

Process & Decision Documentation

Side Quests and A4 (Individual Work)

Keep this section brief, typically 2 to 4 sentences.

Focus on:

- One significant decision or change you made
- Why you made it
- What effect it had on the work

Examples:

- Simplifying a mechanic so it functioned correctly
- Changing an approach after something failed
- Deciding not to pursue an idea due to time or technical limitations

You are not expected to document every alternative or iteration

Entry Header

Name: Abiha Shah

Role(s):the whole thing

Primary responsibility for this work:everything

Goal of Work Session

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

It was to create levels on the grid and make two levels, making the second one tougher

Tools, Resources, or Inputs Used

- GenAI tools
 - o chatGPT

GenAI Documentation

Everyone must complete this section. If not GenAI was used, write, “No GenAI used for this task.” When GenAI is not used, process evidence should still demonstrate iteration, revision, or development over time.

Because GenAI can closely mimic human-created work, instructors or TAs may occasionally request additional process evidence to confirm non-use. This may include original working files (e.g., an illustrator file), intermediate drafts, or a brief check-in with a TA to walk through your process.

These requests are not an assumption of misconduct. They are part of ensuring academic integrity in an environment where distinguishing between human-created and AI-generated work is increasingly difficult.

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

Date Used: 10-02-2026

Tool Disclosure: ChatGPT 5.2

Purpose of Use: To help me with this project

Summary of Interaction: I helped me with the levels

Human Decision Point(s): AI was giving wrong info at a point and kept on changing things and I had to constantly correct them

Integrity & Verification Note: By playing it on live and seeing how it would turn out

Scope of GenAI Use: The level of creativity and changes I had to put in myself

Limitations or Misfires: It had to be constantly corrected

Summary of Process (Human + Tool)

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

- Iteration between drafts or versions
- Discussion with teammates
- Testing and revision
- Moments of uncertainty, failure, or rework

Decision Points & Trade-offs

I had to make sure the creativity and the levels were aligned to what we had to do for this side quest

Verification & Judgement

- I had to constantly see what AI would provide me with and had to play it on live to see if it's what I needed for this side quest

Limitations, Dead Ends, or Open Questions

- Did not work as expected initially and had to be constantly corrected.

Appendix

Please include a full transcript of your conversation with the GenAI. GenAI transcripts are used for transparency and verification only and are not assessed for writing quality or completeness.

Note: If GenAI is used in another language, you must include both the original language and a translated transcript in the appendix

Can you help me understand how to 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

Here's a clean example (single **HTML file**) that:

- Defines levels as **arrays/JSON**
- Uses **loops** to place **tiles, obstacles, and words**
- Loads **Level 2 automatically** when Level 1 is finished (reach the goal)

Copy-paste into `index.html` and open in a browser.

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Array/JSON Level Loader (2 Levels)</title>
```

```

<style>
  body { font-family: system-ui, Arial; display: grid;
place-items: center; height: 100vh; margin: 0; }
  #wrap { width: 520px; }
  canvas { border: 2px solid #111; background: #fafafa;
display: block; }
  .row { display: flex; justify-content: space-between;
align-items: center; margin: 10px 0 0; }
  #msg { font-size: 14px; color: #333; }
  #words { font-size: 14px; color: #111; }
  kbd { padding: 2px 6px; border: 1px solid #aaa; border-
bottom-width: 2px; border-radius: 6px; background: #fff; }
</style>
</head>
<body>
  <div id="wrap">
    <canvas id="c" width="520" height="520"></canvas>
    <div class="row">
      <div id="msg">Move with <kbd>↑</kbd><kbd>↓</
kbd><kbd>←</kbd><kbd>→</kbd>. Collect words. Reach the
green goal.</div>
    </div>
    <div class="row">
      <div id="words">Words collected: <span
id="wordList">(none)</span></div>
      <div id="level">Level: 1</div>
    </div>
  </div>

  <script>
    // -----
    // 1) LEVEL DATA (arrays/JSON)
    // -----
    // Legend:
    // # = wall
    // . = floor
    // P = player start
    // G = goal (finish level)
    // X = obstacle (acts like wall)
    // w = word tile (collectible)

```

```

const LEVELS = [
  {
    name: "Level 1",
    tileSize: 40,
    map: [
      "#####",
      "#P....w.....#...#",
      "#.#####.###..#.#G#",
      "#...X...#.....#.#.#",
      "###.###.#####.###",
      "#.....#.#.....#.#",
      "#.###.#.#####.###",
      "#...#.#.....w...#.#",
      "#.#.#.#####.###",
      "#.#.....#",
      "#####",
    ],
    wordText: ["coffee", "retro"] // collected in order
as you pick up w tiles
  },
  {
    name: "Level 2",
    tileSize: 40,
    map: [
      "#####",
      "#P..w.....#.....#",
      "#.#####.#.###.###",
      "#.....X...#...#.#",
      "#.#####.###",
      "#.....w.....#G#",
      "#.#####.###",
      "#.....#",
      "#####",
    ],
    wordText: ["beans", "bold"]
  }
];

// -----
// 2) GAME STATE

```

```

// -----
const canvas = document.getElementById("c");
const ctx = canvas.getContext("2d");
const levelLabel = document.getElementById("level");
const wordListEl = document.getElementById("wordList");

let levelIndex = 0;
let grid = []; // 2D array of chars
let player = { x: 0, y: 0 };
let goal = { x: 0, y: 0 };
let collectedWords = [];
let wordQueue = []; // words assigned to each 'w'
pickup in that level

function loadLevel(i) {
  levelIndex = i;

  const level = LEVELS[levelIndex];
  levelLabel.textContent = `Level: ${levelIndex + 1}`;

