

Process & Decision Documentation

Side Quests and A4 (Individual Work)

GenAI Documentation

I was trying to turn the Week 4 grid sketch into a simple **data-driven level system** by storing levels in JSON/arrays and using nested loops to render tiles, then adding a player character, collectible “word” items, and an automatic switch to Level 2 after finishing Level 1.

One small decision I made was to **store the player position in grid coordinates (r, c)** instead of pixel coordinates, because it made wall collision and collecting words (tile-by-tile movement) much simpler and consistent with the 2D array map.

GenAI use: I used ChatGPT to help outline the step-by-step plan (levels JSON → loadLevel() → loops for drawing → player movement/collision → word collection → auto level switch) and to suggest code structure; I then implemented and adjusted the final sketch myself.

Appendix

GenAI transcript:

Me: hi, im working on this assignment where I have to generate a level using arrays or JSON data; use loops to dynamically place tiles, obstacles, or words I already have `const grid = [. . .]`. I don't want just one grid though. I want multiple levels. Here is the 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, 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, 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, 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, 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); }

```

ChatGPT: Perfect. Instead of one `grid`, we'll make `levels = [level1, level2]`. Each level will store:

- `grid` (2D array)
- `start` position
- `goal` position
- `words` array for collectibles

Example structure:

```

const levels = [
  { name:"Level 1", grid:[...], start:{r:1,c:1}, goal:{r:9,c:14},
  words:[...] },
  { name:"Level 2", grid:[...], start:{r:1,c:1}, goal:{r:9,c:14},
  words:[...] }
];

```

Me: wait don't make it complicated with classes. Keep it simple JSON.

ChatGPT: Totally. No classes. Just plain objects + a `loadLevel()` function to swap data.

Me: Okay now add a character. I want a player circle and it should move by tiles, not pixels.

ChatGPT: Great. We'll store the player in grid coordinates:

```

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

```

And when drawing, we convert it to screen coordinates:

```

ellipse(player.c*TS + TS/2, player.r*TS + TS/2, TS*0.6);

```

Me: No not like that don't let the player walk through walls. I need collision.

ChatGPT: Yup. We'll add `tryMove(dr, dc)` that checks the next tile before moving:

```

function tryMove(dr, dc) {
  const nr = player.r + dr;
  const nc = player.c + dc;
  if (grid[nr][nc] === 1) return; // wall blocks
  player.r = nr;
  player.c = nc;
}

```

```
}
```

Then arrow keys call it:

```
function keyPressed() {  
  if (keyCode === LEFT_ARROW) tryMove(0,-1);  
  if (keyCode === RIGHT_ARROW) tryMove(0,1);  
  if (keyCode === UP_ARROW) tryMove(-1,0);  
  if (keyCode === DOWN_ARROW) tryMove(1,0);  
}
```

Me: also keep my nested loops to draw the map. It should draw the current level's grid.

ChatGPT: Perfect. We'll keep your grid rendering loop, but swap `grid` to point to the current level's grid:

```
for (let r = 0; r < grid.length; r++) {  
  for (let c = 0; c < grid[0].length; c++) {  
    if (grid[r][c] === 1) fill(30,50,60);  
    else fill(230);  
    rect(c*TS, r*TS, TS, TS);  
  }  
}
```

Me: Next: I need "words" to collect. Put words in level JSON and draw them with a loop.

ChatGPT: Done. Each level gets:

```
words: [  
  { r: 1, c: 2, text: "G" },  
  { r: 5, c: 2, text: "B" }  
]
```

And drawing uses a loop:

```
for (let w of words) {  
  if (w.collected) continue;  
  text(w.text, w.c*TS + TS/2, w.r*TS + TS/2);  
}
```

Me: Wait don't hardcode `text()` multiple times. Use a loop only.

ChatGPT: Yup, fully loop-based. No manual placements. Everything comes from the array data.

Me: Now add collecting. When I step on a word, it disappears.

ChatGPT: We'll check after movement (or every draw). If player is on the same cell, mark collected:

```
function checkCollect() {
  for (let w of words) {
    if (!w.collected && w.r === player.r && w.c === player.c) {
      w.collected = true;
    }
  }
}
```

Me: No no don't delete from the array. Just mark collected so I can track progress.

ChatGPT: Exactly why we use collected: true/false. Then we can count:

```
const collectedCount = words.filter(w => w.collected).length;
```

Me: Add a HUD: "Words: 2/4".

ChatGPT: Easy:

```
text(`${level.name} | Words: ${collectedCount}/${words.length}`, 10, 10);
```

Me: Now I need a goal tile, like green. But it should only finish if I collected everything.

ChatGPT: Good rule. We'll draw the goal:

```
rect(level.goal.c*TS, level.goal.r*TS, TS, TS);
```

And the finish logic:

```
function allCollected() {
  return words.every(w => w.collected);
}

function isLevelFinished() {
  const atGoal = player.r === level.goal.r && player.c === level.goal.c;
  return allCollected() && atGoal;
}
```

Me: Wait make sure goal is in the JSON too, not hardcoded.

ChatGPT: Yes, goal: { r: 9, c: 14 } is stored per level.

Me: Bonus: make level 2 load automatically when level 1 is finished. No button.

ChatGPT: Perfect. We'll have `levelIndex`, and `advanceLevel()`:

```
function advanceLevel() {
  if (levelIndex < levels.length - 1) loadLevel(levelIndex + 1);
  else gameWon = true;
}
```

Then in `draw()`:

```
if (isLevelFinished()) advanceLevel();
```

Me: And when it loads the next level, reset player to start and reset words.

ChatGPT: Exactly. That's what `loadLevel()` does:

```
function loadLevel(i) {
  levelIndex = i;
  level = levels[levelIndex];
  grid = level.grid;
  player.r = level.start.r;
  player.c = level.start.c;
  words = level.words.map(w => ({ ...w, collected: false }));
}
```

Me: Also handle canvas sizing for each level.

ChatGPT: If needed, we can safely do:

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

inside loadLevel().
```

Me: Perfect, if I finish level 2, show "YOU WIN".

ChatGPT: Yep — simple end state:

```
if (gameWon) {
  text("YOU WIN 🎉", width/2, height/2);
  return;
}
```