

# Process & Decision Documentation

## Side Quests and A4 (Individual Work)

I decided to incorporate a “goal” to reach the next level because I thought it was the clearest way for the user to know what they need to achieve.

### Entry Header

Name: Fodé Doumbouya

Role(s): Individual

Primary responsibility for this work: Complete Side Quest 4

### *Goal of Work Session*

- For this side quest, my goal was to add a level to the blob game using a clear “goal” the user would have to reach.

Tools, Resources, or Inputs Used

- Chat GPT 5.2

### *GenAI Documentation*

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

**Date Used:** Tuesday February 10<sup>th</sup>, 2025

**Tool Disclosure:** ChatGPT 5.2

**Purpose of Use:** I used Gen AI to incorporate my ideas into the project

**Summary of Interaction:** I asked the AI model to help me turn my ideas into code I could input to make it work; I also had the AI model help me understand my task better.

**Human Decision Point(s):** I overruled some of what the AI was giving me by making my prompts more specific so it would only give me the requested info

**Integrity & Verification Note:** I reviewed all the code I was given to ensure there would be no errors.

**Scope of GenAI Use:** The ideation, and the “goal” which I changed the colour

**Limitations or Misfires:** I needed to give the tool full context in order for it to help, originally it started giving me code I didn't need.

### *Summary of Process (Human + Tool)*

- I thought of the idea of adding the goal, what it would look like, and how the game would basically work (how the player would get to the next level)

### *Decision Points & Trade-offs*

I decided to add the goal, because it seemed like a good perceived affordance to include in a simple game like this.

### *Verification & Judgement*

- I thought about it, and it made the most sense to include it.

### *Limitations, Dead Ends, or Open Questions*

- There were no real limitations, just that it took time to double check things that the model was giving me because I had to make sure the brackets were right and the indentation was right, etc.

## Appendix

yoo you tryna help me with sum, it's another assignment 🤖

Yoo of course 🤖 always got you.

What's the assignment this time—what course, what's due, and what are you stuck on?

Paste it in and we'll cook it step by step 🔥

this is the assignment description: 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.

Bet — this assignment is basically: **store the level as data** (array/JSON), then **use loops to build it** (place tiles/obstacles/words). And for the bonus, **swap to level 2 when level 1 is complete**.

Below is a clean, “works in almost any JS game” template using **JSON + a 2D grid**. You can adapt the `spawn()` function to whatever engine you're using (p5.js, Phaser, Canvas, etc.).

---

## 1) Level data (arrays / JSON)

Use a grid where each character means something:

- # = wall/obstacle
- . = floor/empty
- P = player start
- E = exit/finish
- W = word collectible

```
const LEVELS = [
  {
    name: "Level 1",
    tileSize: 32,
    grid: [
      "#####",
      "#P.....W.....#",
      "#....#####...#",
      "#.....#",
      "#..#####..###.#",
      "#.....W.....#",
      "#.....E...#",
      "#####",
    ],
    words: {
      // key = grid coordinate "x,y" matching where the W is
      "7,1": "FOCUS",
      "7,5": "MOVE",
    }
  },
  {
    name: "Level 2",
    tileSize: 32,
    grid: [
      "#####",
      "#P.....#",
      "#..#####..###.#",
      "#..W.....#.....#",
      "#..#####.###.#",
      "#.....#...E.#",
      "#.....#.....#",
      "#####",
    ],
    words: {
      "3,3": "SPEED"
    }
  }
];
```

---

## 2) Loop to generate the level dynamically

This function reads the grid and “spawns” stuff. Replace `spawnWall`, `spawnPlayer`, etc. with your engine’s code.

```
let currentLevelIndex = 0;
let levelState = null;

function loadLevel(index) {
  const level = LEVELS[index];
  currentLevelIndex = index;

  // reset everything in your world
  levelState = {
    walls: [],
    words: [],
    exit: null,
    player: null,
    collected: 0,
    totalWords: countWords(level.grid)
  };

  // LOOP through rows/cols and place objects
  for (let y = 0; y < level.grid.length; y++) {
    const row = level.grid[y];
    for (let x = 0; x < row.length; x++) {
      const cell = row[x];
      const px = x * level.tileSize;
      const py = y * level.tileSize;

      if (cell === "#") {
        levelState.walls.push(spawnWall(px, py, level.tileSize));
      } else if (cell === "P") {
        levelState.player = spawnPlayer(px, py, level.tileSize);
      } else if (cell === "E") {
        levelState.exit = spawnExit(px, py, level.tileSize);
      } else if (cell === "W") {
        const key = `${x},${y}`;
        const text = level.words?.[key] ?? "WORD";
        levelState.words.push(spawnWord(px, py, level.tileSize, text));
      }
    }
  }

  console.log("Loaded:", level.name);
}

function countWords(grid) {
  let total = 0;
  for (const row of grid) {
    for (const ch of row) if (ch === "W") total++;
  }
  return total;
}
```