  // convert string rows -> char grid (2D array)
  grid = level.map.map(row => row.split(""));

  collectedWords = [];
  wordQueue = [...level.wordText];
  wordListEl.textContent = "(none)";

  // find player/goal positions by looping the grid
  for (let y = 0; y < grid.length; y++) {
    for (let x = 0; x < grid[y].length; x++) {
      if (grid[y][x] === "P") {
        player.x = x; player.y = y;
        grid[y][x] = "."; // replace start with floor
      }
      if (grid[y][x] === "G") {
        goal.x = x; goal.y = y;
      }
    }
  }
}

```

```

    // resize canvas to fit current map
    const ts = level.tileSize;
    canvas.width = grid[0].length * ts;
    canvas.height = grid.length * ts;

    draw();
}

// -----
// 3) RENDERING (loops place tiles)
// -----
function draw() {
    const level = LEVELS[levelIndex];
    const ts = level.tileSize;

    ctx.clearRect(0, 0, canvas.width, canvas.height);

    for (let y = 0; y < grid.length; y++) {
        for (let x = 0; x < grid[y].length; x++) {
            const cell = grid[y][x];

            // base floor
            ctx.fillStyle = "#f6f6f6";
            ctx.fillRect(x * ts, y * ts, ts, ts);

            // draw walls/obstacles/goal/words based on tile
type
            if (cell === "#" ) {
                ctx.fillStyle = "#222";
                ctx.fillRect(x * ts, y * ts, ts, ts);
            } else if (cell === "X") {
                ctx.fillStyle = "#7a1f1f";
                ctx.fillRect(x * ts, y * ts, ts, ts);
            } else if (cell === "w") {
                ctx.fillStyle = "#ffd54a";
                ctx.beginPath();
                ctx.arc(x * ts + ts/2, y * ts + ts/2, ts *
0.28, 0, Math.PI * 2);
                ctx.fill();
            } else if (cell === "G") {

```

```

        ctx.fillStyle = "#2aa44f";
        ctx.fillRect(x * ts, y * ts, ts, ts);
    }

    // grid lines (optional)
    ctx.strokeStyle = "rgba(0,0,0,0.06)";
    ctx.strokeRect(x * ts, y * ts, ts, ts);
}

// player
ctx.fillStyle = "#2b6cff";
ctx.beginPath();
ctx.arc(player.x * ts + ts/2, player.y * ts + ts/2,
ts * 0.33, 0, Math.PI * 2);
ctx.fill();
}

// -----
// 4) MOVEMENT + COLLISIONS
// -----
function isBlocked(x, y) {
    const cell = grid[y]?.[x];
    return cell === "#" || cell === "x" || cell ===
undefined;
}

function tryMove(dx, dy) {
    const nx = player.x + dx;
    const ny = player.y + dy;

    if (isBlocked(nx, ny)) return;

    // move
    player.x = nx;
    player.y = ny;

    // pickup words
    if (grid[ny][nx] === "w") {
        const nextWord = wordQueue.shift() ?? "word";

```



```

        collectedWords.push(nextWord);
        grid[ny][nx] = ".";

        wordListEl.textContent = collectedWords.join(", ");
    }

    // check finish
    if (grid[ny][nx] === "G") {
        // Bonus: auto-load next level
        if (levelIndex < LEVELS.length - 1) {
            loadLevel(levelIndex + 1);
            return;
        } else {
            alert("You finished all levels 🎉");
        }
    }

    draw();
}

window.addEventListener("keydown", (e) => {
    if (e.key === "ArrowUp") tryMove(0, -1);
    if (e.key === "ArrowDown") tryMove(0, 1);
    if (e.key === "ArrowLeft") tryMove(-1, 0);
    if (e.key === "ArrowRight") tryMove(1, 0);
});

// start
loadLevel(0);
</script>
</body>
</html>

```

If you tell me what environment your assignment uses (p5.js? Phaser? Unity? Godot? Scratch-style?), I can adapt this same “JSON + loops + level switch” pattern to that exactly.

change this based on the instructions /* Week 4 — Example 1: Grid + Static Maze Course: GBDA302 Instructors: Dr. Karen Cochrane and David Han Date: Feb. 5, 2026 PURPOSE: This is the simplest possible p5.js sketch that demonstrates: 1. How a 2D array represents a maze/ game level 2. Nested loops to iterate through grid rows/columns 3. Converting grid coordinates (r,c) → screen coordinates (x,y) 4. Tile-based rendering (every cell = one rectangle) */ 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, 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); }

```

```
/*
```

Week 4 – Example 2: Grid + Dynamic Level Loading (Arrays/JSON)

Course: GBDA302

Instructors: Dr. Karen Cochrane and David Han

Date: Feb. 5, 2026

PURPOSE: This p5.js sketch demonstrates:

