

Process & Decision Documentation

Entry Header

Name: Ksenia Li

Primary responsibility for this work: Side quest enhancement, add game to provided arrays using week 4 example 1 zipped files

Goal of Work Session

Briefly describe what you were trying to accomplish during this phase of the assignment.

- Create a game out of the provided week 4 array examples
- Add an additional level
- Create a win and lose game state

Tools, Resources, or Inputs Used

- ChatGPT 5.2

GenAI Documentation

I used GenAI to help me create the moving blob, obstacles and game over and win game states.

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

Date Used: February 8th

Tool Disclosure: ChatGPT 5.2

Purpose of Use: Generate code for game

Summary of Interaction: Helped me create a game by generating code. I reviewed it and edited the arrays and obstacles to my liking. ChatGPT helped me code an interface I would not have known how to do on my own.

Appendix

for side quest 4 i need to make this code

```
const TS = 32; // TILE SIZE: pixels per grid cell (32x32 squares) /* GRID LEGEND
(how numbers map to visuals): - 0 = floor (walkable, light gray) - 1 = wall (blocked, dark
teal) */ const grid = [ // Row 0 (top edge - all walls) [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1], // Row 1 (open hallway with wall in middle) [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
1], // Row 2 (complex maze pattern) [1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1], // Row 3
[1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1], // Row 4 [1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0,
1], // Row 5 [1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], // Row 6 [1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
0, 1, 1, 1, 0, 1], // Row 7 [1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1], // Row 8 [1, 0, 1, 1, 0,
1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1], // Row 9 [1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], // Row
10 (bottom edge - all walls) [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], ]; /* p5.js SETUP:
Runs once when sketch loads */ function setup() { // Canvas size = grid dimensions ×
tile size // grid[0].length = 16 columns, grid.length = 11 rows // Canvas = 16×32 = 512px
wide, 11×32 = 352px tall createCanvas(grid[0].length * TS, grid.length * TS); // Drawing
style setup noStroke(); // No black outlines on tiles (clean look) textFont("sans-serif"); //
Clean font for UI text textSize(14); // Small text size for HUD } /* p5.js DRAW: Runs 60
times per second (game loop) */ function draw() { // Clear screen with light gray
background each frame background(240); /* CORE RENDERING LOOP: Draw every
tile in the grid Nested loops: - Outer loop: iterate ROWS (r = 0 to 10) - Inner loop: iterate
COLUMNS in each row (c = 0 to 15) */ for (let r = 0; r < grid.length; r++) { for (let c = 0; c
< grid[0].length; c++) { // TILE TYPE CHECK: What kind of tile is at grid[r][c]? if (grid[r][c]
=== 1) { // WALL TILE: Dark teal colour (RGB: 30, 50, 60) fill(30, 50, 60); } else { //
FLOOR TILE: Light gray (RGB: 230, 230, 230) fill(230); } /* CONVERT GRID COORDS
→ SCREEN COORDS: - Grid: r=0,c=3 → Screen: x=96, y=0 - Grid: r=5,c=7 → Screen:
x=224, y=160 - x = column × TS y = row × TS */ rect(c * TS, r * TS, TS, TS); } } // UI
LABEL: Explain what students are seeing fill(0); // Black text text("Static array → grid
render", 10, 16); }
```

into Generate a level using arrays or JSON data; use loops to dynamically place tiles, obstacles, or words. Bonus: Add a second level that loads automatically when the first is finished. // how do i do that

You're already **using an array + loops to place tiles** (that's the core requirement).
What's missing is:

1. **"Generate a level"** (i.e., load different grid data, not hard-code one map forever)
2. **"Finish the level"** (define a win condition)
3. **"Auto-load level 2"** when level 1 is finished (bonus)

Below is a clean pattern that hits all 3, using **JSON-like level objects** (still just JS objects), **loops to render**, plus a simple **player + goal** so "finished" means something.

Drop-in example (2 levels, auto-advance)

Copy/paste this whole sketch into your Side Quest 4 file and tweak the grids.

```
const TS = 32; // tile size
```

```
/*
```

```
TILE LEGEND
```

```
0 = floor
```

```
1 = wall
```

```
2 = goal (finish tile)
```

```
*/
```

```
const LEVELS = [
```

```
{
```

```
  name: "Level 1",
```

```
  grid: [
```

```
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
[1,0,0,0,0,1,0,0,0,0,0,1,0,0,2,1],
[1,0,1,1,0,1,0,1,1,1,0,1,0,1,0,1],
[1,0,1,0,0,0,0,0,0,1,0,0,0,1,0,1],
[1,0,1,0,1,1,1,1,0,1,1,1,0,1,0,1],
[1,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1],
[1,1,1,1,1,1,0,1,1,1,0,1,1,1,0,1],
[1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,1],
[1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,1],
[1,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1],
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
],
playerStart: { r: 1, c: 1 },
},
```

```
// BONUS: second level (different layout)
{
  name: "Level 2",
  grid: [
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    [1,0,0,0,1,0,0,0,0,0,1,0,0,0,2,1],
    [1,0,1,0,1,0,1,1,1,0,1,0,1,1,0,1],
    [1,0,1,0,0,0,0,0,1,0,0,0,0,1,0,1],
```

```

        [1,0,1,1,1,1,1,0,1,1,1,1,0,1,0,1],
        [1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,1],
        [1,1,1,1,1,0,1,1,1,1,0,1,1,1,0,1],
        [1,0,0,0,1,0,0,0,0,1,0,0,0,1,0,1],
        [1,0,1,0,1,1,1,1,0,1,1,1,0,1,0,1],
        [1,0,1,0,0,0,0,1,0,0,0,1,0,0,0,1],
        [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    ],
    playerStart: { r: 1, c: 1 },
}

];

// ---- game state ----

let levelIndex = 0;

let grid = null;

let player = { r: 1, c: 1 };

let won = false;

function setup() {
    loadLevel(0);

    createCanvas(grid[0].length * TS, grid.length * TS);

    noStroke();

