Process & Decision Documentation

Project/Assignment Decisions

For Side Quest Week 3 I led the idea creating game flow. I then used AI – Microsoft 365 CoPilot – to generate the necessary code to bring my idea to life. I outlined the idea of my choose your own adventure game, what I wanted each screen to be and say as well as the outcomes of each action. My game involves choosing between going home and to the forest. In the forest you choose between chasing a bird, which leads to a "lose" screen and leaving the bird alone, leading to a winning screen. Going home just leads to neutral screen. I also decided to add another screen instead of repurposing one screen for two choices to make my game more legitimate. I will explain my interaction with AI and findings in more detail below.

GenAI Documentation

Date Used: January 30th, 2026

Tool Disclosure: Microsoft 365 Copilot

Purpose of Use: I used AI for code creation/structure as well as debugging.

Summary of Interaction: The tool contributed by generating code I could use to replace and add to the example code.

Human Decision Point(s): The GenAI output suggested that I copy and paste the entire code to replace the example code but rather than that I replaced the parts that were necessary with the code that Copilot gave me. I also made the decision to create a separate screen for my game choice, "home", rather than repurpose the "lose" screen.

Integrity & Verification Note: I check the GenAI for appropriateness, bias, and incorrectness by carefully reading through and selecting the code that it gave me.

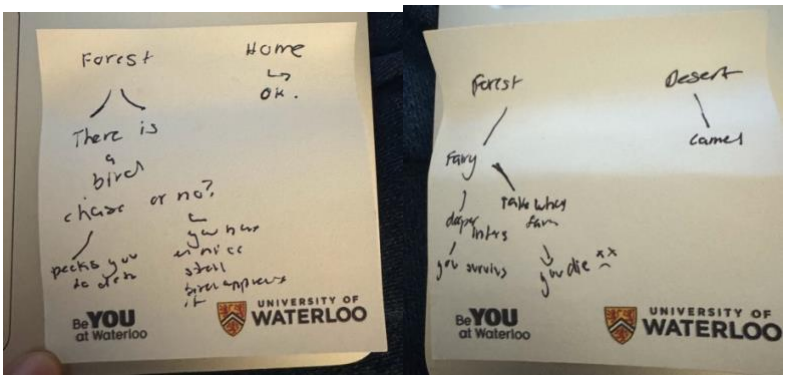
Scope of GenAI Use: The creation of the game idea and structure of how it would play out was done entirely by me. The new game code was generated entirely by AI while the titles in the html was edited by me. The AI did help and remind me to add the new file and game screen to the html file.

Limitations or Misfires: The tool did miss some spacing issues which I manually fixed myself. Some text was overlapping while others were too close together.

Summary of Process (Human + Tool)

Describe what you did, focusing on process rather than outcome. This may include:

- Wrote down some ideas for my game manually
- Decided on my game structure as well as text ideas
- Asked GenAI to create code
- Inserted the code into the example template
- Asked the AI to make a new screen
- Inserted the new screen code into visual studio code
- Play tested the game and realized the “home” button was not working
- Moments of uncertainty, failure, or rework: Asked the AI to help me fix this issue and realized I had forgotten to update the html file to include the new “home” file.



Decision Points & Trade-offs

The key decision I made was deciding to add another screen for the “home” option rather than integrating it with the losing screen. I made this decision because I wanted my game to see more fleshed out rather than an afterthought.

Verification & Judgement

I evaluated by playtesting which is how I caught mistakes and missteps. After most changes I played the game in live server to see what it looked like and if I and the AI was on the right track.

Limitations, Dead Ends, or Open Questions

The AI although very helpful, was unsure when it came to fixing my issue of the “home” button not leading anywhere. The first suggestion it offered said to remove an extra bracket

that was never there. This shows that AI can have an issue of creating things that do not exist which is why it is important to look over and check all generated content.