- 1) Levels stored as arrays/JSON data (not just one hardcoded grid)
- 2) Nested loops to render a grid (rows/columns)
- 3) Converting grid coordinates (r,c) → screen coordinates (x,y)
- 4) Dynamically placing tiles + obstacles + words using loops
- 5) BONUS: A second level loads automatically when Level 1 is finished

```

*/

const TS = 32; // TILE SIZE (pixels per grid cell)

// GRID LEGEND (how characters map to visuals + rules)
// "." = floor (walkable)
// "#" = wall (blocked)
// "X" = obstacle (blocked)
// "P" = player start
// "G" = goal (finish level)
// "w" = word pickup (collectible)
const LEVELS = [
  {
    name: "Level 1",
    // Each string is one row of the level map
    map: [
      "#####",
      "#P....w....#...#",
      "#.#####.###.#.#G#",
      "#...X...#...#.#.#",
      "###.###.#.###.#.#",
      "#.....#.#.....#.#",
      "#.###.#.#####.#.#",
      "#...#.#...w...#.#",
      "#.#.#.#####",
      "#.....#",
      "#####",
    ],
    // The words you collect in this level (assigned in
    order to each 'w')
    words: ["coffee", "retro"]
  },
  {
    name: "Level 2",
    map: [
      "#####",
      "#P..w.....#.....#",
      "#.#####.#.##.#",
      "#.....X...#..#.#",
      "#.#####.#.#",
    ]
  }
]

```

```

        "#.....w.....#G#",
        "#.#####.#",
        "#.....#",
        "#####"
    ],
    words: ["beans", "bold"]
}
];

// -----
// GAME STATE (changes over time)
// -----
let levelIndex = 0;
let grid = []; // 2D array of chars (built from
level.map)
let player = { r: 0, c: 0 };
let goal = { r: 0, c: 0 };

let wordQueue = []; // copy of current level's words
let collected = []; // collected words this level

// -----
// LEVEL LOADING (dynamic)
// -----
function loadLevel(i) {
    levelIndex = i;
    const level = LEVELS[levelIndex];

    // Convert string rows → 2D char grid using loops
    grid = [];
    for (let r = 0; r < level.map.length; r++) {
        grid[r] = level.map[r].split("");
    }

    // reset collectibles + queue
    collected = [];
    wordQueue = [...level.words];

    // Find player start + goal positions by looping the grid
    for (let r = 0; r < grid.length; r++) {

```

```

    for (let c = 0; c < grid[r].length; c++) {
      if (grid[r][c] === "P") {
        player.r = r;
        player.c = c;
        grid[r][c] = "."; // replace start with floor
      }
      if (grid[r][c] === "G") {
        goal.r = r;
        goal.c = c;
      }
    }
  }
}

// Canvas size updates automatically based on the current
level
createCanvas(grid[0].length * TS, grid.length * TS);
}

// -----
// p5.js SETUP
// -----
function setup() {
  textFont("sans-serif");
  textSize(14);
  noStroke();

  // Start on level 1
  loadLevel(0);
}

// -----
// p5.js DRAW (game loop)
// -----
function draw() {
  background(240);

  /*
  CORE RENDER LOOP:
  Nested loops:
  - outer: rows

```

```

- inner: columns
This is where we dynamically place tiles/obstacles/words.
*/
for (let r = 0; r < grid.length; r++) {
  for (let c = 0; c < grid[r].length; c++) {
    const cell = grid[r][c];

    // Base floor
    fill(230);
    rect(c * TS, r * TS, TS, TS);

    // Walls / obstacles / goal / words
    if (cell === "#") {
      fill(30, 50, 60);          // wall
      rect(c * TS, r * TS, TS, TS);
    } else if (cell === "X") {
      fill(120, 35, 35);          // obstacle
      rect(c * TS, r * TS, TS, TS);
    } else if (cell === "G") {
      fill(40, 170, 80);          // goal
      rect(c * TS, r * TS, TS, TS);
    } else if (cell === "w") {
      fill(255, 210, 70);          // word pickup
      circle(c * TS + TS / 2, r * TS + TS / 2, TS *
0.55);
    }
  }
}

// Draw player
fill(40, 110, 255);
circle(player.c * TS + TS / 2, player.r * TS + TS / 2, TS
* 0.65);

// HUD
fill(0);
text(`Level: ${levelIndex + 1} ($
{LEVELS[levelIndex].name})`, 10, 16);
text(`Words: ${collected.length ? collected.join(", ") :
"(none)"}`, 10, 34);

```

```

    text("Arrow keys: move | Collect yellow | Reach green
goal", 10, 52);
}

// -----
// MOVEMENT + COLLISION
// -----
function keyPressed() {
    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) attemptMove(dr, dc);
}

function isBlocked(r, c) {
    // outside grid counts as blocked
    if (r < 0 || r >= grid.length || c < 0 || c >=
grid[0].length) return true;

    const cell = grid[r][c];
    return cell === "#" || cell === "X";
}

function attemptMove(dr, dc) {
    const nr = player.r + dr;
    const nc = player.c + dc;

    if (isBlocked(nr, nc)) return;

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

    // If you step on a word tile, collect it
    if (grid[nr][nc] === "w") {
        const next = wordQueue.shift() || "word";
        collected.push(next);
    }
}

```

```

    grid[nr][nc] = ".";
  }

  // If you reach the goal, auto-load next level (BONUS)
  if (grid[nr][nc] === "G") {
    if (levelIndex < LEVELS.length - 1) {
      loadLevel(levelIndex + 1);
    } else {
      // finished last level
      // (kept simple so it matches class expectations)
      console.log("All levels finished!");
    }
  }
}

```

what am i supposed to change in this

You don't need to "rewrite everything" — you just need to change your **static single grid** into **level data** (array/JSON) and then use loops to **load + draw** that data, plus (bonus) **switch levels**.

Here's exactly what changes, using your file as the baseline:

1) Replace **const grid = [...]** with **const LEVELS = [...]**

Right now you have **one** hardcoded grid. The instructions want **levels stored in arrays/JSON**, like:

- `LEVELS[0].map` = Level 1 grid
- `LEVELS[1].map` = Level 2 grid (bonus)

So you remove:

- `const grid = [...]`

and add:

- `const LEVELS = [{ map: [...] }, { map: [...] }]`

2) Add a **loadLevel()** function

Your current sketch never "loads" anything — it just draws `grid`.