---

### 3) Bonus: auto-load level 2 when level 1 finishes

Define what “finished” means. Common choices:

- player reaches exit **AND** collected all words
- or just reaches exit

Here’s: **collect all words, then touch exit.**

```
function updateGame() {
  // move player, collisions, etc...

  // word collection
  for (const w of levelState.words) {
    if (!w.collected && intersects(levelState.player, w)) {
      w.collected = true;
      levelState.collected++;
      console.log("Collected:", w.text);
    }
  }

  // check finish condition
  const allWordsCollected = (levelState.collected === levelState.totalWords);
  const atExit = levelState.exit && intersects(levelState.player,
levelState.exit);

  if (allWordsCollected && atExit) {
    goToNextLevel();
  }
}

function goToNextLevel() {
  const next = currentLevelIndex + 1;
  if (next < LEVELS.length) {
    loadLevel(next); // AUTO loads level 2
  } else {
    console.log("Game complete!");
  }
}
```

---

### 4) Minimal “spawn” + collision placeholders (so you see the structure)

You’ll replace these with your actual drawing/engine objects.

```
function spawnWall(x, y, s) { return { type: "wall", x, y, w: s, h: s }; }
function spawnPlayer(x, y, s) { return { type: "player", x, y, w: s, h: s }; }
function spawnExit(x, y, s) { return { type: "exit", x, y, w: s, h: s }; }
```

```
function spawnWord(x, y, s, text) { return { type: "word", x, y, w: s, h: s,
text, collected: false }; }

function intersects(a, b) {
  return (
    a.x < b.x + b.w &&
    a.x + a.w > b.x &&
    a.y < b.y + b.h &&
    a.y + a.h > b.y
  );
}
```

Start the game with:

```
loadLevel(0);
```

---

If you tell me what you're building in (p5.js? Phaser? Unity but with JSON? Scratch-like?), I can **translate this into the exact code style you need** (same logic, just the right functions + rendering).

I think im doing it from my profs examples: here is the code in the sketch.js file in vs code /\*  
Week 4 — Example 5: Example 5: Blob Platformer (JSON + Classes) Course: GBDA302  
Instructors: Dr. Karen Cochrane and David Han Date: Feb. 5, 2026 This file orchestrates  
everything: - load JSON in preload() - create WorldLevel from JSON - create BlobPlayer -  
update + draw each frame - handle input events (jump, optional next level) This matches  
the structure of the original blob sketch from Week 2 but moves details into classes. \*/ let  
data; // raw JSON data let levelIndex = 0; let world; // WorldLevel instance (current level) let  
player; // BlobPlayer instance function preload() { // Load the level data from disk before  
setup runs. data = loadJSON("levels.json"); } function setup() { // Create the player once (it  
will be respawned per level). player = new BlobPlayer(); // Load the first level. loadLevel(0);  
// Simple shared style setup. noStroke(); textFont("sans-serif"); textSize(14); } function  
draw() { // 1) Draw the world (background + platforms) world.drawWorld(); // 2) Update and  
draw the player on top of the world player.update(world.platforms);  
player.draw(world.theme.blob); // 3) HUD fill(0); text(world.name, 10, 18); text("Move: A/D  
or </> • Jump: Space/W/↑ • Next: N", 10, 36); } function keyPressed() { // Jump keys if (key  
=== " " || key === "W" || key === "w" || keyCode === UP\_ARROW) { player.jump(); } //  
Optional: cycle levels with N (as with the earlier examples) if (key === "n" || key === "N") {  
const next = (levelIndex + 1) % data.levels.length; loadLevel(next); } } /\* Load a level by  
index: - create a WorldLevel instance from JSON - resize canvas based on inferred  
geometry - spawn player using level start + physics \*/ function loadLevel(i) { levelIndex = i;  
// Create the world object from the JSON level object. world = new  
WorldLevel(data.levels[levelIndex]); // Fit canvas to world geometry (or defaults if needed).

```
const W = world.inferWidth(640); const H = world.inferHeight(360); resizeCanvas(W, H); //
Apply level settings + respawn. player.spawnFromLevel(world); } idk how to load the json
file tho I tried to open this on GitHub website but the link brought me to a blank page, idk if
it's because I need to add my own code or bc im not doing something right, maybe json?
```

You're actually **very close** — nothing is “broken,” you're just missing **one required file** and one key concept about how `loadJSON()` works in p5.

This is a super common GBDA302 gotcha, so don't stress 🤔

---

## The key thing you're missing

`loadJSON("levels.json")` **does NOT fetch from GitHub**  
It loads a **local file** that must exist **in the same project folder** as `sketch.js`.