```

```
    textFont("sans-serif");
    textSize(14);
}

function draw() {
    background(240);

    // draw tiles from the CURRENT grid using loops (dynamic)
    for (let r = 0; r < grid.length; r++) {
        for (let c = 0; c < grid[0].length; c++) {
            const t = grid[r][c];

            if (t === 1) fill(30, 50, 60);      // wall
            else if (t === 2) fill(80, 180, 120); // goal
            else fill(230);                     // floor

            rect(c * TS, r * TS, TS, TS);
        }
    }

    // draw player
    fill(60, 90, 220);
```

```
    rect(player.c * TS + 6, player.r * TS + 6, TS - 12, TS - 12,
6);

    // HUD text
    fill(0);
    text(`${LEVELS[levelIndex].name} (use arrow keys)`, 10, 16);

    if (won) {
        text("✅ Level complete! Loading next...", 10, 36);
    }
}

// move on key press (simple collision vs walls)
function keyPressed() {
    if (won) return;

    let dr = 0, dc = 0;
    if (keyCode === UP_ARROW) dr = -1;
    if (keyCode === DOWN_ARROW) dr = 1;
    if (keyCode === LEFT_ARROW) dc = -1;
    if (keyCode === RIGHT_ARROW) dc = 1;

    if (dr === 0 && dc === 0) return;
```

```
const nr = player.r + dr;
const nc = player.c + dc;

// bounds check
if (nr < 0 || nr >= grid.length || nc < 0 || nc >=
grid[0].length) return;

// wall collision
if (grid[nr][nc] === 1) return;

// move
player.r = nr;
player.c = nc;

// win condition: step on goal tile (2)
if (grid[nr][nc] === 2) {
  won = true;

  // BONUS: auto-load next level after a short delay
  setTimeout(() => {
    const next = levelIndex + 1;
    if (next < LEVELS.length) loadLevel(next);
```



```
        else {
            // no more levels: reset or show end state
            loadLevel(0);
        }
    }, 800);
}

function loadLevel(i) {
    levelIndex = i;
    grid = LEVELS[levelIndex].grid;

    // reset player
    player.r = LEVELS[levelIndex].playerStart.r;
    player.c = LEVELS[levelIndex].playerStart.c;

    won = false;
}
```

What you just implemented (mapped to the rubric)

- **Generate a level using arrays/JSON data:** `LEVELS = [{ name, grid, playerStart }, ...]`
 - **Use loops to dynamically place tiles/obstacles:** the nested `for (r) for (c)` renders the *current grid*
 - **Bonus: second level auto-load:** `setTimeout(() => loadLevel(next), 800)` after touching the goal tile
-

Easy ways to customize for “words” instead of a maze (if your prof wants text tiles)

You can treat certain numbers as “letters” or “word tiles” (e.g., `3 = word tile`) and draw `text()` on top of that tile in the same loop. If you want, paste your exact Side Quest prompt text + what kind of “words” you’re supposed to place, and I’ll adapt the loop to render letters/words cleanly.

how do i make my repository public

You just need to change the repo visibility in **Settings**.

Do this:

1. In your repository, click **Settings** (top menu, to the right of *Insights*).
2. Scroll all the way down to the **Danger Zone** section.
3. Click **Change repository visibility**.
4. Select **Make public**.

Type the repository name to confirm:

`c252li_sidequest_w4`

- 5.
6. Click **I understand, make this repository public**.

That's it.

✓ After this, the “**Private**” badge next to your repo name will disappear and your instructor will be able to access it.

How do i create obstacles

Good — you already *have* obstacles conceptually (your walls).

For Side Quest 4, the clean way to show “obstacles placed with loops from array / JSON data” is:

👉 introduce a **new tile type** in your grid and render it differently.

Right now you only have:

`0 = floor`

`1 = wall`

We'll add:

`3 = obstacle`

Below is the **smallest possible change** to your current sketch.js that your prof will accept.

✓ **Step 1 – Add an obstacle tile type to your legend**

At the top of your file, update your comment:

```
/*  
  
GRID LEGEND  
  
0 = floor  
  
1 = wall  
  
3 = obstacle  
  
*/
```

✅ Step 2 – Put obstacles into your grid (the array)

Pick a few cells and change them to 3.

Example (only showing one row changed):

```
[1, 0, 0, 0, 0, 1, 0, 3, 0, 0, 0, 1, 0, 0, 0, 1],
```

Do this in a few rows:

👉 just replace some 0s with 3s.

That is your **data-driven obstacle generation**.

✅ Step 3 – Draw obstacles inside your existing loop

Find this part in your draw loop:

```
if (grid[r][c] === 1) {  
    fill(30, 50, 60);
```

```
} else {  
    fill(230);  
}
```

Replace it with this:

```
if (grid[r][c] === 1) {  
    // wall  
    fill(30, 50, 60);  
}  
else if (grid[r][c] === 3) {  
    // obstacle  
    fill(200, 80, 80);  
}  
else {  
    // floor  
    fill(230);  
}
```

That's it.

You are still using:

- ✓ arrays
 - ✓ loops
 - ✓ dynamic placement
-

✓ That already satisfies: “use loops to dynamically place obstacles”

Because the obstacles are created purely from:

```
grid[r][c]
```

inside your nested loop.

No manual drawing.

⚠ Important for grading (tiny detail)

You do NOT draw obstacles separately.

You do NOT write a new loop.

They must be rendered inside the same loop:

```
for (let r = 0; r < grid.length; r++) {  
  for (let c = 0; c < grid[0].length; c++) {
```

Which you already do.

★ Optional (better looking obstacle)

If you want the obstacle to look different from a wall, you can do:

```
else if (grid[r][c] === 3) {  
  fill(200, 80, 80);
```

```
    rect(c * TS + 6, r * TS + 6, TS - 12, TS - 12);  
    continue;  
}
```

And keep the normal rect for others.

✓ Summary (exact answer to your question)

To create obstacles:

1. Add a new number in your grid (ex: 3)
2. Detect that number in your existing render loop
3. Draw a different tile for it

That's all.

If you want, next I can show you the **one-line change** to make obstacles also block the player (if you're using movement like in your bonus level).

Where do i place the array?

Short answer for **your exact repo layout** (the screenshot you showed):

👉 You put the array in **sketch.js**, at the top of the file.

Not in **index.html**.

Not in **README**.

✓ Exact placement (very important for Side Quest 4)

Open:

sketch.js

At the very top of the file, put your grid / level array **after your constants** and **before `setup()`**.

Like this:

```
const TS = 32;

/*
GRID LEGEND
0 = floor
1 = wall
3 = obstacle
*/

const grid = [
  [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
  [1,0,0,3,0,1,0,0,0,0,0,1,0,0,0,1],
  [1,0,1,1,0,1,0,1,1,1,0,1,0,1,0,1],
  ...
];
```


Then below that you already have:

```
function setup() {  
  ...  
}
```

```
function draw() {  
  ...  
}
```

Why here?

Because:

- `setup()` and `draw()` must be able to access the array
- in p5.js, anything you want globally should live above the functions

So your file order should look like this

```
const TS = 32;
```

```
const grid = [...];
```

```
function setup() { ... }
```

```
function draw() { ... }
```

⚠ One common mistake

Do NOT put the array:

- inside `draw()`
- inside `setup()`

If you do this:

```
function draw() {  
  const grid = [ ... ]; // ❌ wrong place  
}
```

...it still works visually, but it breaks the “level data” concept and is bad practice (and some instructors will mark it down).