You need a function that:

- takes a level index
- converts its map into a grid (2D array)
- finds player start / goal
- resets collected words

This is what makes it “generated from arrays/JSON data”.

3) Change **setup()** so it calls **loadLevel(0)**

Instead of:

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

You do:

```
loadLevel(0);
```

because the canvas size comes from whatever level is loaded.

4) Keep your nested loops in **draw()**, but draw more than walls/floor

Your current loops only check:

- 1 = wall
- 0 = floor

To meet the instructions, your loop should also place:

- obstacles (like "X")
- words/pickups (like "w")
- goal (like "G")

That's the “dynamically place tiles, obstacles, or words” part.

5) Bonus: Add logic that loads Level 2 automatically

You need a “finish condition”, typically:





- player reaches goal tile ("G")

When that happens:

```
loadLevel(levelIndex + 1);
```

So... what are you *supposed* to change?

Minimum changes to satisfy the instructions:

-  Swap the single `grid` for a `LEVELS` array (JSON-style data)
-  Add `loadLevel()` to build the grid from that data
-  Use the nested loops to draw walls/floor + at least one of: obstacles/words
-  Add a check to auto-load Level 2 when Level 1 is finished (bonus)

If you want, paste your assignment’s exact wording (or rubric line) and I’ll tell you which of these are required vs bonus in *your* marking scheme.

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.

You’re supposed to change your sketch so it **isn’t one hard-coded grid anymore**. Instead:

- store your level as **data** (arrays/JSON)
- use **loops** to draw/place everything from that data
- (bonus) **switch to level 2** when level 1 is finished

Below is a **drop-in p5.js version** that matches those instructions *directly* (levels as JSON-ish objects, loops to render tiles + obstacles + words, and auto-loads level 2 when you reach the goal).




Copy/paste this whole sketch

```
/*
```

GBDA302 — Week 4

Example: Levels from Arrays/JSON + Loops + Auto Level Switch

Meets instructions:

- 1) Generate a level using arrays/JSON data  (LEVELS)
 - 2) Use loops to dynamically place tiles/obstacles/words 
(nested for-loops in draw)
 - 3) Bonus: Level 2 loads when Level 1 is finished  (reach 'G')
- */

```
const TS = 32;
```

```
// Tile legend (characters)
```

```
const TILES = {  
  FLOOR: ".",  
  WALL: "#",  
  OBSTACLE: "X",  
  WORD: "w",  
  PLAYER: "P",  
  GOAL: "G",  
};
```

```
// 1) LEVELS STORED AS JSON-LIKE DATA (array of objects)
```

```
const LEVELS = [  
  {  
    name: "Level 1",  
    map: [  
      "#####",  
      "#P...w...#...#",  
      "#.#####.###.#.#G#",  
      "#...X...#...#.#.#",  
      "###.###.#.###.#.#",  
      "#.....#.#.....#.#",  
      "#.###.#.#####.#.#",  
      "#...#.#...w...#.#",  
      "#.#.#.#####",  
      "#.....#",  
    ],  
  },  
];
```

```

        "#####"
    ],
    words: ["coffee", "retro"], // assigned in order to
    each 'w' you pick up
  },
  {
    name: "Level 2",
    map: [
      "#####",
      "#P..w.....#....#",
      "#.#####.#.##.#",
      "#.....X...#..#.#",
      "#.#####.#.##",
      "#.....w....#G#",
      "#.#####.#",
      "#.....#",
      "#####",
    ],
    words: ["beans", "bold"],
  },
];

```

```

// GAME STATE
let levelIndex = 0;
let grid = []; // 2D array of chars
let player = { r: 0, c: 0 };
let collected = [];
let wordQueue = [];

// 2) LOAD LEVEL: converts JSON data → grid + finds player
function loadLevel(i) {
  levelIndex = i;

  const level = LEVELS[levelIndex];

  // build 2D grid from strings
  grid = [];
  for (let r = 0; r < level.map.length; r++) {
    grid[r] = level.map[r].split("");
  }
}

```

```

// reset words
collected = [];
wordQueue = [...level.words];

// find player start and replace with floor
for (let r = 0; r < grid.length; r++) {
  for (let c = 0; c < grid[r].length; c++) {
    if (grid[r][c] === TILES.PLAYER) {
      player.r = r;
      player.c = c;
      grid[r][c] = TILES.FLOOR;
    }
  }
}

// canvas size based on level data
createCanvas(grid[0].length * TS, grid.length * TS);
}

function setup() {
  textFont("sans-serif");
  textSize(14);
  noStroke();
  loadLevel(0);
}

function draw() {
  background(240);

  // 3) LOOPS dynamically place tiles/obstacles/words
  (render everything from grid data)
  for (let r = 0; r < grid.length; r++) {
    for (let c = 0; c < grid[r].length; c++) {
      const cell = grid[r][c];

      // base floor
      fill(230);
      rect(c * TS, r * TS, TS, TS);
    }
  }
}

