

Process & Decision Documentation

Side Quests and A4 (Individual Work)

For my sidequest for this week, I focused on changing the functionality of the win/lose example to a version that allows users to play the game by detecting the circles as they spawn on the screen. Users can also end the game and 'lose' when they decide to.

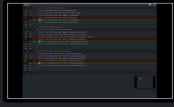
One significant design decision I had to make was including a 'lose' functionality. Unlike the original example where losing or winning was based on a random number, my game's functionality did not inherently have a way to lose or exit out of the game screen. Due to time constraints, I simply included "endbtn" similar to "gamebtn" that loaded up the 'lose' screen when pressed.

```
game.js
44 49
45 - // ---- Draw the button ----
46 - // We pass the button object to a helper function
47 - drawGameButton(gameBtn);
50 + if (!gameStarted) {
51 +   text("Click the button to begin.", width / 2, 210);
52 +
53 +   // Draw button ONLY before the game starts
54 +   drawGameButton(gameBtn);
55 +
56 +   // Cursor changes only for button hover
57 +   cursor(isHover(gameBtn) ? HAND : ARROW);
58 +   return; // stop here so circle doesn't draw yet
59 + }
60 +
61 + // After game starts:
62 + text("Click the circle to spawn the next one.", width / 2, 210);
63 +
64 + // Draw the circle target
65 + fill(255, 0, 0);
66 + circle(x, y, r * 2);
48 67
49 - // ---- Cursor feedback ----
50 - // If the mouse is over the button, show a hand cursor
51 - // Otherwise, show the normal arrow cursor
52 - cursor(isHover(gameBtn) ? HAND : ARROW);
68 + drawGameButton(endBtn);
69 +
70 + // Cursor becomes a hand when hovering the circle or the button
71 + const overCircle = dist(mouseX, mouseY, x, y) < r;
72 + const overEndBtn = isHover(endBtn);
73 + cursor(overCircle || overEndBtn ? HAND : ARROW);
53 74
54 75
55 76 // -----
@@ -91,9 +112,21 @@ function drawGameButton({ x, y, w, h, label }) {
91 112 // only when currentScreen === "game"
92 113 function gameMousePressed() {
```

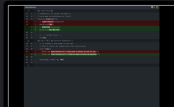
```
game.js
94 - if (isHover(gameBtn)) {
95 -   triggerRandomOutcome();
115 + if (!gameStarted) {
116 +   if (isHover(gameBtn)) {
117 +     gameStarted = true;
118 +     newTarget();
119 +   }
120 +   return;
96 121 }
122 +
123 + if (isHover(endBtn)) {
124 +   currentScreen = "lose";
125 +   return;
126 + }
127 +
128 + // After start: clicking the circle spawns the next one
129 + checkhit(mouseX, mouseY);
97 130 }
98 131
99 132 // -----
133 +
134 + @@ -101,28 +134,31 @@ function gameMousePressed() {
135 + // -----
136 + // Allows keyboard-only interaction (accessibility + design)
137 + function gameKeyPressed() {
138 +   // ENTER key triggers the same behaviour as clicking the button
139 +   if (keyCode === ENTER) {
140 +     triggerRandomOutcome();
141 +   }
142 +   if (!gameStarted && keyCode === ENTER) {
143 +     gameStarted = true;
144 +     newTarget();
145 +     return;
146 +   }
147 +   if (gameStarted && keyCode === ENTER) {
148 +     checkhit(mouseX, mouseY);
107 149 }
108 150 }
109 151
110 152 // -----
```

```
instructions.js
11 11 @@ -11,8 +11,8 @@
12 12 // drawInstr() is called from main.js
13 13 // only when currentScreen === "instr"
14 14 function drawInstr() {
15 15 - // Light neutral background
16 16 - background(240);
17 17 + // dusk blue background
18 18 + background(53, 80, 112);
19 19
20 20 // ---- Screen title ----
21 21 fill(0);
22 22
23 23 @@ -26,7 +26,7 @@ function drawInstr() {
24 24 // \n creates a line break in the text
25 25 // This is useful for simple multi-line instructions
26 26 const lines =
27 27 - "Press the game button.\n" + "You have a chance to win or lose!";
28 28 + "Press the start button.\n" + "Find as many circles as possible!";
29 29
30 30 text(lines, width / 2, 160);
31 31
32 32
```