Appendix

My prompt:

If I wanted to create a CYOA game using this code which js file woudl ie dit for each aspect. I want it to be you are on a walk and you can choose to go to the forest or home. if you go home then it just says okay boring have a nice day. If you go to forest you then have teh choice of you see a bird do you chase it or leave it alone? if you chase it, it pecks you to death and you die and if you don't you "win" and have a nice walk. This is the 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); }) this is game.js (// NOTE: Do NOT add setup() or draw() in this file // setup() and draw() live in main.js // This file only defines: // 1) drawGame() → what the game screen looks like // 2) input handlers → what happens when the player clicks or presses keys // 3) helper functions specific to this screen

// ------------------------------ // Button data // ------------------------------ // This object stores all the information needed to draw // and interact with the button on the game screen. // Keeping this in one object makes it easier to move, // resize, or restyle the button later. const gameBtn = { x: 400, // x position (centre of the button) y: 550, // y position (centre of the button) w: 260, // width h: 90, // height label: "PRESS HERE", // text shown on the button };

// ------------------------------ // Main draw function for this screen // ------------------------------ // drawGame() is called from main.js *only* // when currentScreen === "game" function drawGame() { // Set background colour for the game screen background(240, 230, 140);

// ---- Title and instructions text ---- fill(0); // black text textSize(32); textAlign(CENTER, CENTER); text("Game Screen", width / 2, 160);

textSize(18); text("Click the button (or press ENTER) for a random result.", width / 2, 210,);

// ---- Draw the button ---- // We pass the button object to a helper function drawGameButton(gameBtn);

// ---- Cursor feedback ---- // If the mouse is over the button, show a hand cursor // Otherwise, show the normal arrow cursor cursor(isHover(gameBtn) ? HAND : ARROW); }

// ------------------------------ // Button drawing helper // ------------------------------ // This function is responsible *only* for drawing the button. // It does NOT handle clicks or game logic. function drawGameButton({ x, y, w, h, label }) { rectMode(CENTER);

// Check if the mouse is hovering over the button // isHover() is defined in main.js so it can be shared const hover = isHover({ x, y, w, h });

noStroke();

// Change button colour when hovered // This gives visual feedback to the player fill(hover ? color(180, 220, 255, 220) // lighter blue on hover : color(200, 220, 255, 190), // normal state);

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

// Draw the button text fill(0); textSize(28); textAlign(CENTER, CENTER); text(label, x, y); }

// ------------------------------ // Mouse input for this screen // ------------------------------ // This function is called from main.js // only when currentScreen === "game" function gameMousePressed() { // Only trigger the outcome if the button is clicked if (isHover(gameBtn)) { triggerRandomOutcome(); } }

// ------------------------------ // Keyboard input for this screen // ------------------------------ // Allows keyboard-only interaction (accessibility + design) function gameKeyPressed() { // ENTER key triggers the same behaviour as clicking the button if (keyCode === ENTER) { triggerRandomOutcome(); } }

// ------------------------------ // Game logic: win or lose // ------------------------------ // This function decides what happens next in the game. // It does NOT draw anything. function triggerRandomOutcome() { // random() returns a value between 0 and 1 // Here we use a 50/50 chance: // - less than 0.5 → win // - 0.5 or greater → lose // // You can bias this later, for example: // random() < 0.7 → 70% chance to win if (random() < 0.5) { currentScreen = "win"; } else { currentScreen = "lose"; } }) this is win.js (// 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"; } }) this is lose.js (// NOTE: Do NOT add setup() or draw() in this file // setup() and draw() live in main.js // This file only defines: // 1) drawLose() → what the lose screen looks like // 2) input handlers → how the player returns to the start screen

// ---------- // Main draw function for lose screen // ---------- // drawLose() is called from main.js // only when currentScreen === "lose" function drawLose() { // Red-tinted background to communicate failure background(255, 210, 210);

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

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

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

// ---------- // Mouse input for lose screen // ---------- // Any mouse click returns the player to the start screen // (no buttons needed for this simple end state) function loseMousePressed() { currentScreen = "start"; }

// ---------- // Keyboard input for lose screen // ---------- // R is commonly used for “restart” in games function loseKeyPressed() { if (key === "r" || key === "R") { currentScreen = "start"; } }) this is start.js (// 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); })

Microsoft Copilot’s Response:

Great question! You already have the perfect structure for a small CYOA in p5.js: **main.js** routes between screens, and each screen lives in its own JS file. Here’s exactly **which file to edit for each part** of your “Walk → Forest vs Home → Bird choice” story, plus ready-to-paste code.