```

```

        // draw based on tile type
        if (cell === TILES.WALL) {
            fill(30, 50, 60);
            rect(c * TS, r * TS, TS, TS);
        } else if (cell === TILES.OBSTACLE) {
            fill(120, 35, 35);
            rect(c * TS, r * TS, TS, TS);
        } else if (cell === TILES.WORD) {
            fill(255, 210, 70);
            circle(c * TS + TS / 2, r * TS + TS / 2, TS *
0.55);
        } else if (cell === TILES.GOAL) {
            fill(40, 170, 80);
            rect(c * TS, r * TS, TS, TS);
        }
    }
}

// player
fill(40, 110, 255);
circle(player.c * TS + TS / 2, player.r * TS + TS / 2, TS
* 0.65);

// HUD
fill(0);
text(`Level: ${levelIndex + 1} - $
{LEVELS[levelIndex].name}`, 10, 16);
text(`Words: ${collected.length ? collected.join(", ") :
"(none)"}`, 10, 34);
text("Arrows move | Yellow = word | Green = goal", 10,
52);
}

// movement + collision
function keyPressed() {
    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 || dc) attemptMove(dr, dc);
}

function blocked(r, c) {
    if (r < 0 || r >= grid.length || c < 0 || c >=
grid[0].length) return true;
    return grid[r][c] === TILES.WALL || grid[r][c] ===
TILES.OBSTACLE;
}

function attemptMove(dr, dc) {
    const nr = player.r + dr;
    const nc = player.c + dc;

    if (blocked(nr, nc)) return;

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

    // collect words
    if (grid[nr][nc] === TILES.WORD) {
        collected.push(wordQueue.shift() ?? "word");
        grid[nr][nc] = TILES.FLOOR;
    }

    // BONUS: auto-load next level when goal reached
    if (grid[nr][nc] === TILES.GOAL) {
        if (levelIndex < LEVELS.length - 1) {
            loadLevel(levelIndex + 1);
        } else {
            console.log("Finished all levels!");
        }
    }
}

```

What this satisfies (mapped to your instruction lines)

- **“Generate a level using arrays or JSON data”** → LEVELS is an array of objects holding map + words.

- “use loops to dynamically place tiles, obstacles, or words” → nested for loops in `draw() render #, X, w, G`.
- “Bonus: Add a second level that loads automatically when the first is finished” → reaching G calls `loadLevel(levelIndex + 1)`.

If you want it to stay closer to your original **0/1 numeric grid**, I can convert this to `0=floor, 1=wall, 2=obstacle, 3=word, 4=goal` — but the logic stays exactly the same.

it should have the same format as the one I gave u

```
/*
Week 4 — Example 1 (UPDATED): Grid + Dynamic Levels
(Arrays/JSON) + Auto Level Switch
Course: GBDA302
Instructors: Dr. Karen Cochran and David Han
Date: Feb. 5, 2026

PURPOSE: Same “format” as the original, but now it meets
the instructions:
1. Generate a level using arrays or JSON data (we store TWO
levels in LEVELS[])
2. Nested loops to iterate through grid rows/columns
3. Converting grid coordinates (r,c) → screen coordinates
(x,y)
4. Tile-based rendering (every cell = one rectangle)
5. Loops dynamically place tiles, obstacles, and words
(tile types 0–4)
BONUS: Level 2 loads automatically when Level 1 is finished
(reach the goal tile)
*/

const TS = 32; // TILE SIZE: pixels per grid cell (32x32
squares)

/*
GRID LEGEND (how numbers map to visuals + rules):
- 0 = floor (walkable, light gray)
```



```

- 1 = wall (blocked, dark teal)
- 2 = obstacle (blocked, dark red)
- 3 = word pickup (collectible, yellow)
- 4 = goal (finish tile, green)
*/

/*
LEVEL DATA (ARRAYS / JSON-STYLE OBJECTS):
Instead of ONE static "grid", we store multiple levels
inside LEVELS.
Each level is its own 2D array.
*/
const LEVELS = [
  {
    name: "Level 1",
    grid: [
      // 16 columns × 11 rows (same shape as your original)
      [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
      [1, 0, 0, 0, 0, 1, 0, 0, 3, 0, 0, 1, 0, 0, 0, 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, 3, 1],
      [1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1],
      [1, 0, 0, 1, 0, 0, 2, 1, 0, 0, 0, 1, 0, 0, 4, 1],
      [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    ],
  },
  {
    name: "Level 2",
    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],
      [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, 3, 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, 2, 1, 0, 0, 0, 1, 0, 0, 4, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    ],
},
];

/*
ACTIVE LEVEL STATE:
Instead of drawing one grid forever, we keep track of the
current level index
and swap grids when a level is completed.
*/
let levelIndex = 0;
let grid = []; // this will point to the CURRENT level's
grid

/*
PLAYER (simple so the level can actually be "finished"):
- player moves with arrow keys
- cannot walk through walls (1) or obstacles (2)
- can collect words (3)
- finishes level by stepping on goal tile (4)
*/
let playerR = 1;
let playerC = 1;

/*
LOAD LEVEL: Copies the level grid + resets player position.
(Using JSON/array data to "generate" the level.)
*/
function loadLevel(i) {
    levelIndex = i;

    // IMPORTANT: copy the grid so we can change word tiles
    (3 → 0) when collected
    const source = LEVELS[levelIndex].grid;
    grid = [];
    for (let r = 0; r < source.length; r++) {

```

```

    grid[r] = [...source[r]];
}

// Simple player spawn (top-left open area)
playerR = 1;
playerC = 1;

// Canvas size = grid dimensions * tile size (same logic
as your original)
createCanvas(grid[0].length * TS, grid.length * TS);
}

/*
p5.js SETUP: Runs once when sketch loads
*/
function setup() {
  noStroke();
  textFont("sans-serif");
  textSize(14);

  // Start at Level 1
  loadLevel(0);
}

/*
p5.js DRAW: Runs 60 times per second (game loop)
*/
function draw() {
  background(240);

  /*
  CORE RENDERING LOOP: Draw every tile in the CURRENT level
  grid

  Nested loops:
  - Outer loop: iterate ROWS (r)
  - Inner loop: iterate COLUMNS (c)
  */
  for (let r = 0; r < grid.length; r++) {
    for (let c = 0; c < grid[0].length; c++) {