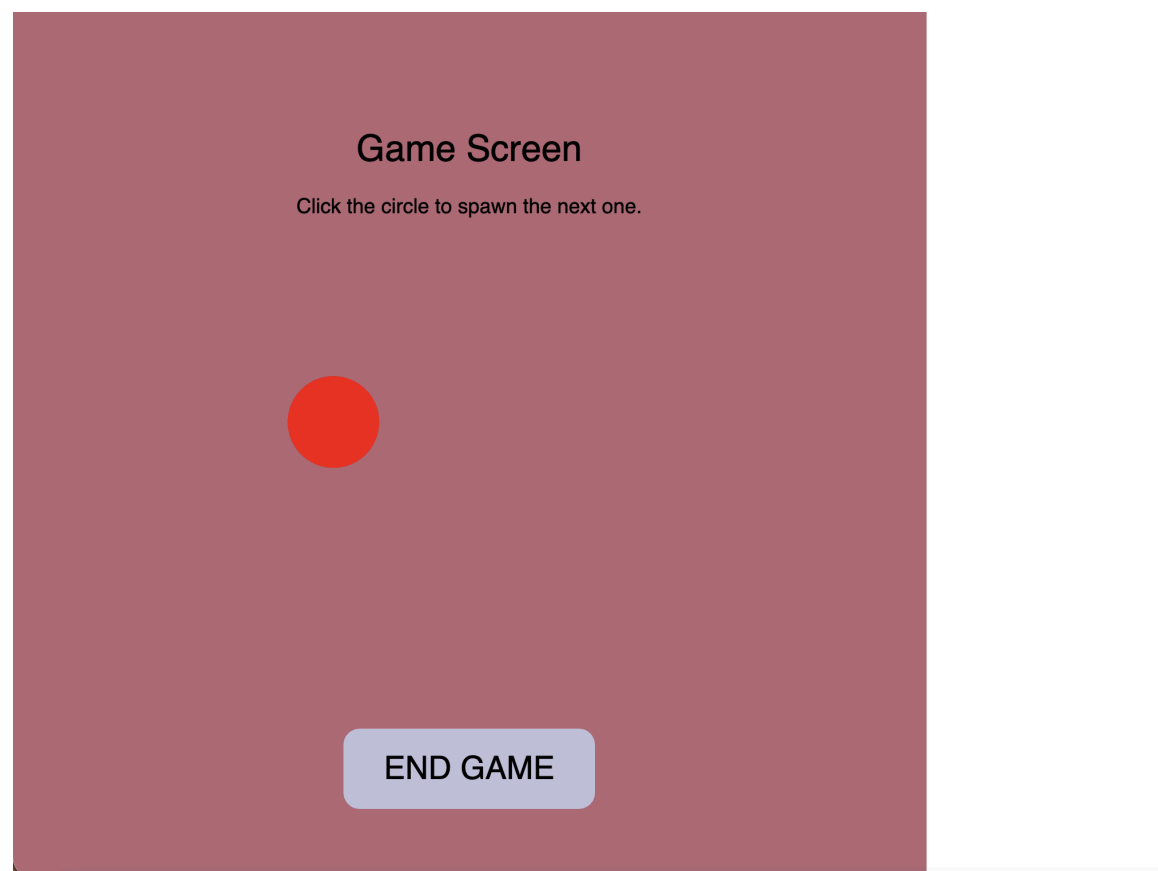
```
start.js
↑
@@ -12,13 +12,13 @@
12 12 // currentScreen == "start"
13 13 function drawStart() {
14 14 // Background colour for the start screen
15 15 - background(180, 225, 228); // soft teal background
16 16 + background(229, 107, 111); // light coral
17 17 // ---- Title text ----
18 18 fill(30, 50, 60);
19 19 textSize(46);
20 20 textAlign(CENTER, CENTER);
21 21 - text("Win or Lose", width / 2, 180);
22 22 + text("Lose or Lose", width / 2, 180);
23 23 // ---- Buttons (data only) ----
24 24 // These objects store the position/size/label for each button.
↓
```



```
lose.js
↑
@@ -10,8 +10,8 @@
10 10 // drawLose() is called from main.js
11 11 // only when currentScreen == "lose"
12 12 function drawLose() {
13 13 - // Red-tinted background to communicate failure
14 14 - background(255, 210, 210);
15 15 + // dusty lavender
16 16 + background(109, 89, 122);
17 17 fill(0);
18 18 textAlign(CENTER, CENTER);
↓
```



```
main.js
@@ -51,7 +51,7 @@ function draw() {
51 51     if (currentScreen === "start") drawStart();
52 52     else if (currentScreen === "instr") drawInstr();
53 53     else if (currentScreen === "game") drawGame();
54 -   else if (currentScreen === "win") drawWin();
54 +   // else if (currentScreen === "win") drawWin();
55 55     else if (currentScreen === "lose") drawLose();
56 56
57 57     // (Optional teaching note)
@@ -78,7 +78,7 @@ function mousePressed() {
78 78     else if (currentScreen === "game") gameMousePressed();
79 79     // The ?() means "call this function only if it exists"
80 80     // This prevents errors if a screen doesn't implement a handler.
81 -   else if (currentScreen === "win") winMousePressed?();
81 +   // else if (currentScreen === "win") winMousePressed?();
82 82     else if (currentScreen === "lose") loseMousePressed?();
83 83 }
84 84
@@ -97,7 +97,7 @@ function keyPressed() {
97 97     if (currentScreen === "start") startKeyPressed();
98 98     else if (currentScreen === "instr") instrKeyPressed();
99 99     else if (currentScreen === "game") gameKeyPressed?();
100 -   else if (currentScreen === "win") winKeyPressed?();
100 +   // else if (currentScreen === "win") winKeyPressed?();
101 101     else if (currentScreen === "lose") loseKeyPressed?();
102 102 }
103 103
```