What to edit where (high level)

- **`start.js`** → Change the start menu to present the first choice:
  - Buttons: **FOREST** and **HOME**
  - If **HOME** → go straight to an end screen that says “Okay, boring, have a nice day.”
  - If **FOREST** → go to the forest scene (we’ll reuse the existing `game` screen for that).
- **`game.js`** → Turn this into the **forest** scene:
  - Show the prompt: *“You see a bird. Do you chase it or leave it alone?”*
  - Two buttons: **CHASE IT** and **LEAVE IT ALONE**
  - If **CHASE IT** → go to **lose**

    - If **LEAVE IT ALONE** → go to **win**
    - Remove the random outcome (no coin flip anymore).
- **`win.js`** → Change text to your "win" ending:
    - *"You leave the bird alone and enjoy a nice walk. You win!"*
- **`lose.js`** → Use this for **two endings** (to keep your existing router unchanged):
    - If you want: use it only for the **pecked to death** ending.
    - Or, simplest: also use it for **HOME** (boring ending) by changing the message to something neutral.
    - Below, I give you a version that keeps the **pecked** ending here and routes HOME → lose screen with the "boring" text. (If you want them separated, I'll show how to add a new "home" screen too.)
- **`main.js`** → **No changes required** unless you want brand new screen names. We'll reuse:
    - `start` (menu)
    - `game` (forest with bird choice)
    - `win` (nice walk)
    - `lose` (peck ending, and optionally home boring ending)

You don't need to add `setup()`/`draw()` anywhere else—they already live in `main.js`.

Drop-in changes per file

1) `start.js` → Show "FOREST" and "HOME"

Replace the **entire** file content with this:

```
// Start Screen: Choose FOREST or HOME
function drawStart() {
  background(180, 225, 220); // soft teal

  // Title
  fill(30, 50, 60);
  textSize(46);
  textAlign(CENTER, CENTER);
  text("You're on a Walk", width / 2, 160);

  // Prompt
  textSize(22);
  text("Where do you go?", width / 2, 210);

  // Buttons
  const forestBtn = { x: width / 2, y: 320, w: 240, h: 80, label: "FOREST" };
  const homeBtn   = { x: width / 2, y: 430, w: 240, h: 80, label: "HOME" };
```

```
  drawButton(forestBtn);
  drawButton(homeBtn);

  // Hand cursor if hovering
  const over = isHover(forestBtn) || isHover(homeBtn);
  cursor(over ? HAND : ARROW);
}

function startMousePressed() {
  const forestBtn = { x: width / 2, y: 320, w: 240, h: 80 };
  const homeBtn   = { x: width / 2, y: 430, w: 240, h: 80 };

  if (isHover(forestBtn)) {
    currentScreen = "game"; // → forest scene
  } else if (isHover(homeBtn)) {
    // Simplest: send HOME to the lose screen (we'll show the boring message there)
    currentScreen = "lose";
  }
}

function startKeyPressed() {
  // Optional keyboard shortcuts
  if (keyCode === ENTER) {
    currentScreen = "game"; // default to forest if they press ENTER
  }
  if (key === "h" || key === "H") {
    currentScreen = "lose"; // home via H
  }
}

// Reuse your existing drawButton() from your start.js (unchanged)
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);
}
```

2) `game.js` → Forest screen with "Bird: Chase or Leave?"

Replace the **entire** file content with this:

```
// Forest Screen: You see a bird. Chase it or leave it alone?

// Two buttons for the forest choice
const chaseBtn = {
  x: 260, y: 560, w: 260, h: 90, label: "CHASE IT",
};
const leaveBtn = {
  x: 540, y: 560, w: 260, h: 90, label: "LEAVE IT ALONE",
};

function drawGame() {
  background(240, 230, 140); // a warm outdoor tone

  // Title / Prompt
  fill(0);
  textAlign(CENTER, CENTER);

  textSize(36);
  text("Forest", width / 2, 140);

  textSize(22);
  text("You see a bird. What do you do?", width / 2, 200);

  // Scene hint (optional)
  textSize(16);
  text("Tip: choose wisely…", width / 2, 240);

  // Draw buttons
  drawChoiceButton(chaseBtn);
  drawChoiceButton(leaveBtn);

  // Cursor feedback
  const over = isHover(chaseBtn) || isHover(leaveBtn);
```

```
  cursor(over ? HAND : ARROW);
}

// Button rendering helper
function drawChoiceButton({ x, y, w, h, label }) {
  rectMode(CENTER);
  const hover = isHover({ x, y, w, h });

  noStroke();
  fill(hover ? color(180, 220, 255, 220) : color(200, 220, 255, 190));
  rect(x, y, w, h, 14);

  fill(0);
  textSize(24);
  textAlign(CENTER, CENTER);
  text(label, x, y);
}

function gameMousePressed() {
  if (isHover(chaseBtn)) {
    currentScreen = "lose"; // pecked to death
  } else if (isHover(leaveBtn)) {
    currentScreen = "win";  // peaceful walk
  }
}

function gameKeyPressed() {
  // Optional keyboard shortcuts:
  if (key === "c" || key === "C") currentScreen = "lose"; // chase
  if (key === "l" || key === "L") currentScreen = "win";  // leave
}
```