```

```

const tile = grid[r][c];

// Choose fill color based on tile type
if (tile === 1) {
    // WALL
    fill(30, 50, 60);
} else if (tile === 2) {
    // OBSTACLE
    fill(120, 35, 35);
} else if (tile === 3) {
    // WORD PICKUP
    fill(255, 210, 70);
} else if (tile === 4) {
    // GOAL
    fill(40, 170, 80);
} else {
    // FLOOR
    fill(230);
}

/*
CONVERT GRID COORDS → SCREEN COORDS:
x = column × TS
y = row × TS
*/
rect(c * TS, r * TS, TS, TS);

// Optional: tiny label for word tile (keeps it
obvious)
if (tile === 3) {
    fill(0);
    text("w", c * TS + 12, r * TS + 20);
}
}

// Draw the player (on top of tiles)
fill(40, 110, 255);
rect(playerC * TS + 6, playerR * TS + 6, TS - 12, TS -
12);

```

```

    // UI LABELS (same vibe as your original)
    fill(0);
    text(`Level: ${levelIndex + 1} ($
{LEVELS[levelIndex].name})`, 10, 16);
    text("Arrow keys to move. Yellow = word. Green = goal.",
10, 34);
}

/*
INPUT: Move player with arrow keys
(Needed so a level can be "finished" and switch to the next
level.)
*/
function keyPressed() {
    let dr = 0;
    let 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) attemptMove(dr, dc);
}

/*
MOVEMENT RULES:
- Blocked if wall (1) or obstacle (2)
- Collect word if tile (3) (turn it into floor)
- Finish level if tile (4) → auto load next level (BONUS)
*/
function attemptMove(dr, dc) {
    const nr = playerR + dr;
    const nc = playerC + dc;

    // Bounds check (treat outside as blocked)
    if (nr < 0 || nr >= grid.length || nc < 0 || nc >=
grid[0].length) return;

```

```

const nextTile = grid[nr][nc];

// Collision: walls + obstacles block movement
if (nextTile === 1 || nextTile === 2) return;

// Move player
playerR = nr;
playerC = nc;

// Collect word
if (nextTile === 3) {
  grid[nr][nc] = 0; // remove pickup (now it's floor)
}

// Finish level (BONUS: auto load level 2)
if (nextTile === 4) {
  if (levelIndex < LEVELS.length - 1) {
    loadLevel(levelIndex + 1);
  } else {
    // If they reach the goal on the last level
    console.log("All levels complete!");
  }
}
}

```

This keeps the **same structure/style** as what you gave (big header comment, **TS**, grid legend, `setup()`, `draw()` with nested loops, tile rendering), but now it satisfies your instructions because:

- the level comes from **LEVELS data** (arrays/JSON style)
- the nested loops **dynamically place** walls/obstacles/words/goal
- Level 2 **loads automatically** when you reach the goal in Level 1

If you want, I can swap the “finish condition” from **reach goal** to **collect all words** (also super common for assignments).

make it more simple

```
/*
```

Week 4 – Simple Example: 2 Levels + Arrays + Loops

Course: GBDA302

Instructors: Dr. Karen Cochrane and David Han

Date: Feb. 5, 2026

PURPOSE (matches the instructions):

1. Generate a level using arrays/JSON data (LEVELS)

2. Use nested loops to draw tiles + obstacles + words

BONUS: Level 2 loads automatically when Level 1 is finished
(reach the goal)

*/

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

```
/*
```

```
GRID LEGEND (numbers):
```

```
0 = floor
```

```
1 = wall
```

```
2 = obstacle
```

```
3 = word
```

```
4 = goal
```

```
*/
```

```
// LEVEL DATA (arrays/JSON-style)
```

```
const LEVELS = [
```

```
{
```

```
  grid: [
```

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

```
    [1,0,0,0,3,0,0,0,4,1],
```

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

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

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

```
  ],
```

```
},
```

```
{
```

```
  grid: [
```

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

```
    [1,0,3,0,0,0,1,0,4,1],
```

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

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

```

        [1,1,1,1,1,1,1,1,1,1],
    ],
},
];

// current level + player
let levelIndex = 0;
let grid = [];
let pr = 1; // player row
let pc = 1; // player col

function loadLevel(i) {
    levelIndex = i;

    // copy level grid so we can edit it (remove words)
    const src = LEVELS[levelIndex].grid;
    grid = [];
    for (let r = 0; r < src.length; r++) {
        grid[r] = [...src[r]];
    }

    // reset player
    pr = 1;
    pc = 1;

    // canvas matches grid size
    createCanvas(grid[0].length * TS, grid.length * TS);
    noStroke();
}

function setup() {
    loadLevel(0);
}

function draw() {
    background(240);

    // LOOP: draw tiles from the array (dynamic placement)
    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(120, 35, 35);    // obstacle
    else if (t === 3) fill(255, 210, 70);   // word
    else if (t === 4) fill(40, 170, 80);    // goal
    else fill(230);                          // floor

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

// player
fill(40, 110, 255);
rect(pc * TS + 6, pr * TS + 6, TS - 12, TS - 12);

// tiny label
fill(0);
textSize(14);
text("Level " + (levelIndex + 1), 10, 16);
}

function keyPressed() {
  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;

  const nr = pr + dr;
  const nc = pc + dc;

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

  const next = grid[nr][nc];

  // blocked by wall or obstacle
  if (next === 1 || next === 2) return;

```

```

// move
pr = nr;
pc = nc;

// collect word (turn into floor)
if (next === 3) grid[nr][nc] = 0;





// BONUS: finish level (goal) -> load next
if (next === 4 && levelIndex < LEVELS.length - 1) {
    loadLevel(levelIndex + 1);
}
}

```

make it the same dimension i gave u in teh start

Perfect — we'll keep it **16 columns × 11 rows**, exactly like your original grid.