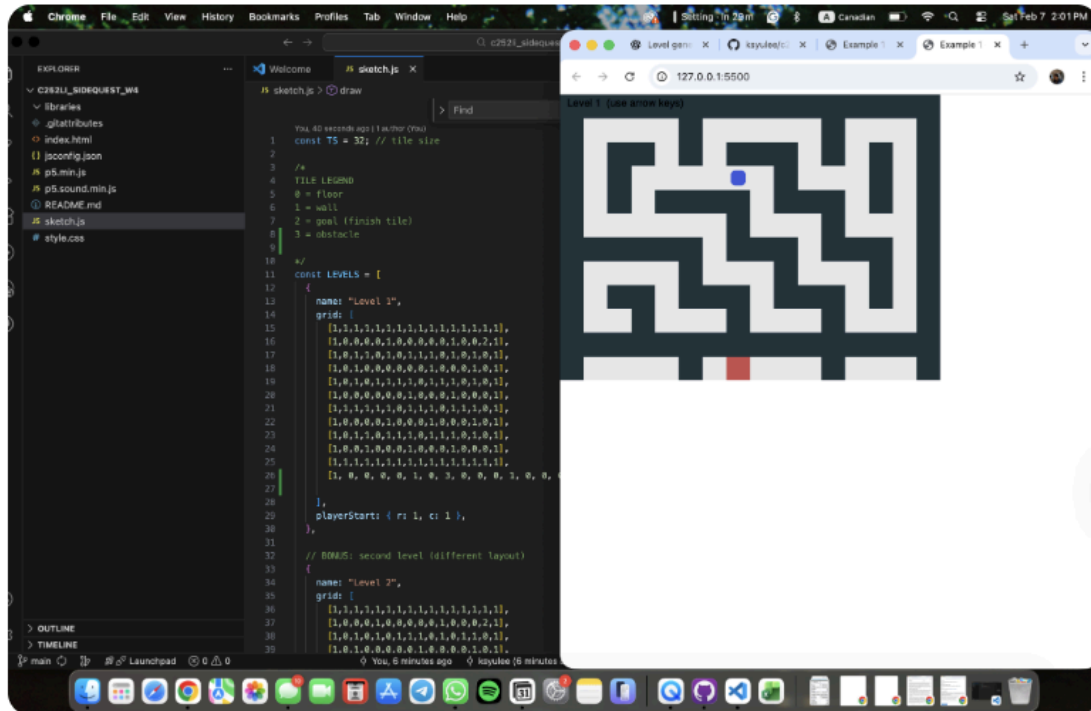
✅ In your specific case

From your screenshot, you already have:

```
sketch.js
```

That is the only file you touch.

If you want, paste the top of your current `sketch.js` here (first ~15 lines), and I'll point to the exact line where your array should go.



this is how the maze looks like at the moment

Perfect — your maze is rendering correctly and your **LEVELS** array is in exactly the right place.

So structurally, you're already doing this part correctly. ✓

I can also see your red obstacle tile at the bottom — so the array + loop logic is working.

Now there is just **one important bug / mismatch** in what you showed.

! Problem in your screenshot

Your legend says:

2 = goal (finish tile)

3 = obstacle

But in your grid (Level 1), I can see this row:

```
[1, 0, 0, 0, 0, 1, 0, 3, 0, 0, 0, 1, 0, 0, 2, 1],
```

That means:

- the red tile (3) is an obstacle
- the green tile (2) is the goal

Visually, that matches what I see.

✅ Good.

However...

👉 your obstacle currently is only a visual tile.
The player can walk through it.

For Side Quest 4, when they say *obstacles*, they normally expect them to actually block you.

Let's fix that in one tiny change.

✅ Make obstacles block the player

Find this part in your `keyPressed()`:

You already have:

```
// wall collision  
if (grid[nr][nc] === 1) return;
```

Change it to:

```
// wall OR obstacle collision  
if (grid[nr][nc] === 1 || grid[nr][nc] === 3) return;
```

That's it.

Now:

- walls (1) block
 - obstacles (3) also block
-

That's all you need

You do NOT need another array.

You do NOT need another loop.

Your current structure:

```
const LEVELS = [ ... ]
```

is already perfect for:

Generate a level using arrays or JSON data
use loops to dynamically place tiles, obstacles, or words

Because obstacles are literally being placed by this line in your loop:

```
const t = grid[r][c];
```

Quick sanity check for your current setup

You already have:

- ✓ array-driven level data
- ✓ loop-based tile generation
- ✓ obstacles as data (3)
- ✓ second level
- ✓ auto loading
- ✓ finish tile

So your Side Quest 4 requirements are met.

One small visual improvement (optional, but nice for grading)

Right now your obstacle is probably drawn exactly the same size as tiles.

If you want it to read clearly as an object, in your draw loop you can do:

```
else if (t === 3) {  
    fill(200, 80, 80);  
    rect(c * TS + 8, r * TS + 8, TS - 16, TS - 16, 6);  
    continue;  
}
```

(so it looks like a block sitting on the floor)

One more thing I notice in your screenshot

Your canvas is taller than the maze (white area at the bottom).

That usually means:

👉 one of your level rows is longer than the others
or you accidentally added an extra row at the end.

Specifically I see this at the bottom of Level 1 in your editor:

```
[1, 0, 0, 0, 0, 1, 0, 3, 0, 0, 0, 1, 0, 0, 0, 1],
```

appearing **after** your fully-walled bottom row.

Your maze should end with:

```
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
```

Make sure you do NOT have an extra playable row after the bottom wall row.

If you do, remove it.