Note: we removed the old random win/lose logic and the single “PRESS HERE” button.

---

3) `win.js` → Nice walk ending

Replace the **entire** file content with this:

```
function drawWin() {
  background(200, 255, 200); // green-tinted: success
  fill(0);
  textAlign(CENTER, CENTER);

  textSize(40);
```

```
  text("You Win!", width / 2, 260);

  textSize(22);
  text("You leave the bird alone and enjoy a nice walk.", width / 2, 320);

  textSize(18);
  text("Click or press R to return to the Start.", width / 2, 380);
}

function winMousePressed() {
  currentScreen = "start";
}

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

4) `lose.js` → Two possible texts (home or pecked)

Simplest option: Use the same `lose` screen for both "HOME (boring)" and "CHASE (pecked)." We'll display a combined message that covers both outcomes.

Replace the **entire** file content with this:

```
function drawLose() {
  background(255, 210, 210); // red-tinted: failure or end
  fill(0);
  textAlign(CENTER, CENTER);

  textSize(40);
  text("The End", width / 2, 240);

  textSize(22);
  // Combined message so it works whether you arrived by HOME or CHASE
  text(
    "If you went home: Okay, boring—have a nice day.\n" +
    "If you chased the bird: It pecks you to death. You die.",
    width / 2, 320
  );

  textSize(18);
  text("Click or press R to return to the Start.", width / 2, 410);
}

function loseMousePressed() {
  currentScreen = "start";
```

```
}

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

Do I need to change `main.js`?

No, not if you:

- Route **FOREST** → `"game"`
- Route **HOME** → `"lose"`
- Route **CHASE** → `"lose"`
- Route **LEAVE ALONE** → `"win"`

Your existing `main.js` already knows how to draw `start`, `instr`, `game`, `win`, `lose` and forward mouse/keyboard events correctly.

(Optional) Showing exactly which file controls what

- **First choice (Forest/Home)** → **`start.js`**
- **Forest scene & the bird choice** → **`game.js`**
- **Ending: nice walk** → **`win.js`**
- **Ending: boring home or pecked** → **`lose.js`**
- **Routing & input dispatch** → **`main.js`** (no story text here)

Want a dedicated “Home (boring)” screen?

Say the word and I’ll provide:

- `home.js` with its own draw + input
- A one-line addition in `main.js` router
- Change `start.js` to route HOME → `"home"` instead of `"lose"`

If you paste the snippets above into the matching files, your CYOA should run right away. Want me to also tweak the **instructions screen** (if you have one) to describe the new choices?

My prompt:

Would prefer separate endings please for the losing

Microsoft Copilot's Response:

Awesome—let's give **HOME** its own separate ending screen and keep **CHASE → pecked** on the existing **lose** screen.

Below are small, focused edits you can paste in. You'll:

1. Add a **new screen file**: `home.js`
2. Make a **tiny update** to `main.js` (teach the router about `"home"`)
3. Update **`start.js`** to route the **HOME** button to `"home"` (instead of `"lose"`)
4. Keep **`lose.js`** exclusively for the "pecked to death" outcome
5. Keep **`win.js`** for "nice walk" outcome

---

1) New file: `home.js` (boring-but-happy ending)

Create a new file `home.js` and include it in your `index.html` after `main.js` (and before or after other screens—it'll work as long as it loads).

```
// home.js
// A simple separate ending for going home.

function drawHome() {
  background(225, 235, 245); // calm bluish tone
  fill(0);
  textAlign(CENTER, CENTER);

  textSize(40);
  text("Home", width / 2, 240);

  textSize(22);
  text("Okay, boring—have a nice day.", width / 2, 300);

  textSize(18);
  text("Click or press R to return to the Start.", width / 2, 360);
}
```

```
function homeMousePressed() {
  currentScreen = "start";
}

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

2) Update `main.js`: add `"home"` to the router

You only need to add `"home"` to:

- the `currentScreen` type comment (optional but nice),
- the `draw()` router,
- `mousePressed()` router, and
- `keyPressed()` router.

Minimal edits in `main.js`

A. (Optional) Update the comment where `currentScreen` is declared:

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

B. Add `home` to the `draw()` router:

```
function draw() {
  if (currentScreen === "start") drawStart();
  else if (currentScreen === "instr") drawInstr();
  else if (currentScreen === "game") drawGame();
  else if (currentScreen === "win") drawWin();
  else if (currentScreen === "lose") drawLose();
  else if (currentScreen === "home") drawHome(); // ← add this line
}
```

C. Add `home` to `mousePressed()`:

JavaScript

```
function mousePressed() {
if (currentScreen === "start") startMousePressed();
else if (currentScreen === "instr") instrMousePressed();
else if (currentScreen === "game") gameMousePressed();
else if (currentScreen === "win") winMousePressed?.();
else if (currentScreen === "lose") loseMousePressed?.();
else if (currentScreen === "home") homeMousePressed?.(); // ← add this line
```

```
}
```

D. Add `home` to `keyPressed()`:

```
function keyPressed() {
  if (currentScreen === "start") startKeyPressed();
  else if (currentScreen === "instr") instrKeyPressed();
  else if (currentScreen === "game") gameKeyPressed?.();
  else if (currentScreen === "win") winKeyPressed?.();
  else if (currentScreen === "lose") loseKeyPressed?.();
  else if (currentScreen === "home") homeKeyPressed?.(); // ← add this line
}
```

3) Update `start.js`: route HOME → `"home"`

If you used my earlier "combined ending" version, change the **HOME** route back to `"home"`. Below is a full `start.js` that matches your CYOA:

```
// Start Screen: Choose FOREST or HOME
function drawStart() {
  background(180, 225, 220); // soft teal

  // Title
  fill(30, 50, 60);
  textSize(46);
  textAlign(CENTER, CENTER);
  text("You're on a Walk", width / 2, 160);

  // Prompt
  textSize(22);
  text("Where do you go?", width / 2, 210);

  // Buttons
  const forestBtn = { x: width / 2, y: 320, w: 240, h: 80, label: "FOREST" };
  const homeBtn   = { x: width / 2, y: 430, w: 240, h: 80, label: "HOME" };

  drawButton(forestBtn);
  drawButton(homeBtn);

  // Hand cursor if hovering
  const over = isHover(forestBtn) || isHover(homeBtn);
  cursor(over ? HAND : ARROW);
}

function startMousePressed() {
  const forestBtn = { x: width / 2, y: 320, w: 240, h: 80 };
```

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

  if (isHover(forestBtn)) {
    currentScreen = "game"; // → forest scene
  } else if (isHover(homeBtn)) {
    currentScreen = "home"; // → separate HOME ending
  }
}

function startKeyPressed() {
  // Optional keyboard shortcuts
  if (keyCode === ENTER) currentScreen = "game"; // default to forest
  if (key === "h" || key === "H") currentScreen = "home";
}

// Keep your existing drawButton() helper (unchanged)
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);
}
```

4) `game.js` (forest choice) — keep as the bird scene

If you already swapped `game.js` to the forest scene (CHASE vs LEAVE), you're good. Here it is again for convenience:

```
// Forest Screen: You see a bird. Chase it or leave it alone?
```

```
const chaseBtn = { x: 260, y: 560, w: 260, h: 90, label: "CHASE IT" };
const leaveBtn = { x: 540, y: 560, w: 260, h: 90, label: "LEAVE IT ALONE" };

function drawGame() {
  background(240, 230, 140); // warm outdoor tone

  fill(0);
  textAlign(CENTER, CENTER);
  textSize(36);
  text("Forest", width / 2, 140);

  textSize(22);
  text("You see a bird. What do you do?", width / 2, 200);

  textSize(16);
  text("Tip: choose wisely…", width / 2, 240);

  drawChoiceButton(chaseBtn);
  drawChoiceButton(leaveBtn);

  const over = isHover(chaseBtn) || isHover(leaveBtn);
  cursor(over ? HAND : ARROW);
}

function drawChoiceButton({ x, y, w, h, label }) {
  rectMode(CENTER);
  const hover = isHover({ x, y, w, h });

  noStroke();
  fill(hover ? color(180, 220, 255, 220) : color(200, 220, 255, 190));
  rect(x, y, w, h, 14);

  fill(0);
  textSize(24);
  textAlign(CENTER, CENTER);
  text(label, x, y);
}

function gameMousePressed() {
  if (isHover(chaseBtn)) currentScreen = "lose"; // pecked
  else if (isHover(leaveBtn)) currentScreen = "win"; // peaceful walk
}

function gameKeyPressed() {
  if (key === "c" || key === "C") currentScreen = "lose";
```

```
  if (key === "l" || key === "L") currentScreen = "win";
}
```