Here is the **simplified version**, same dimensions, same structure style, but now:

-  Uses LEVELS array (JSON-style data)
-  Uses nested loops to draw tiles dynamically
-  Has obstacles + words + goal
-  BONUS: Level 2 loads automatically

/*

Week 4 — Simple Example: Same Dimensions (16x11) + Dynamic Levels

Course: GBDA302

Instructors: Dr. Karen Cochrane and David Han

Date: Feb. 5, 2026

PURPOSE:

1. Generate level using arrays/JSON data (LEVELS)
 2. Use loops to dynamically place tiles, obstacles, words
- BONUS: Level 2 loads automatically when Level 1 is finished

```

*/

const TS = 32;

/*
GRID LEGEND:
0 = floor
1 = wall
2 = obstacle
3 = word
4 = goal
*/

// 16 columns × 11 rows (SAME size as original)
const LEVELS = [
    {
        grid: [
            [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
            [1,0,0,0,0,1,0,0,3,0,0,1,0,0,4,1],
            [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],
            [1,0,0,0,0,1,0,0,0,1,0,0,0,1,3,1],
            [1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,1],
            [1,0,0,1,0,0,2,1,0,0,0,1,0,0,0,1],
            [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
        ],
    },
    {
        grid: [
            [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
            [1,0,3,0,0,1,0,0,0,0,0,1,0,0,4,1],
            [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],
            [1,0,0,0,0,1,0,0,0,1,0,0,0,1,3,1],
        ],
    }
];

```

```

        [1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,1],
        [1,0,0,1,0,0,2,1,0,0,0,1,0,0,0,1],
        [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    ],
},
];

let levelIndex = 0;
let grid = [];
let pr = 1; // player row
let pc = 1; // player col

function loadLevel(i) {
    levelIndex = i;

    // copy grid so we can edit it
    const src = LEVELS[levelIndex].grid;
    grid = [];
    for (let r = 0; r < src.length; r++) {
        grid[r] = [...src[r]];
    }

    pr = 1;
    pc = 1;

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

function setup() {
    loadLevel(0);
}

function draw() {
    background(240);

    // NESTED LOOPS (dynamic tile placement)
    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(120, 35, 35); // obstacle
        else if (t === 3) fill(255, 210, 70); // word
        else if (t === 4) fill(40, 170, 80); // goal
        else fill(230);                          // floor

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

// player
fill(40, 110, 255);
rect(pc * TS + 6, pr * TS + 6, TS - 12, TS - 12);

fill(0);
textSize(14);
text("Level " + (levelIndex + 1), 10, 16);
}

function keyPressed() {
    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;

    const nr = pr + dr;
    const nc = pc + dc;

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

    const next = grid[nr][nc];

    if (next === 1 || next === 2) return; // blocked

    pr = nr;
    pc = nc;

```

```

    if (next === 3) grid[nr][nc] = 0; // collect word

    // BONUS: load next level when goal reached
    if (next === 4 && levelIndex < LEVELS.length - 1) {
        loadLevel(levelIndex + 1);
    }
}

```

This keeps:

- **16 columns**
- **11 rows**
- Same tile size
- Same structure style as your original file

If you want it *even closer* to your original formatting (like matching comment tone exactly), I can adjust it to look almost identical to your Week 4 template.

keep the grid size teh same 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);

Ahhh okay — I see what you want.

You want to **keep this exact setup format**:

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

That means:

- `grid` must already exist before `setup()`
- We should NOT call `createCanvas()` inside `loadLevel()`
- The grid size must always be **16 × 11**

So here is the **clean, simple version**, same format as yours, same canvas logic, but now:

- Levels stored in `LEVELS`
- `grid` starts as Level 1

- Level 2 loads automatically
- Canvas line stays EXACTLY how you wrote it

```
/*
Week 4 – Grid + Dynamic Levels (Same Setup Format)
Course: GBDA302
Instructors: Dr. Karen Cochrane and David Han
Date: Feb. 5, 2026
```

PURPOSE:

1. Generate level using arrays/JSON data (LEVELS)
2. Use nested loops to dynamically place tiles

BONUS: Level 2 loads automatically

```
*/
```

```
const TS = 32;
```

```
/*
GRID LEGEND:
0 = floor
1 = wall
2 = obstacle
3 = word
4 = goal
*/
```

```
// 16 columns × 11 rows (SAME as original)
```

```
const LEVELS = [
  {
    grid: [
      [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
      [1,0,0,0,0,1,0,0,3,0,0,1,0,0,4,1],
      [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],
      [1,0,0,0,0,1,0,0,0,1,0,0,0,1,3,1],
```

```

        [1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,1],
        [1,0,0,1,0,0,2,1,0,0,0,1,0,0,0,1],
        [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    ],
},
{
    grid: [
        [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
        [1,0,3,0,0,1,0,0,0,0,0,1,0,0,4,1],
        [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],
        [1,0,0,0,0,1,0,0,0,1,0,0,0,1,3,1],
        [1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,1],
        [1,0,0,1,0,0,2,1,0,0,0,1,0,0,0,1],
        [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    ],
},
];

let levelIndex = 0;
let grid = LEVELS[0].grid; // START WITH LEVEL 1

let pr = 1;
let pc = 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);

    noStroke();
    textFont("sans-serif");