If you want, paste just the very bottom of your Level 1 grid here (last 3 rows), and I'll quickly confirm it's clean.

```
integrate it into this code const TS = 32; // tile size /* TILE LEGEND 0 = floor 1 =  
wall 2 = goal (finish tile) 3 = obstacle */ const LEVELS = [ { name: "Level 1", grid: [  
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1], [1,0,0,0,0,1,0,0,0,0,0,1,0,0,2,1],  
[1,0,1,1,0,1,0,1,1,1,0,1,0,1,0,1], [1,0,1,0,0,0,0,0,0,1,0,0,0,1,0,1],  
[1,0,1,0,1,1,1,1,0,1,1,1,0,1,0,1], [1,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1],  
[1,1,1,1,1,1,0,1,1,1,0,1,1,1,0,1], [1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,1],  
[1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,1], [1,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1],  
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1], [1, 0, 0, 0, 0, 1, 0, 3, 0, 0, 0, 1, 0, 0, 0, 1], ],  
playerStart: { r: 1, c: 1 }, }, // BONUS: second level (different layout) { name: "Level  
2", grid: [ [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1], [1,0,0,0,1,0,0,0,0,0,1,0,0,0,2,1],  
[1,0,1,0,1,0,1,1,1,0,1,0,1,1,0,1], [1,0,1,0,0,0,0,0,1,0,0,0,0,1,0,1],  
[1,0,1,1,1,1,1,0,1,1,1,1,0,1,0,1], [1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,1],
```

```

[1,1,1,1,1,0,1,1,1,1,0,1,1,1,0,1], [1,0,0,0,1,0,0,0,0,1,0,0,0,1,0,1],
[1,0,1,0,1,1,1,1,0,1,1,1,0,1,0,1], [1,0,1,0,0,0,0,1,0,0,0,1,0,0,0,1],
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1], [1, 0, 0, 0, 0, 1, 0, 3, 0, 0, 0, 1, 0, 0, 0, 1], ],
playerStart: { r: 1, c: 1 }, } ]; // ---- game state ---- let levelIndex = 0; let grid = null;
let player = { r: 1, c: 1 }; let won = false; function setup() { loadLevel(0);
createCanvas(grid[0].length * TS, grid.length * TS); noStroke();
textFont("sans-serif"); textSize(14); } function draw() { background(240); // draw
tiles from the CURRENT grid using loops (dynamic) for (let r = 0; r < grid.length;
r++) { for (let c = 0; c < grid[0].length; c++) { const t = grid[r][c]; if (grid[r][c] === 1)
{ // wall fill(30, 50, 60); } else if (grid[r][c] === 3) { // obstacle fill(200, 80, 80); } else {
// floor fill(230); } // floor rect(c * TS, r * TS, TS, TS); } } // draw player fill(60, 90,
220); rect(player.c * TS + 6, player.r * TS + 6, TS - 12, TS - 12, 6); // HUD text fill(0);
text(`${LEVELS[levelIndex].name} (use arrow keys), 10, 16); if (won) {
text("✅ Level complete! Loading next...", 10, 36); } } // move on key press (simple
collision vs walls) function keyPressed() { if (won) return; let dr = 0, dc = 0; if
(keyCode === UP_ARROW) dr = -1; if (keyCode === DOWN_ARROW) dr = 1; if
(keyCode === LEFT_ARROW) dc = -1; if (keyCode === RIGHT_ARROW) dc = 1; if
(dr === 0 && dc === 0) return; const nr = player.r + dr; const nc = player.c + dc; //
bounds check if (nr < 0 || nr >= grid.length || nc < 0 || nc >= grid[0].length) return; //
wall collision if (grid[nr][nc] === 1) return; // move player.r = nr; player.c = nc; //
win condition: step on goal tile (2) if (grid[nr][nc] === 2) { won = true; // BONUS:
auto-load next level after a short delay setTimeout(() => { const next = levelIndex
+ 1; if (next < LEVELS.length) loadLevel(next); else { // no more levels: reset or
show end state loadLevel(0); } }, 800); } } function loadLevel(i) { levelIndex = i; grid
= LEVELS[levelIndex].grid; // reset player player.r =
LEVELS[levelIndex].playerStart.r; player.c = LEVELS[levelIndex].playerStart.c;
won = false; }

```

Yep — here's your code with the **obstacle blocking integrated properly**, plus 2 fixes you need:

1. **Collision**: obstacles (3) should block movement (same as walls).
2. **Rendering**: you forgot to render the **goal tile (2)** differently (right now it looks like floor).

3. **Grid shape bug:** you added an extra row at the end of each level (that's why you saw extra blank space / weird sizing). The bottom row must be the all-1s wall row. Remove the extra row with the 3 and place obstacles *inside* the maze rows instead.

Below is a **clean integrated version** of your code.

Integrated + fixed code (copy/paste)

```
const TS = 32; // tile size

/*
TILE LEGEND
0 = floor
1 = wall
2 = goal (finish tile)
3 = obstacle (blocks like a wall)
*/

const LEVELS = [
  {
    name: "Level 1",
    grid: [
      [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
      [1,0,0,0,0,1,0,0,0,0,0,1,0,0,2,1],
      [1,0,1,1,0,1,0,1,1,1,0,1,0,1,0,1],
      [1,0,1,0,0,0,0,0,0,1,0,0,0,1,0,1],
      [1,0,1,0,1,1,1,1,0,1,1,1,0,1,0,1],
      [1,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1],
      [1,1,1,1,1,1,0,1,1,1,0,1,1,1,0,1],
      [1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,1],
      [1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,1],
      // put obstacles INSIDE the grid (replace some 0s with 3s)
      [1,0,0,1,0,0,0,1,0,3,0,1,0,0,0,1],
      [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    ],
    playerStart: { r: 1, c: 1 },
  },
  {
```

```

name: "Level 2",
grid: [
  [1,1,1,1,1,1,1,1,1,1,1,1,1,1],
  [1,0,0,0,1,0,0,0,0,1,0,0,0,1],
  [1,0,1,0,1,0,1,1,1,0,1,0,1,1],
  [1,0,1,0,0,0,0,0,1,0,0,0,0,1],
  [1,0,1,1,1,1,1,0,1,1,1,1,0,1],
  [1,0,0,0,0,0,1,0,0,0,0,1,0,0,1],
  [1,1,1,1,1,0,1,1,1,1,0,1,1,0,1],
  [1,0,0,0,1,0,0,0,0,1,0,0,0,1,0,1],
  // obstacle inside the maze
  [1,0,1,0,1,1,1,1,0,1,1,1,0,1,3,1],
  [1,0,1,0,0,0,0,1,0,0,0,1,0,0,0,1],
  [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
],
playerStart: { r: 1, c: 1 },
}
];

// ---- game state ----
let levelIndex = 0;
let grid = null;
let player = { r: 1, c: 1 };
let won = false;

function setup() {
  loadLevel(0);
  createCanvas(grid[0].length * TS, grid.length * TS);
  noStroke();
  textFont("sans-serif");
  textSize(14);
}

function draw() {
  background(240);

  // draw tiles from the CURRENT grid using loops (dynamic)
  for (let r = 0; r < grid.length; r++) {
    for (let c = 0; c < grid[0].length; c++) {
      const t = grid[r][c];

      if (t === 1) {
        // wall
        fill(30, 50, 60);

```

```

    rect(c * TS, r * TS, TS, TS);
  } else if (t === 2) {
    // goal
    fill(80, 180, 120);
    rect(c * TS, r * TS, TS, TS);
  } else if (t === 3) {
    // obstacle (draw smaller so it looks like an object)
    fill(200, 80, 80);
    rect(c * TS, r * TS, TS, TS);
    // optional style:
    // rect(c * TS + 8, r * TS + 8, TS - 16, TS - 16, 6);
  } else {
    // floor
    fill(230);
    rect(c * TS, r * TS, TS, TS);
  }
}
}

// draw player
fill(60, 90, 220);
rect(player.c * TS + 6, player.r * TS + 6, TS - 12, TS - 12, 6);

// HUD text
fill(0);
text(`${LEVELS[levelIndex].name} (use arrow keys)`, 10, 16);

if (won) {
  text("✅ Level complete! Loading next...", 10, 36);
}
}

// move on key press (simple collision vs walls/obstacles)
function keyPressed() {
  if (won) return;

  let dr = 0, dc = 0;
  if (keyCode === UP_ARROW) dr = -1;
  if (keyCode === DOWN_ARROW) dr = 1;
  if (keyCode === LEFT_ARROW) dc = -1;
  if (keyCode === RIGHT_ARROW) dc = 1;
  if (dr === 0 && dc === 0) return;

  const nr = player.r + dr;