5) `lose.js` (pecked ending only)

Make sure `lose.js` describes just the **pecked to death** outcome now:

```
function drawLose() {
  background(255, 210, 210); // red-tinted
  fill(0);
  textAlign(CENTER, CENTER);

  textSize(40);
  text("You Lose!", width / 2, 240);

  textSize(22);
  text("You chased the bird. It pecks you to death. You die.", width / 2, 300);

  textSize(18);
  text("Click or press R to return to the Start.", width / 2, 360);
}

function loseMousePressed() {
  currentScreen = "start";
}

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

6) `win.js` (unchanged—nice walk)

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

  textSize(40);
  text("You Win!", width / 2, 260);

  textSize(22);
  text("You leave the bird alone and enjoy a nice walk.", width / 2, 320);

  textSize(18);
  text("Click or press R to return to the Start.", width / 2, 380);
}

function winMousePressed() {
```

```
  currentScreen = "start";
}

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

Final checklist

- Add `home.js` and include it in `index.html`.
- Update `main.js` to route `"home"`.
- Update `start.js` HOME button → `currentScreen = "home"`.
- Keep `lose.js` only for the **pecked to death** ending.
- Keep `win.js` as the **nice walk** ending.
- (Optional) Update your instructions screen (`instructions.js`) to reflect the new choices.

If you want, I can also make the **instructions** screen match this story flow or add **keyboard shortcuts** to HOME/FOREST directly from Start. Want me to do that?

My Prompt

what issue could I possibly be having because pressing the home button or the h key does nothing and essentially freezes my screen. this is my home.js (// home.js // A simple separate ending for going home.

function drawHome() { background(225, 235, 245); // calm bluish tone fill(0); textAlign(CENTER, CENTER);

textSize(40); text("Home", width / 2, 240);

textSize(22); text("Okay, boring—have a nice day.", width / 2, 300);

textSize(18); text("Click or press R to return to the Start.", width / 2, 360); }

function homeMousePressed() { currentScreen = "start"; }

function homeKeyPressed() { if (key === "r" || key === "R") { currentScreen = "start"; } }) this is my 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" | "home"

// ------------------------------ // 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() // home.js → drawHome()

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

// ------------------------------ // 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() // home.js → homeMousePressed()

if (currentScreen === "start") startMousePressed(); else if (currentScreen === "instr") instrMousePressed(); else if (currentScreen === "game") gameMousePressed(); else if (currentScreen === "home") homeMousePressed(); // 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() // home.js → homeKeyPressed()

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

(currentScreen === "win") winKeyPressed?.(); else if (currentScreen === "home") homeKeyPressed?.(); 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); }) this is my start.js (// 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("You're on a Walk", width / 2, 180);

// Prompt textSize(22); text("Where do you go?", width / 2, 240);

// ---- 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 forestBtn = { x: width / 2, y: 320, w: 240, h: 80, label: "FOREST", };

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

// Draw both buttons drawButton(forestBtn); drawButton(homeBtn);

// ---- Cursor feedback ---- // If the mouse is over either button, show a hand cursor // so the player knows it is clickable. const over = isHover(forestBtn) || isHover(homeBtn); 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 forestBtn = { x: width / 2, y: 320, w: 240, h: 80 }; const homeBtn = { x: width / 2, y: 430, w: 240, h: 80 };

// If FOREST is clicked, go to the game screen if (isHover(forestBtn)) { currentScreen = "game"; } // If HOME is clicked, go to the home screen else if (isHover(homeBtn)) { currentScreen = "home"; } }

// ---------- // Keyboard input for the start screen // ---------- // Provides keyboard shortcuts: // - ENTER defaults to forest // - h takes you home function startKeyPressed() { if (keyCode === ENTER) { currentScreen = "game"; }

if (key === "h" || key === "H") { currentScreen = "home"; } }

// ---------- // 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); }) additionally some of my js files have an M next to them in VS Code

Microsoft Copilot's Response

Good news: the behavior you describe (clicking **HOME** or pressing **H** does nothing and the app "freezes") is almost always caused by a **JavaScript error** that stops p5's draw loop and input handlers from running. I can see at least **one definite syntax error** in your `home.js`, plus a couple of common gotchas to check.

---

1) Fix the syntax error in `home.js`

Your `home.js` ends with `});`. That extra `)` will throw a syntax error and prevent the rest of your scripts from running properly.

Your file (problem):

