentScreen = "game";
  }
  // If INSTRUCTIONS is clicked, go to the instructions screen
  else if (isHover(instrBtn)) {
    currentScreen = "instr";
  }
}

// ------------------------------------------------------------
// Keyboard input for the start screen
// ------------------------------------------------------------
// Provides keyboard shortcuts:
// - ENTER starts the game
// - I opens instructions
function startKeyPressed() {
  if (keyCode === ENTER) {
    currentScreen = "game";
  }

  if (key === "i" || key === "I") {
    currentScreen = "instr";
  }
}

// ------------------------------------------------------------
// Helper: drawButton()
// ------------------------------------------------------------
// This function draws a button and changes its appearance on hover.
// It does NOT decide what happens when you click the button.
// That logic lives in startMousePressed() above.
//
// Keeping drawing separate from input/logic makes code easier to read.
```

```
function drawButton({ x, y, w, h, label }) {
  rectMode(CENTER);


  // Check if the mouse is over the button rectangle
  const hover = isHover({ x, y, w, h });


  noStroke();


  // ---- Visual feedback (hover vs not hover) ----
  // This is a common UI idea:
  // - normal state is calmer
  // - hover state is brighter + more “active”
  //
  // We also add a shadow using drawingContext (p5 lets you access the
  // underlying canvas context for effects like shadows).
  if (hover) {
    fill(255, 200, 150, 220); // warm coral on hover


    // Shadow settings (only when hovered)
    drawingContext.shadowBlur = 20;
    drawingContext.shadowColor = color(255, 180, 120);
  } else {
    fill(255, 240, 210, 210); // soft cream base

    // Softer shadow when not hovered
    drawingContext.shadowBlur = 8;
    drawingContext.shadowColor = color(220, 220, 220);
  }

  // Draw the rounded rectangle button
  rect(x, y, w, h, 14);

  // Important: reset shadow so it does not affect other drawings
  drawingContext.shadowBlur = 0;

  // Draw the label text on top of the button
  fill(40, 60, 70);
  textSize(28);
  textAlign(CENTER, CENTER);
  text(label, x, y);
}  change the code accordingly and also make sure you give a line break after title
```

Response

You’re close. Two fixes:

1. You accidentally replaced the **title** with the **subtitle** (you need *both*).
2. You want a **line break** after the title → easiest is: **draw title and subtitle as two separate `text()` calls** at different Y positions (p5 doesn’t auto-line-break unless you use `\n`, but spacing with two calls looks cleaner).

Here’s the updated `start.js` with:

- Title: **DISPATCH: Night Shift**
- Subtitle on the next line: **02:13 AM • Tornado warning • Lines unstable**
- A clear line break/spacing between them
- Everything else unchanged (buttons + logic)

```
// NOTE: Do NOT add setup() or draw() in this file

// setup() and draw() live in main.js

// This file only defines:

// 1) drawStart() → what the start/menu screen looks like

// 2) input handlers → what happens on click / key press on this screen

// 3) a helper function to draw menu buttons


// ------------------------------------------------------------

// Start screen visuals

// ------------------------------------------------------------

// drawStart() is called from main.js only when:

// currentScreen === "start"

function drawStart() {
```

```
// Background colour for the start screen
background(180, 225, 220); // soft teal background

// ---- Title + subtitle text ----
fill(30, 50, 60);
textAlign(CENTER, CENTER);

// Title (top line)
textSize(50);
text("DISPATCH: Night Shift", width / 2, 160);

// Subtitle (line break / next line)
textSize(18);
text("02:13 AM • Tornado warning • Lines unstable", width / 2, 205);

// ---- Buttons (data only) ----
const startBtn = {
  x: width / 2,
  y: 320,
  w: 240,
  h: 80,
    label: "BEGIN SHIFT",
  };

  const instrBtn = {
    x: width / 2,
    y: 430,
    w: 240,
    h: 80,
    label: "INSTRUCTIONS",
  };

  // Draw both buttons
  drawButton(startBtn);
  drawButton(instrBtn);

  // ---- Cursor feedback ----
  const over = isHover(startBtn) || isHover(instrBtn);
  cursor(over ? HAND : ARROW);
}