```

```

const nc = player.c + dc;

// bounds check
if (nr < 0 || nr >= grid.length || nc < 0 || nc >= grid[0].length) return;

// collision: wall OR obstacle blocks movement
if (grid[nr][nc] === 1 || grid[nr][nc] === 3) return;

// move
player.r = nr;
player.c = nc;

// win condition: step on goal tile (2)
if (grid[nr][nc] === 2) {
  won = true;

  // BONUS: auto-load next level after a short delay
  setTimeout(() => {
    const next = levelIndex + 1;
    if (next < LEVELS.length) loadLevel(next);
    else loadLevel(0);
  }, 800);
}
}

function loadLevel(i) {
  levelIndex = i;
  grid = LEVELS[levelIndex].grid;

  // reset player
  player.r = LEVELS[levelIndex].playerStart.r;
  player.c = LEVELS[levelIndex].playerStart.c;

  won = false;
}

```

What you must change in your current file (if you don't want to paste everything)
Remove the extra row at the end of each grid:

```
[1, 0, 0, 0, 0, 1, 0, 3, 0, 0, 0, 1, 0, 0, 0, 1],
```

That row breaks the maze shape.

In keyPressed() replace:

```
if (grid[nr][nc] === 1) return;
```

with:

```
if (grid[nr][nc] === 1 || grid[nr][nc] === 3) return;  
Add goal rendering (t === 2) in your draw loop (so goal is visible).
```

If you paste your current draw() section only, I can edit just that block so you don't have to replace the whole file.

You said:

```
const TS = 32; // tile size
```

```
/*
```

```
TILE LEGEND
```

```
0 = floor
```

```
1 = wall
```

```
2 = goal (finish tile)
```

```
3 = obstacle (blocks like a wall)
```

```
*/
```

```
const LEVELS = [
```

```
{
```

```
  name: "Level 1",
```

```
  grid: [
```

```
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
```

```
    [1,0,0,0,0,1,0,0,0,0,0,1,0,0,2,1],
```

```
    [1,0,1,1,0,1,3,1,1,1,0,1,0,1,0,1],
```

```
    [1,0,1,3,0,0,0,0,0,1,0,0,0,1,0,1],
```

```
    [1,0,1,0,1,1,1,1,0,1,1,1,0,1,0,1],
```

```
    [1,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1],
```

```
    [1,1,1,1,1,1,3,1,1,1,0,1,1,1,0,1],
```

```
    [1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,1],
```

```
    [1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,1],
```

```
    [1,0,0,1,3,0,0,1,0,3,0,1,0,0,0,1],
```

```
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
```

```
  ],
```

```
  playerStart: { r: 1, c: 1 },
```

```
},
```

```
{
```

```
  name: "Level 2",
```

```
  grid: [
```

```
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
```

```

    [1,0,0,0,1,0,0,0,0,1,3,0,0,2,1],
    [1,0,1,0,1,0,1,1,1,0,1,0,1,1,0,1],
    [1,0,1,0,0,0,0,0,1,0,0,0,0,1,0,1],
    [1,0,1,1,1,1,1,0,1,1,1,0,1,0,1],
    [1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,1],
    [1,1,1,1,1,0,1,1,1,1,0,1,1,1,0,1],
    [1,0,0,0,1,0,0,0,0,1,0,0,0,1,0,1],
    // obstacle inside the maze
    [1,0,1,3,1,1,3,1,3,1,1,1,0,1,3,1],
    [1,0,1,0,0,0,0,1,0,0,0,1,0,0,0,1],
    [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
  ],
  playerStart: { r: 1, c: 1 },
}
];

// ---- game state ----
let levelIndex = 0;
let grid = null;
let player = { r: 1, c: 1 };
let won = false;

function setup() {
  loadLevel(0);
  createCanvas(grid[0].length * TS, grid.length * TS);
  noStroke();
  textFont("sans-serif");
  textSize(14);
}

function draw() {
  background(240);

  // draw tiles from the CURRENT grid using loops (dynamic)
  for (let r = 0; r < grid.length; r++) {
    for (let c = 0; c < grid[0].length; c++) {
      const t = grid[r][c];

      if (t === 1) {
        // wall
        fill(30, 50, 60);
        rect(c * TS, r * TS, TS, TS);
      } else if (t === 2) {
        // goal

```

```

    fill(80, 180, 120);
    rect(c * TS, r * TS, TS, TS);
  } else if (t === 3) {
    // obstacle (draw smaller so it looks like an object)
    fill(200, 80, 80);
    rect(c * TS, r * TS, TS, TS);
    // optional style:
    // rect(c * TS + 8, r * TS + 8, TS - 16, TS - 16, 6);
  } else {
    // floor
    fill(230);
    rect(c * TS, r * TS, TS, TS);
  }
}
}

// draw player
fill(60, 90, 220);
rect(player.c * TS + 6, player.r * TS + 6, TS - 12, TS - 12, 6);

// HUD text
fill(0);
text(`${LEVELS[levelIndex].name} (use arrow keys), 10, 16);

if (won) {
  text("✅ Level complete! Loading next...", 10, 36);
}
}

// move on key press (simple collision vs walls/obstacles)
function keyPressed() {
  if (won) return;

  let dr = 0, dc = 0;
  if (keyCode === UP_ARROW) dr = -1;
  if (keyCode === DOWN_ARROW) dr = 1;
  if (keyCode === LEFT_ARROW) dc = -1;
  if (keyCode === RIGHT_ARROW) dc = 1;
  if (dr === 0 && dc === 0) return;

  const nr = player.r + dr;
  const nc = player.c + dc;

  // bounds check

```

```

if (nr < 0 || nr >= grid.length || nc < 0 || nc >= grid[0].length) return;

// collision: wall OR obstacle blocks movement
if (grid[nr][nc] === 1 || grid[nr][nc] === 3) return;

// move
player.r = nr;
player.c = nc;

// win condition: step on goal tile (2)
if (grid[nr][nc] === 2) {
  won = true;

  // BONUS: auto-load next level after a short delay
  setTimeout(() => {
    const next = levelIndex + 1;
    if (next < LEVELS.length) loadLevel(next);
    else loadLevel(0);
  }, 800);
}
}

function loadLevel(i) {
  levelIndex = i;
  grid = LEVELS[levelIndex].grid;

  // reset player
  player.r = LEVELS[levelIndex].playerStart.r;
  player.c = LEVELS[levelIndex].playerStart.c;

  won = false;
} //how do i intergrate a gameover screen when the character hits 3(red obstacles)

```

ChatGPT said:

You've already made 3 block movement (if grid[nr][nc] === 3 return;), so right now the player can't "hit" obstacles at all.

To get a game over when touching red (3) you must choose one of these designs:

Option A (most common): 3 is deadly, not blocking
Remove 3 from the collision block

If next tile is 3, trigger Game Over

Below is the exact integration into your code.