```
function homeKeyPressed() {
  if (key === "r" || key === "R") {
    currentScreen = "start";
  }
}) // ← extra ) here
```

Fix it by deleting the extra `)` so the file ends like this:

```
// home.js
// A simple separate ending for going home.

function drawHome() {
  background(225, 235, 245); // calm bluish tone
  fill(0);
  textAlign(CENTER, CENTER);

  textSize(40);
  text("Home", width / 2, 240);

  textSize(22);
  text("Okay, boring—have a nice day.", width / 2, 300);

  textSize(18);
  text("Click or press R to return to the Start.", width / 2, 360);
}

function homeMousePressed() {
  currentScreen = "start";
}

function homeKeyPressed() {
```

```
  if (key === "r" || key === "R") {
    currentScreen = "start";
  }
}
```

fter saving, **reload the page** and test again.

---

2) Make sure `home.js` is actually loaded in `index.html`

If `home.js` isn't included (or its `<script>` tag has a typo), then when you switch to the `"home"` screen, `drawHome`/`homeMousePressed`/`homeKeyPressed` won't exist and the app can appear to freeze.

In your `index.html`, ensure you have a script tag for `home.js`. Order isn't super strict with your router, but a safe order is:

```
<script src="p5.min.js"></script>

<script src="main.js"></script>
<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="home.js"></script> <!-- ← make sure this is present -->
``
```

(If you don't have an `instructions.js`, omit it, but the comment in `main.js` references it.)

---

3) Double-check your router edits (they look mostly right)

Your `main.js` routing looks good. Two small best-practice tweaks:

- It's okay that you call `homeMousePressed()` without optional chaining, because you only do that when `currentScreen === "home"`. Just make sure `home.js` is **loaded** by the time you can navigate to `"home"` (see #2).
- In `keyPressed()` you already use optional chaining for most screens; you also did it for `home` (`homeKeyPressed?.()`), which is perfect.

---

4) Confirm the Start screen is actually sending you to `"home"`

Your `start.js` is correct:

```
else if (isHover(homeBtn)) {
  currentScreen = "home";
}
```

and:

```
if (key === "h" || key === "H") {
  currentScreen = "home";
}
```

So as long as `home.js` loads and has no syntax errors, pressing **H** or clicking **HOME** should switch to the Home screen.

---

5) Check the browser console (this pinpoints the issue)

Open **DevTools** → **Console** and reload the page. If you still see problems, look for red error messages like:

- `Uncaught SyntaxError: Unexpected token ')'` → that's the extra `)` in `home.js`.
- `Uncaught ReferenceError: drawHome is not defined` → `home.js` wasn't loaded or had a syntax error earlier in the file.
- `Uncaught TypeError: homeMousePressed is not a function` → same root cause: `home.js` didn't load correctly.

Fix the reported error, reload, and it should work.

---

6) About the "M" next to files in VS Code

That **"M"** in the Explorer is just **Git** telling you the file is **Modified** (has changes not yet committed). It's not an error and won't affect your program running.

Quick sanity checklist

- `home.js` ends cleanly with `}` (no extra `)` or stray characters).
- `home.js` is included in `index.html` via `home.js</script>`.
- Console shows **no red errors** after reload.

- From the Start screen:
    - Clicking **HOME** sets `currentScreen` to `"home"`.
    - Pressing **H** sets `currentScreen` to `"home"`.
- On the Home screen, clicking or pressing **R** returns to Start.

If you still hit an issue after fixing the stray `)`, paste me the **console error message** (if any) and your **`index.html` script tags**, and I'll pinpoint the next step in seconds.