// ------------------------------------------------------------
// Mouse input for the start screen
// ------------------------------------------------------------
function startMousePressed() {
  const startBtn = { x: width / 2, y: 320, w: 240, h: 80 };
  const instrBtn = { x: width / 2, y: 430, w: 240, h: 80 };

  if (isHover(startBtn)) {
    currentScreen = "game";
  } else if (isHover(instrBtn)) {
    currentScreen = "instr";
  }
}

// ------------------------------------------------------------
// Keyboard input for the start screen
// ------------------------------------------------------------
function startKeyPressed() {
  if (keyCode === ENTER) {
    currentScreen = "game";
  }
  if (key === "i" || key === "I") {
    currentScreen = "instr";
  }
}

// ------------------------------------------------------------
// Helper: drawButton()
// ------------------------------------------------------------
function drawButton({ x, y, w, h, label }) {
  rectMode(CENTER);

  const hover = isHover({ x, y, w, h });

  noStroke();

  if (hover) {
    fill(255, 200, 150, 220);
    drawingContext.shadowBlur = 20;
    drawingContext.shadowColor = color(255, 180, 120);
  } else {
    fill(255, 240, 210, 210);
    drawingContext.shadowBlur = 8;
    drawingContext.shadowColor = color(220, 220, 220);
  }

  rect(x, y, w, h, 14);

  drawingContext.shadowBlur = 0;

  fill(40, 60, 70);
  textSize(28);
  textAlign(CENTER, CENTER);
  text(label, x, y);
}
```

If you paste your `instructions.js` next, I’ll rewrite the instruction text so it matches the Dispatch decision tree (and remove the “random result” wording).

Prompt

this is win // NOTE: Do NOT add setup() or draw() in this file

// setup() and draw() live in main.js

// This file only defines:

```
// 1) drawWin() → what the win screen looks like
// 2) input handlers → how the player returns to the start screen
//
// This file is intentionally very similar to lose.js.
// The goal is to show that win/lose screens are often
// simple “end states” with minimal logic.


// ------------------------------------------------------------
// Main draw function for win screen
// ------------------------------------------------------------
// drawWin() is called from main.js
// only when currentScreen === "win"
function drawWin() {
  // Green-tinted background to communicate success
  background(200, 255, 200);


  fill(0);
  textAlign(CENTER, CENTER);


  // Main success message
  textSize(40);
  text("You Win!", width / 2, 300);

  // Instruction text
  textSize(20);
  text("Click or press R to return to Start.", width / 2, 360);
}

// ------------------------------------------------------------
// Mouse input for win screen
// ------------------------------------------------------------
// Any mouse click returns the player to the start screen
function winMousePressed() {
  currentScreen = "start";
}

// ------------------------------------------------------------
// Keyboard input for win screen
// ------------------------------------------------------------
// R is commonly used for “restart” in games
function winKeyPressed() {
  if (key === "r" || key === "R") {
    currentScreen = "start";
  }
}
```

Response

```
// NOTE: Do NOT add setup() or draw() in this file
// setup() and draw() live in main.js
// This file only defines:
// 1) drawWin() → what the win screen looks like
// 2) input handlers → how the player returns to the start screen
//
// This file is intentionally very similar to lose.js.
// The goal is to show that win/lose screens are often
// simple “end states” with minimal logic.

// ------------------------------------------------------------
// Main draw function for win screen
// ------------------------------------------------------------
// drawWin() is called from main.js
// only when currentScreen === "win"
function drawWin() {
  // Calm green-tinted background (success / relief)
  background(200, 255, 200);


  fill(0);

  textAlign(CENTER, CENTER);


  // ---- Title (top line) ----

  textSize(44);

  text("CONTROLLED RESOLUTION", width / 2, 260);


  // ---- Outcome (line break / next line) ----

  textSize(18);

  const outcome =

    "Rescue arrives prepared.\n" +

    "No lives lost.\n" +

    "You traded urgency for judgment.";

  text(outcome, width / 2, 335);


  // ---- Instruction (separate line) ----

  textSize(18);

  text("Click or press R to return to Start.", width / 2, 470);

}
```

```
// ------------------------------------------------------------

// Mouse input for win screen

// ------------------------------------------------------------

// Any mouse click returns the player to the start screen

function winMousePressed() {

  currentScreen = "start";