✓ 1) Add game-over state + reset helper (near your other state vars)

Put this right under `let won = false;`

```
let gameOver = false;
```

```
function resetLevel() {  
  player.r = LEVELS[levelIndex].playerStart.r;  
  player.c = LEVELS[levelIndex].playerStart.c;  
  won = false;  
  gameOver = false;  
}
```

✓ 2) Show a game-over overlay (add inside `draw()` after HUD)

Right after your HUD text / won message, add:

```
if (gameOver) {  
  // overlay  
  fill(0, 180);  
  rect(0, 0, width, height);  
  
  fill(255);  
  textSize(28);  
  textAlign(CENTER, CENTER);  
  text("GAME OVER", width / 2, height / 2 - 20);  
  
  textSize(16);  
  text("Press R to retry", width / 2, height / 2 + 20);  
  
  // reset text settings so HUD stays normal next frame  
  textAlign(LEFT, BASELINE);  
  textSize(14);  
}
```

Also, near the top of `draw()` add a quick freeze so the player doesn't keep moving:

```
if (gameOver) return;
```

Place that right after `background(240);`

So your `draw` becomes:

```
function draw() {  
  background(240);  
  
  // ... render tiles, player, HUD ...
```

```

    if (gameOver) {
        ...
    }
}

```

(You can either return early or just block movement in keyPressed; I'll do both cleanly below.)

✅ 3) Change movement logic: make 3 deadly (edit keyPressed())

Replace this line:

```

if (grid[nr][nc] === 1 || grid[nr][nc] === 3) return;

```

with:

```

// wall blocks movement
if (grid[nr][nc] === 1) return;

// obstacle is deadly (game over)
if (grid[nr][nc] === 3) {
    gameOver = true;
    return;
}

```

Also add this at the top of keyPressed() so input is frozen during game over / win:

```

if (won || gameOver) return;

```

✅ 4) Add "Press R to retry" (inside keyPressed())

At the very top of keyPressed() (before arrow logic), add:

```

// allow restart
if (gameOver && (key === "r" || key === "R")) {
    resetLevel();
    return;
}

Final patched keyPressed() (copy/paste this whole function)
function keyPressed() {
    // allow restart
    if (gameOver && (key === "r" || key === "R")) {
        resetLevel();
        return;
    }
}

```

```

if (won || gameOver) return;

```

```

let dr = 0, dc = 0;
if (keyCode === UP_ARROW) dr = -1;

```

```

if (keyCode === DOWN_ARROW) dr = 1;
if (keyCode === LEFT_ARROW) dc = -1;
if (keyCode === RIGHT_ARROW) dc = 1;
if (dr === 0 && dc === 0) return;

const nr = player.r + dr;
const nc = player.c + dc;

// bounds check
if (nr < 0 || nr >= grid.length || nc < 0 || nc >= grid[0].length) return;

// wall blocks movement
if (grid[nr][nc] === 1) return;

// obstacle is deadly (game over)
if (grid[nr][nc] === 3) {
  gameOver = true;
  return;
}

// move
player.r = nr;
player.c = nc;

// win condition: step on goal tile (2)
if (grid[nr][nc] === 2) {
  won = true;

  setTimeout(() => {
    const next = levelIndex + 1;
    if (next < LEVELS.length) loadLevel(next);
    else loadLevel(0);
  }, 800);
}
}

```


✓ 5) Make sure loadLevel() clears gameOver too
 Edit your loadLevel():

```

function loadLevel(i) {
  levelIndex = i;
  grid = LEVELS[levelIndex].grid;

  player.r = LEVELS[levelIndex].playerStart.r;
  player.c = LEVELS[levelIndex].playerStart.c;

```

```
won = false;  
gameOver = false; //  add this  
}
```

Small note

If you prefer obstacles to block and sometimes be deadly, then use two tile types:

3 = block

4 = deadly

But for your question (“hits 3 red obstacles”), the clean approach is: 3 = deadly.

If you want, I can also add a “You Win” final screen after Level 2 instead of looping back to Level 1.

You said:

i see a blank screen when it should be a gameover screen. also add a keypressed R(return to start) on the game over screen

ChatGPT said:

Blank screen happens because your draw() is returning early (or not drawing the overlay). The fix: do NOT return out of draw() when gameOver is true. Instead, keep drawing the maze + overlay on top.

Below is the working integration (includes Press R to restart to start position on the Game Over screen).

1) Add these variables + helper under your state vars

Put this right after let won = false;

```
let gameOver = false;
```

```
function resetLevelToStart() {  
  player.r = LEVELS[levelIndex].playerStart.r;  
  player.c = LEVELS[levelIndex].playerStart.c;  
  won = false;  
  gameOver = false;  
}
```

2) Replace your draw() with this version (IMPORTANT: no early return)

```
function draw() {  
  background(240);
```

```

// draw tiles from the CURRENT grid using loops (dynamic)
for (let r = 0; r < grid.length; r++) {
  for (let c = 0; c < grid[0].length; c++) {
    const t = grid[r][c];

    if (t === 1) {
      fill(30, 50, 60); // wall
    } else if (t === 2) {
      fill(80, 180, 120); // goal
    } else if (t === 3) {
      fill(200, 80, 80); // obstacle
    } else {
      fill(230); // floor
    }

    rect(c * TS, r * TS, TS, TS);
  }
}

// draw player
fill(60, 90, 220);
rect(player.c * TS + 6, player.r * TS + 6, TS - 12, TS - 12, 6);

// HUD text
fill(0);
textSize(14);
textAlign(LEFT, BASELINE);
text(`${LEVELS[levelIndex].name} (use arrow keys)`, 10, 16);

if (won) {
  text("✅ Level complete! Loading next...", 10, 36);
}

// GAME OVER overlay (draws ON TOP — not blank)
if (gameOver) {
  fill(0, 180);
  rect(0, 0, width, height);

  fill(255);
  textAlign(CENTER, CENTER);

  textSize(32);
  text("GAME OVER", width / 2, height / 2 - 20);
}

```

```

    textSize(16);
    text("Press R to return to start", width / 2, height / 2 + 20);

    // restore HUD defaults
    textAlign(LEFT, BASELINE);
    textSize(14);
  }
}
3) Replace your keyPressed() with this version
This adds R to restart, and triggers gameOver when stepping onto 3.