Appendix

what is wrong with my code-

```
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(181, 101, 118);
// ---- 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 to
begin. \n", 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)) { checkhit(mouseX, mouseY); } } let x, y, r = 40; //circle
//let score = 0; function newTarget() { x = random(width); // new random x position y =
random(height); // new random y position } // ----- // 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) { checkhit(mouseX, mouseY); } } //
```

```

----- // Game logic: click the circle // ----- // This
function decides what happens next in the game. // It does NOT draw anything. function
checkhit(mx, my) { //check if the circle let d = dist(mx, my, x, y); if (d < r) { newTarget(); }
} 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); newTarget(); // 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() { fill(255, 0, 0);
circle(x, y, r * 2); // 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 ); }

```

Thought for 7s

Two big things are “wrong” here (and they’re the reason your circle/click logic feels broken):

1) You draw the circle in **main.js** before the game screen draws its background

In **draw()** you do:

```
fill(255, 0, 0);
```

```
circle(x, y, r * 2);
```

```
else if (currentScreen === "game") drawGame();
```

But inside **drawGame()** you call:

```
background(181, 101, 118);
```

So the circle gets drawn... then immediately gets covered by the background. That's why it looks like it's not spawning.

Fix (best)

Move the circle drawing into `drawGame()` **after** the background:

```
function drawGame() {  
  background(181, 101, 118);  
  
  // draw target circle here (after background)  
  fill(255, 0, 0);  
  circle(x, y, r * 2);  
  
  // ...rest of your UI  
  drawGameButton(gameBtn);  
  cursor(isHover(gameBtn) ? HAND : ARROW);  
}
```

And **delete** the circle drawing from `main.js`.

2) Your click handler only checks the button, not the circle

Right now your `gameMousePressed()` does this:

```
if (isHover(gameBtn)) {  
  checkhit(mouseX, mouseY);  
}
```

So `checkhit()` only ever runs when you click the **button**, not when you click the **circle**.

Fix (simple)

Let clicks anywhere on the game screen check the circle, but still keep the button for whatever purpose:

```
function gameMousePressed() {  
    // if you want the button to do something specific, do it here  
    if (isHover(gameBtn)) {  
        // example: start/reset, go to instructions, etc.  
        return;  
    }  
  
    // otherwise, clicking anywhere checks if you hit the circle  
    checkhit(mouseX, mouseY);  
}
```

Or if you want *only clicking the circle* to matter, you still need to call `checkhit()` even when not on the button.

Bonus gotcha (common)

Make sure `index.html` loads files in an order where `newTarget()` exists before `setup()` runs.

If your script order is:

```
<script src="main.js"></script>  
<script src="game.js"></script>
```

Then `setup()` runs and calls `newTarget()` before `game.js` defines it → crash / undefined.

Safe order:

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

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

(Or define `newTarget()` in `main.js`, since it's shared.)

If you tell me your `index.html` script order + what you want the button to do (start game? reset target? go to lose screen?), I can give you the cleanest final version.

Now create a end game button that takes us to the lose game screen

Perfect, this fits really cleanly into your existing screen system 👍

We'll add **an End Game button on the game screen** that switches `currentScreen` to `"lose"`.

I'll keep this **simple, readable, and consistent** with how your other buttons work.

① Add an End Game button object (GAME.JS)

Put this **under your existing `gameBtn`**:

```
const endGameBtn = {  
  x: 400,  
  y: 680,  
  w: 260,  
  h: 70,  
  label: "END GAME",  
};
```

② Draw the End Game button in `drawGame()`

Inside `drawGame()`, **after** your main button and circle are drawn:

```
// draw end game button
```

```
drawGameButton(endGameBtn);
```

Your `drawGame()` should now conceptually look like:

```
function drawGame() {
```

```
    background(181, 101, 118);
```

```
    // target circle
```

```
    fill(255, 0, 0);
```

```
    circle(x, y, r * 2);
```

```
    // text
```

```
    fill(0);
```

```
    textSize(32);
```

```
    textAlign(CENTER, CENTER);
```

```
    text("Game Screen", width / 2, 160);
```

```
    // buttons
```

```
    drawGameButton(gameBtn);
```

```
    drawGameButton(endGameBtn);
```

```
    cursor(
```

```
        isHover(gameBtn) || isHover(endGameBtn)
```

```
        ? HAND
```

```
        : ARROW
```

```
    );
```

```
}
```

③ Handle clicking the End Game button

Update `gameMousePressed()` in `game.js`:

```
function gameMousePressed() {  
  // end game button → go to lose screen  
  if (isHover(endGameBtn)) {  
    currentScreen = "lose";  
    return;  
  }  
  
  // optional: other button behaviour  
  if (isHover(gameBtn)) {  
    return;  
  }  
  
  // clicking anywhere else checks circle hit  
  checkhit(mouseX, mouseY);  
}
```

④ (Optional but recommended) Keyboard support

Inside `gameKeyPressed()`:

```
function gameKeyPressed() {  
  // ESC key ends the game  
  if (keyCode === ESCAPE) {
```

```
    currentScreen = "lose";  
}  
  
if (keyCode === ENTER) {  
    checkhit(mouseX, mouseY);  
}  
}
```