}



// ------------------------------------------------------------

// Keyboard input for win screen

// ------------------------------------------------------------

// R is commonly used for “restart” in games

function winKeyPressed() {

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

    currentScreen = "start";

  }

}
```

Prompt

```
main.js // ------------------------------------------------------------

// main.js = the “router” (traffic controller) for the whole game
// ------------------------------------------------------------
//
// Idea: this project has multiple screens (start, instructions, game, win, lose).
// Instead of putting everything in one giant file, each screen lives in its own
// file and defines two main things:
//   1) drawX()         → how that screen looks
//   2) XMousePressed() / XKeyPressed() → how that screen handles input
//
// This main.js file does 3 important jobs:
//   A) stores the current screen in a single shared variable
//   B) calls the correct draw function each frame
//   C) sends mouse/keyboard input to the correct screen handler

// ------------------------------
// Global game state
// ------------------------------
// This variable is shared across all files because all files run in the same
// global JavaScript scope when loaded in index.html.
//
// We store the “name” of the current screen as a string.
// Only one screen should be active at a time.
```

```
let currentScreen = "start"; // "start" | "instr" | "game" | "win" | "lose"


// ------------------------------
// setup() runs ONCE at the beginning
// ------------------------------
// This is where you usually set canvas size and initial settings.
function setup() {
  createCanvas(800, 800);

  // Sets a default font for all text() calls
  // (This can be changed later per-screen if you want.)
  textFont("sans-serif");
}


// ------------------------------
// draw() runs every frame (many times per second)
// ------------------------------
// This is the core “router” for visuals.
// Depending on currentScreen, we call the correct draw function.
function draw() {
  // Each screen file defines its own draw function:
```

```
  //   start.js        → drawStart()
  //   instructions.js  → drawInstr()
  //   game.js         → drawGame()
  //   win.js          → drawWin()
  //   lose.js         → drawLose()

  if (currentScreen === "start") drawStart();
  else if (currentScreen === "instr") drawInstr();
  else if (currentScreen === "game") drawGame();
  else if (currentScreen === "win") drawWin();
  else if (currentScreen === "lose") drawLose();

  // (Optional teaching note)
  // This “if/else chain” is a very common early approach.
  // Later in the course you might replace it with:
  // - a switch statement, or
  // - an object/map of screens
}

// ------------------------------
// mousePressed() runs once each time the mouse is clicked
// ------------------------------
// This routes mouse input to the correct screen handler.
function mousePressed() {
  // Each screen *may* define a mouse handler:
  // start.js        → startMousePressed()
  // instructions.js  → instrMousePressed()
  // game.js         → gameMousePressed()
  // win.js          → winMousePressed()
  // lose.js         → loseMousePressed()

  if (currentScreen === "start") startMousePressed();
  else if (currentScreen === "instr") instrMousePressed();
  else if (currentScreen === "game") gameMousePressed();
  // The ?.() means “call this function only if it exists”
  // This prevents errors if a screen doesn’t implement a handler.
  else if (currentScreen === "win") winMousePressed?.();
  else if (currentScreen === "lose") loseMousePressed?.();
}

// ------------------------------
// keyPressed() runs once each time a key is pressed
// ------------------------------
// This routes keyboard input to the correct screen handler.
function keyPressed() {
  // Each screen *may* define a key handler:
  // start.js        → startKeyPressed()
  // instructions.js  → instrKeyPressed()
  // game.js         → gameKeyPressed()
  // win.js          → winKeyPressed()
  // lose.js         → loseKeyPressed()

  if (currentScreen === "start") startKeyPressed();
  else if (currentScreen === "instr") instrKeyPressed();
  else if (currentScreen === "game") gameKeyPressed?.();
  else if (currentScreen === "win") winKeyPressed?.();
  else if (currentScreen === "lose") loseKeyPressed?.();
}

// ------------------------------------------------------------
// Shared helper function: isHover()
// ------------------------------------------------------------
//
```