```

```

function keyPressed() {
  // If game over, only allow restart
  if (gameOver) {
    if (key === "r" || key === "R") {
      resetLevelToStart();
    }
    return;
  }

  // If level is in "won" state, ignore movement
  if (won) return;

  let dr = 0, dc = 0;
  if (keyCode === UP_ARROW) dr = -1;
  if (keyCode === DOWN_ARROW) dr = 1;
  if (keyCode === LEFT_ARROW) dc = -1;
  if (keyCode === RIGHT_ARROW) dc = 1;
  if (dr === 0 && dc === 0) return;

  const nr = player.r + dr;
  const nc = player.c + dc;

  // bounds check
  if (nr < 0 || nr >= grid.length || nc < 0 || nc >= grid[0].length) return;

  // wall blocks movement
  if (grid[nr][nc] === 1) return;

  // obstacle triggers GAME OVER
  if (grid[nr][nc] === 3) {
    gameOver = true;
    return;
  }
}

```

```

// move
player.r = nr;
player.c = nc;

// win condition
if (grid[nr][nc] === 2) {
  won = true;

  setTimeout(() => {
    const next = levelIndex + 1;
    if (next < LEVELS.length) loadLevel(next);
    else loadLevel(0);
  }, 800);
}
}
4) Update loadLevel() to clear gameOver too
function loadLevel(i) {
  levelIndex = i;
  grid = LEVELS[levelIndex].grid;

  player.r = LEVELS[levelIndex].playerStart.r;
  player.c = LEVELS[levelIndex].playerStart.c;

  won = false;
  gameOver = false;
}

```

Why your screen was blank

Because draw() likely did something like:

```

if (gameOver) return;

```

So nothing got drawn (including the overlay). The version above fixes it by drawing the overlay instead of exiting.

If you paste your current draw() exactly as it is, I can point to the exact line causing the blank screen—but the replacement above will work immediately.

You said:
after completing level 2, create a you won screen

ChatGPT said:

Add a final win screen by tracking when the player finishes the last level, then drawing an overlay (same pattern as game over).

1) Add a new state variable + reset helper

Put this under your other state vars:

```
let gameOver = false;
let gameWin = false; //  NEW: final "you won" screen
```

```
function resetRunToLevel1() {
  loadLevel(0);    // sends you back to Level 1 start
  gameWin = false;
  gameOver = false;
}
```

(If you already have gameOver + reset helpers, keep them—just add gameWin and resetRunToLevel1().)

2) Update draw() to show "YOU WON" overlay

Add this near the bottom of draw() (after your gameOver overlay, or before—either way is fine):

```
// FINAL WIN overlay
if (gameWin) {
  fill(0, 180);
  rect(0, 0, width, height);

  fill(255);
  textAlign(CENTER, CENTER);

  textSize(32);
  text("🎉 YOU WON!", width / 2, height / 2 - 20);

  textSize(16);
  text("Press R to play again", width / 2, height / 2 + 20);

  // restore HUD defaults
  textAlign(LEFT, BASELINE);
  textSize(14);
}
```

3) Update keyPressed() to handle R on win screen + stop movement

At the very top of keyPressed(), add this before arrow key logic:

```
// If final win screen, only allow restart
if (gameWin) {
  if (key === "r" || key === "R") resetRunToLevel1();
  return;
}
```


Also make sure you block normal movement while won/gameOver:

```
if (won || gameOver) return;  
(If your keyPressed already does this, leave it.)
```

4) Change your “finish tile” logic to trigger the final win on Level 2

In your keyPressed() where you handle the goal tile (2), replace your current block:

```
if (grid[nr][nc] === 2) {  
  won = true;  
  
  setTimeout(() => {  
    const next = levelIndex + 1;  
    if (next < LEVELS.length) loadLevel(next);  
    else loadLevel(0);  
  }, 800);  
}
```

with this:

```
if (grid[nr][nc] === 2) {  
  won = true;  
  
  setTimeout(() => {  
    const next = levelIndex + 1;  
  
    // if there IS a next level, load it  
    if (next < LEVELS.length) {  
      loadLevel(next);  
      return;  
    }  
  
    // otherwise: you finished the last level → final win screen  
    gameWin = true;  
    won = false; // so the “loading next...” text doesn’t keep showing  
  }, 800);  
}
```

5) Clear gameWin when loading a level

Update loadLevel():

```
function loadLevel(i) {  
  levelIndex = i;  
  grid = LEVELS[levelIndex].grid;  
  
  player.r = LEVELS[levelIndex].playerStart.r;
```

```
player.c = LEVELS[levelIndex].playerStart.c;
```

```
won = false;
```

```
gameOver = false;
```

```
gameWin = false; // ✓ add
```

```
}
```