```



```

    textSize(14);
}

/*
p5.js DRAW: Runs 60 times per second
*/
function draw() {

    background(240);

    // NESTED LOOPS: dynamically draw tiles from grid data
    for (let r = 0; r < grid.length; r++) {
        for (let c = 0; c < grid[0].length; c++) {

            let tile = grid[r][c];

            if (tile === 1) fill(30, 50, 60);           // wall
            else if (tile === 2) fill(120, 35, 35);    // obstacle
            else if (tile === 3) fill(255, 210, 70);   // word
            else if (tile === 4) fill(40, 170, 80);    // goal
            else fill(230);                             // floor

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

    // Player
    fill(40, 110, 255);
    rect(pc * TS + 6, pr * TS + 6, TS - 12, TS - 12);

    fill(0);
    text("Level " + (levelIndex + 1), 10, 16);
}

function keyPressed() {

    let dr = 0;
    let 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;

    let nr = pr + dr;
    let nc = pc + dc;

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

    let next = grid[nr][nc];

    if (next === 1 || next === 2) return; // blocked






    pr = nr;
    pc = nc;

    if (next === 3) grid[nr][nc] = 0; // collect word

    // BONUS: auto-load Level 2 when reaching goal
    if (next === 4 && levelIndex < LEVELS.length - 1) {
        levelIndex++;
        grid = LEVELS[levelIndex].grid;
        pr = 1;
        pc = 1;
    }
}

```

Now:

-  Your `createCanvas(grid[0].length * TS, grid.length * TS);` line stays exactly the same.
-  Grid size stays **16 × 11**.
-  Uses arrays/JSON (LEVELS).
-  Uses nested loops.
-  Auto loads Level 2.

If you want it *even closer* to your original (like keeping the exact comment tone), I can refine it further.

make the second level ma bit challenging than teh first

```
/*
Week 4 – Grid + Dynamic Levels (Same Setup Format)
Course: GBDA302
Instructors: Dr. Karen Cochran and David Han
Date: Feb. 5, 2026
```

PURPOSE:

1. Generate level using arrays/JSON data (LEVELS)
2. Use nested loops to dynamically place tiles

BONUS: Level 2 loads automatically

```
*/
```

```
const TS = 32;
```

```
/*
GRID LEGEND:
0 = floor
1 = wall
2 = obstacle
3 = word
4 = goal
*/
```

```
// 16 columns × 11 rows (SAME as original)
```

```
const LEVELS = [
  {
    // LEVEL 1 (easier)
    grid: [
      [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
      [1,0,0,0,0,1,0,0,3,0,0,1,0,0,4,1],
      [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],
      [1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,1],
    ]
  }
]
```

```

        [1,1,1,1,1,1,0,1,1,1,0,1,1,1,0,1],
        [1,0,0,0,0,1,0,0,0,1,0,0,0,1,3,1],
        [1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,1],
        [1,0,0,1,0,0,2,1,0,0,0,1,0,0,0,1],
        [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    ],
},
{
    // LEVEL 2 (harder: tighter corridors + more obstacles
+ a "trap" section)
    grid: [
        [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],
        [1,0,1,1,0,1,0,1,1,1,0,1,0,1,0,1],
        [1,0,1,0,0,0,0,2,0,1,0,0,0,1,0,1],
        [1,0,1,0,1,1,1,1,0,1,1,1,0,1,0,1],
        [1,0,0,0,0,2,0,1,0,0,0,1,0,0,0,1],
        [1,1,1,1,1,1,0,1,1,1,0,1,1,1,0,1],
        [1,0,3,0,0,1,0,0,0,1,0,0,2,1,0,1],
        [1,0,1,1,0,1,1,1,0,1,1,1,0,1,0,1],
        [1,0,0,1,0,0,2,1,0,0,0,1,0,0,4,1],
        [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
    ],
},
];

let levelIndex = 0;
let grid = LEVELS[0].grid; // START WITH LEVEL 1

let pr = 1;
let pc = 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);

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

/*
p5.js DRAW: Runs 60 times per second
*/
function draw() {

    background(240);

    // NESTED LOOPS: dynamically draw tiles from grid data
    for (let r = 0; r < grid.length; r++) {
        for (let c = 0; c < grid[0].length; c++) {

            let tile = grid[r][c];

            if (tile === 1) fill(30, 50, 60);           // wall
            else if (tile === 2) fill(120, 35, 35);    // obstacle
            else if (tile === 3) fill(255, 210, 70);   // word
            else if (tile === 4) fill(40, 170, 80);    // goal
            else fill(230);                             // floor

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

    // Player
    fill(40, 110, 255);
    rect(pc * TS + 6, pr * TS + 6, TS - 12, TS - 12);

    fill(0);
    text("Level " + (levelIndex + 1), 10, 16);
}

function keyPressed() {

```

```

let dr = 0;
let 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;

let nr = pr + dr;
let nc = pc + dc;

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

let next = grid[nr][nc];

if (next === 1 || next === 2) return; // blocked

pr = nr;
pc = nc;

if (next === 3) grid[nr][nc] = 0; // collect word

// BONUS: auto-load Level 2 when reaching goal
if (next === 4 && levelIndex < LEVELS.length - 1) {
  levelIndex++;
  grid = LEVELS[levelIndex].grid;
  pr = 1;
  pc = 1;
}
}

```