```
// Many screens have buttons.
// This helper checks whether the mouse is inside a rectangle.
//
// Important: our buttons are drawn using rectMode(CENTER),
// meaning x,y is the CENTRE of the rectangle.
// So we check mouseX and mouseY against half-width/half-height bounds.
//
// Input:  an object with { x, y, w, h }
// Output: true if mouse is over the rectangle, otherwise false
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
  );
}
```

Reponse

Your `main.js` is **already correct** and should work with all the changes we made. ✅

What I checked (and why it's good)

- `currentScreen` values match your screens: `"start" | "instr" | "game" | "win" | "lose"` ✅
- `draw()` routes correctly to `drawStart`, `drawInstr`, `drawGame`, `drawWin`, `drawLose` ✅
- `mousePressed()` calls the right handlers ✅
- `keyPressed()` calls `gameKeyPressed?.()` which is perfect because your new `game.js` defines `gameKeyPressed()` ✅
- `isHover()` matches your buttons since you draw them with `rectMode(CENTER)` ✅

Only 1 optional improvement (not required)

In `mousePressed()` you call `gameMousePressed()` without `?.()`. That's fine because your `game.js` defines `gameMousePressed()`.

If you ever remove/rename that function later, it would error. If you want it extra safe, you *could* change:

else if (currentScreen === "game") gameMousePressed();

to:

else if (currentScreen === "game") gameMousePressed?.();

But again: **not necessary** as long as `gameMousePressed()` exists (it does).

Last thing to verify for GitHub Pages

Your `index.html` must load files in this order (screen files first, then `main.js` last): `start.js`, `instructions.js`, `game.js`, `win.js`, `lose.js`, `main.js`

If you paste your `index.html` I'll confirm it's exactly right and you're basically submission-ready.

Prompt

fix the main .js and cleanup redundant code // --

// main.js = the "router" (traffic controller) for the whole game

// --

//

// Idea: this project has multiple screens (start, instructions, game, win, lose).

// Instead of putting everything in one giant file, each screen lives in its own

// file and defines two main things:

// 1) drawX() → how that screen looks

// 2) XMousePressed() / XKeyPressed() → how that screen handles input

//

// This main.js file does 3 important jobs:

// A) stores the current screen in a single shared variable

// B) calls the correct draw function each frame

// C) sends mouse/keyboard input to the correct screen handler

// ------------------------------

// Global game state

```
// ------------------------------
// This variable is shared across all files because all files run in the same
// global JavaScript scope when loaded in index.html.
//
// We store the “name” of the current screen as a string.
// Only one screen should be active at a time.
let currentScreen = "start"; // "start" | "instr" | "game" | "win" | "lose"

// ------------------------------
// setup() runs ONCE at the beginning
// ------------------------------
// This is where you usually set canvas size and initial settings.
function setup() {
  createCanvas(800, 800);

  // Sets a default font for all text() calls
  // (This can be changed later per-screen if you want.)
  textFont("sans-serif");
}

// ------------------------------
// draw() runs every frame (many times per second)
// ------------------------------
// This is the core “router” for visuals.
// Depending on currentScreen, we call the correct draw function.
function draw() {
  // Each screen file defines its own draw function:
  //   start.js         → drawStart()
  //   instructions.js  → drawInstr()
  //   game.js          → drawGame()
  //   win.js           → drawWin()
  //   lose.js          → drawLose()

  if (currentScreen === "start") drawStart();
  else if (currentScreen === "instr") drawInstr();
  else if (currentScreen === "game") drawGame();
  else if (currentScreen === "win") drawWin();
  else if (currentScreen === "lose") drawLose();

  // (Optional teaching note)
  // This “if/else chain” is a very common early approach.
  // Later in the course you might replace it with:
```

```
  // - a switch statement, or
  // - an object/map of screens
}


// ------------------------------
// mousePressed() runs once each time the mouse is clicked
// ------------------------------
// This routes mouse input to the correct screen handler.
function mousePressed() {
  // Each screen *may* define a mouse handler:
  // start.js        → startMousePressed()
  // instructions.js  → instrMousePressed()
  // game.js         → gameMousePressed()
  // win.js          → winMousePressed()
  // lose.js         → loseMousePressed()

  if (currentScreen === "start") startMousePressed();
  else if (currentScreen === "instr") instrMousePressed();
  else if (currentScreen === "game") gameMousePressed();
  // The ?.() means “call this function only if it exists”
  // This prevents errors if a screen doesn’t implement a handler.
  else if (currentScreen === "win") winMousePressed?.();
  else if (currentScreen === "lose") loseMousePressed?.();
}