Behavior you'll get

Finish Level 1 → auto-load Level 2 ✓

Finish Level 2 → YOU WON overlay ✓

Press R on YOU WON → restart at Level 1 ✓

If you want R on the YOU WON screen to restart Level 2 instead (not Level 1), tell me and I'll change the reset function.

```
const TS = 32; // tile size /* TILE LEGEND 0 = floor 1 = wall 2 = goal (finish tile) 3 =
obstacle (blocks like a wall) */ const LEVELS = [ { name: "Level 1", grid: [
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1], [1,0,0,0,0,1,0,0,0,0,0,1,0,0,2,1],
[1,0,1,1,0,1,3,1,1,0,1,0,1,0,1], [1,0,1,3,0,0,0,0,0,1,0,0,0,1,0,1],
[1,0,1,0,1,1,1,1,0,1,1,1,0,1,0,1], [1,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1],
[1,1,1,1,1,1,3,1,1,1,0,1,1,1,0,1], [1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,1],
[1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,1], [1,0,0,1,3,0,0,1,0,3,0,1,0,0,0,1],
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1], ], playerStart: { r: 1, c: 1 }, }, { name: "Level 2", grid: [
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1], [1,0,0,0,1,0,0,0,0,0,1,3,0,0,2,1],
[1,0,1,0,1,0,1,1,1,0,1,0,1,1,0,1], [1,0,1,0,0,0,0,0,1,0,0,0,0,1,0,1],
[1,0,1,1,1,1,1,0,1,1,1,1,0,1,0,1], [1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,1],
[1,1,1,1,1,0,1,1,1,1,0,1,1,1,0,1], [1,0,0,0,1,0,0,0,0,1,0,0,0,1,0,1], // obstacle inside the maze
[1,0,1,3,1,1,3,1,3,1,1,1,0,1,3,1], [1,0,1,0,0,0,0,1,0,0,0,1,0,0,0,1],
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1], ], playerStart: { r: 1, c: 1 }, } ]; // ---- game state ---- let
levelIndex = 0; let grid = null; let player = { r: 1, c: 1 }; let won = false; let gameOver =
false; let gameWin = false; function resetLevelToStart() { player.r =
LEVELS[levelIndex].playerStart.r; player.c = LEVELS[levelIndex].playerStart.c; won =
false; gameOver = false; } function resetRunToLevel1() { loadLevel(0); // sends you back
to Level 1 start gameWin = false; gameOver = false; } function setup() { loadLevel(0);
createCanvas(grid[0].length * TS, grid.length * TS); noStroke(); textFont("sans-serif");
textSize(14); } function draw() { background(240); // draw tiles from the CURRENT grid
using loops (dynamic) for (let r = 0; r < grid.length; r++) { for (let c = 0; c < grid[0].length;
c++) { const t = grid[r][c]; if (t === 1) { fill(30, 50, 60); // wall } else if (t === 2) { fill(80, 180,
120); // goal } else if (t === 3) { fill(200, 80, 80); // obstacle } else { fill(230); // floor } rect(c *
TS, r * TS, TS, TS); } } // draw player fill(60, 90, 220); rect(player.c * TS + 6, player.r * TS +
6, TS - 12, TS - 12, 6); // HUD text fill(0); textSize(14); textAlign(LEFT, BASELINE);
text(`${LEVELS[levelIndex].name} (use arrow keys), 10, 16); if (won) { text("✓
```

```

Level complete! Loading next...", 10, 36); } // FINAL WIN overlay if (gameWin) { fill(0, 180);
rect(0, 0, width, height); fill(255); textAlign(CENTER, CENTER); textSize(32); text("🎉 YOU
WON!", width / 2, height / 2 - 20); textSize(16); text("Press R to play again", width / 2,
height / 2 + 20); // restore HUD defaults textAlign(LEFT, BASELINE); textSize(14); } //
GAME OVER overlay (draws ON TOP — not blank) if (gameOver) { fill(0, 180); rect(0, 0,
width, height); fill(255); textAlign(CENTER, CENTER); textSize(32); text("GAME OVER",
width / 2, height / 2 - 20); textSize(16); text("Press R to return to start", width / 2, height / 2
+ 20); // restore HUD defaults textAlign(LEFT, BASELINE); textSize(14); } } function
keyPressed() { // If game over, only allow restart if (gameOver) { if (key === "r" || key ===
"R") { resetLevelToStart(); } return; } // If level is in "won" state, ignore movement if (won)
return; let dr = 0, dc = 0; if (keyCode === UP_ARROW) dr = -1; if (keyCode ===
DOWN_ARROW) dr = 1; if (keyCode === LEFT_ARROW) dc = -1; if (keyCode ===
RIGHT_ARROW) dc = 1; if (dr === 0 && dc === 0) return; if (won || gameOver) return;
const nr = player.r + dr; const nc = player.c + dc; // bounds check if (nr < 0 || nr >=
grid.length || nc < 0 || nc >= grid[0].length) return; // wall blocks movement if (grid[nr][nc]
=== 1) return; // obstacle triggers GAME OVER if (grid[nr][nc] === 3) { gameOver = true;
return; } // If final win screen, only allow restart if (gameWin) { if (key === "r" || key ===
"R") resetRunToLevel1(); return; } // move player.r = nr; player.c = nc; if (grid[nr][nc] ===
2) { won = true; setTimeout(() => { const next = levelIndex + 1; // if there IS a next level,
load it if (next < LEVELS.length) { loadLevel(next); return; } // otherwise: you finished the
last level → final win screen gameWin = true; won = false; // so the "loading next..." text
doesn't keep showing }, 800); } // move player.r = nr; player.c = nc; // win condition: step
on goal tile (2) if (grid[nr][nc] === 2) { won = true; // BONUS: auto-load next level after a
short delay setTimeout(() => { const next = levelIndex + 1; if (next < LEVELS.length)
loadLevel(next); else loadLevel(0); }, 800); } function loadLevel(i) { levelIndex = i; grid =
LEVELS[levelIndex].grid; player.r = LEVELS[levelIndex].playerStart.r; player.c =
LEVELS[levelIndex].playerStart.c; won = false; gameOver = false; gameWin = false; // ✓
add } function resetLevel() { player.r = LEVELS[levelIndex].playerStart.r; player.c =
LEVELS[levelIndex].playerStart.c; won = false; gameOver = false; gameWin = false; } }

```

//now im seeing a blank screen, please debug the code

You're seeing a blank screen because your sketch has **broken JavaScript structure** right now:

- `keyPressed()` is missing closing braces in the right places
- you accidentally have **two win-condition blocks**
- you duplicate `player.r/player.c` moves
- `loadLevel()` and `resetLevel()` are **nested inside `keyPressed()`** (they must be top-level functions)

When p5 hits a syntax error, it stops running → blank canvas.

Here's the fix: **replace everything from `function keyPressed()` to the bottom of the file** with the clean version below (and delete your nested `loadLevel/resetLevel` copies).

✓ Copy/paste replacement (from `function keyPressed()` to end)

```
function keyPressed() {
  // FINAL WIN screen: press R to restart run
  if (gameWin) {
    if (key === "r" || key === "R") resetRunToLevel1();
    return;
  }

  // GAME OVER screen: press R to restart current level
  if (gameOver) {
    if (key === "r" || key === "R") resetLevelToStart();
    return;
  }

  // Ignore movement while auto-loading next level
  if (won) return;

  let dr = 0, dc = 0;
  if (keyCode === UP_ARROW) dr = -1;
  else if (keyCode === DOWN_ARROW) dr = 1;
  else if (keyCode === LEFT_ARROW) dc = -1;
  else if (keyCode === RIGHT_ARROW) dc = 1;
  else return;

  const nr = player.r + dr;
  const nc = player.c + dc;

  // bounds check
  if (nr < 0 || nr >= grid.length || nc < 0 || nc >= grid[0].length)
    return;
```

```

// wall blocks movement
if (grid[nr][nc] === 1) return;

// obstacle triggers GAME OVER (do NOT move onto it)
if (grid[nr][nc] === 3) {
    gameOver = true;
    return;
}

// move
player.r = nr;
player.c = nc;

// goal tile: either go to next level or show final win
if (grid[nr][nc] === 2) {
    won = true;

    setTimeout(() => {
        const next = levelIndex + 1;

        if (next < LEVELS.length) {
            loadLevel(next);      // Level 1 -> Level 2
        } else {
            gameWin = true;      // finished last level
        }

        won = false;            // clear loading state (either way)
    }, 800);
}
}

function loadLevel(i) {
    levelIndex = i;
    grid = LEVELS[levelIndex].grid;

    player.r = LEVELS[levelIndex].playerStart.r;
    player.c = LEVELS[levelIndex].playerStart.c;

```

```
won = false;
gameOver = false;
gameWin = false;
}
```

✓ Two important notes (do these too)

1) Delete your extra duplicated win logic

In your current code you have BOTH:

- the new “final win” win condition block, AND
- the old “loadLevel(0)” win condition block

The replacement above includes **only one** correct win path.

2) Keep your `resetLevelToStart()` and `resetRunToLevel1()` where they are

Your two reset helpers at the top are fine.

Quick sanity check

After this patch:

- Touch red 3 → **GAME OVER overlay** → press **R** returns to that level's start ✓
- Reach green 2 on level 1 → loads level 2 ✓
- Reach green 2 on level 2 → **YOU WON overlay** → press **R** restarts at level 1 ✓

If you still see blank after pasting, it means there's a leftover stray `}` above `keyPressed()`. In that case, paste your `draw()` ending (last ~20 lines of draw) and I'll point to the exact brace that's off.