// ------------------------------
// keyPressed() runs once each time a key is pressed
// ------------------------------
// This routes keyboard input to the correct screen handler.
function keyPressed() {
  // Each screen *may* define a key handler:
  // start.js        → startKeyPressed()
  // instructions.js  → instrKeyPressed()
  // game.js          → gameKeyPressed()
  // win.js          → winKeyPressed()
  // lose.js         → loseKeyPressed()

  if (currentScreen === "start") startKeyPressed();
  else if (currentScreen === "instr") instrKeyPressed();
  else if (currentScreen === "game") gameKeyPressed?.();
  else if (currentScreen === "win") winKeyPressed?.();
  else if (currentScreen === "lose") loseKeyPressed?.();
```

```
}


// ------------------------------------------------------------

// Shared helper function: isHover()

// ------------------------------------------------------------

//

// Many screens have buttons.

// This helper checks whether the mouse is inside a rectangle.

//

// Important: our buttons are drawn using rectMode(CENTER),

// meaning x,y is the CENTRE of the rectangle.

// So we check mouseX and mouseY against half-width/half-height bounds.

//

// Input:  an object with { x, y, w, h }

// Output: true if mouse is over the rectangle, otherwise false

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
```

```
 );
}
```

Response

Here’s a “cleaner + safer” `main.js` that matches your updated project. Changes I made:

- Uses a **switch** (cleaner than long if/else)
- Uses optional chaining `?.()` for **all** handlers (prevents crashes if something is missing)
- Keeps `isHover()` exactly the same
- No changes to your screen names (`"start"`, `"instr"`, `"game"`, `"win"`, `"lose"`)

Paste this over your `main.js`:

```
// ------------------------------------------------------------
// main.js = the “router” (traffic controller) for the whole game
// ------------------------------------------------------------
//
// This project has multiple screens (start, instructions, game, win, lose).
// Each screen lives in its own file and defines:
//   1) drawX()             → how that screen looks
//   2) XMousePressed/XKeyPressed → how that screen handles input
//
// This main.js file:
//   A) stores the current screen
//   B) calls the correct draw function each frame
//   C) routes mouse/keyboard input to the correct handler


// ------------------------------

// Global game state

// ------------------------------

let currentScreen = "start"; // "start" | "instr" | "game" | "win" | "lose"


// ------------------------------

// setup() runs ONCE at the beginning

// ------------------------------

function setup() {

  createCanvas(800, 800);

  textFont("sans-serif");

}


// ------------------------------

// draw() runs every frame

// ------------------------------

function draw() {

  switch (currentScreen) {

    case "start":
      drawStart?.();

      break;

    case "instr":

      drawInstr?.();

      break;

    case "game":

      drawGame?.();

      break;

    case "win":

      drawWin?.();

      break;

    case "lose":

      drawLose?.();

      break;

    default:

      // Failsafe: if something sets an unknown screen, return to start

      currentScreen = "start";

      break;

  }

}
```

```
// ------------------------------

// mousePressed() routes mouse input

// ------------------------------

function mousePressed() {

  switch (currentScreen) {

    case "start":

      startMousePressed?.();

      break;

    case "instr":

      instrMousePressed?.();

      break;

    case "game":

      gameMousePressed?.();

      break;

    case "win":

      winMousePressed?.();

      break;

    case "lose":

      loseMousePressed?.();

      break;

  }
```

```
}

// ------------------------------
// keyPressed() routes keyboard input
// ------------------------------
function keyPressed() {
  switch (currentScreen) {
    case "start":
      startKeyPressed?.();
      break;
    case "instr":
      instrKeyPressed?.();
      break;
    case "game":
      gameKeyPressed?.();
      break;
    case "win":
      winKeyPressed?.();
      break;
    case "lose":
      loseKeyPressed?.();
      break;
  }
}


// ------------------------------------------------------------
// Shared helper: isHover()
// ------------------------------------------------------------
// Buttons are drawn using rectMode(CENTER).
// This helper checks if mouseX/mouseY are inside the rectangle.
function isHover({ x, y, w, h }) {
  return (
    mouseX > x - w / 2 &&
    mouseX < x + w / 2 &&
    mouseY > y - h / 2 &&
    mouseY < y + h / 2
  );
}
```

If you paste your `index.html`, I'll confirm the script order is correct for GitHub Pages (that's the #1 reason these multi-file sketches break).