So right now:

- Your `sketch.js` is fine ✅
- p5 is looking for `levels.json`
- ...but that file **doesn't exist locally**, so data is empty → blank / broken behavior

That's why GitHub showed a blank page — there's no JSON file being served to your sketch.

---

## What your folder should look like

In VS Code:

```
project-folder/
├── index.html
├── sketch.js
├── levels.json ← YOU NEED THIS
├── BlobPlayer.js
├── WorldLevel.js
├── libraries/
│   └── p5.js
```

If `levels.json` isn't there, `loadJSON()` can't load anything.




---

## Minimal `levels.json` you can copy-paste (WORKS)

Create a new file called `levels.json` and paste this in:

```
{
  "levels": [
    {
      "name": "Level 1: Warm Up",
      "theme": {
        "background": "#e8f0ff",
        "platform": "#333333",
        "blob": "#ff6f61"
      },
      "gravity": 0.8,
      "jumpStrength": -14,
      "start": { "x": 80, "y": 200 },
      "platforms": [
        { "x": 0, "y": 300, "w": 640, "h": 60 },
        { "x": 200, "y": 240, "w": 120, "h": 20 },
        { "x": 380, "y": 190, "w": 120, "h": 20 }
      ]
    },
    {
      "name": "Level 2: Higher Ground",
      "theme": {
        "background": "#fff3e0",
        "platform": "#222222",
        "blob": "#4caf50"
      },
      "gravity": 0.9,
      "jumpStrength": -15,
      "start": { "x": 60, "y": 180 },
      "platforms": [
        { "x": 0, "y": 320, "w": 640, "h": 60 },
        { "x": 150, "y": 260, "w": 100, "h": 20 },
        { "x": 300, "y": 210, "w": 100, "h": 20 },
        { "x": 450, "y": 160, "w": 100, "h": 20 }
      ]
    }
  ]
}
```

This already satisfies the assignment:

-  **JSON-based level data**
  -  **Multiple levels**
  -  **Automatically loadable via index**
-



## Why your prof's example uses JSON like this

Your sketch does:

```
data = loadJSON("levels.json");
world = new WorldLevel(data.levels[levelIndex]);
```

So `WorldLevel` expects **one level object**, shaped like:

```
{
  name,
  theme,
  gravity,
  start,
  platforms
}
```

You're not supposed to generate platforms in `sketch.js` — that happens **inside `WorldLevel` using loops**, which is exactly what the assignment asks for.

This line in your assignment description:

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



That loop is almost certainly **already inside `WorldLevel`**.

So you're doing this **the professor-approved way** 👍

---

## How to check if JSON is loading (quick debug)

Add this inside `setup()` temporarily:

```
console.log(data);
```

If you see the levels printed → you're good

If you see `undefined` → file path / file missing

---

## Bonus: Auto-load next level without pressing N

Later, you can replace the `N` key with something like:

```
if (player.reachedGoal) {  
  loadLevel(levelIndex + 1);  
}
```

That'll get you **bonus marks** with zero extra structure.

---

## TL;DR

- You **don't** load JSON from GitHub
- You **must create** `levels.json` locally
- Your `sketch.js` is already correct
- This setup is **exactly what your prof wants**

If you want, next we can:

- sanity-check your `WorldLevel` class
- make sure loops are clearly visible for grading
- add a win condition so Level 2 loads automatically 🙄

worldLevel.js

JavaScript

style.css

File

sketch.js

JavaScript

README.md

File

platform.js

JavaScript

levels.json

File

jsconfig.json

File

index.html

File

blobPlayer.js

JavaScript

these are all my files, think you could lmk wassup from here?

Thought for 1m 30s

Yeah — looking at what you uploaded, **nothing is “mysteriously wrong” with JSON**. The two biggest “why is it blank?” issues from here are:

## 1) Your GitHub Pages blank page is almost certainly a filename/case issue

In your `index.html`, you’re loading these files:

- `Platform.js`
- `WorldLevel.js`
- `BlobPlayer.js`

`index`

But your actual files are named (lowercase):

- `platform.js`
- `worldLevel.js`
- `blobPlayer.js`

On **Windows/Mac**, this often “works anyway.”

On **GitHub Pages (Linux)** it’s **case-sensitive**, so those scripts 404 → nothing runs → blank page.

✅ Fix (pick one):

- **Option A (easiest):** change your `index.html` to match your real filenames:
  - `<script src="platform.js"></script>`
  - `<script src="worldLevel.js"></script>`
  - `<script src="blobPlayer.js"></script>`
- **Option B:** rename the files to `Platform.js`, `WorldLevel.js`, `BlobPlayer.js` to match the HTML.

That alone usually fixes the “blank GitHub page” problem.

---

## 2) If you're opening `index.html` by double-clicking, `loadJSON` may fail

`loadJSON("levels.json")` needs the file served by a local server. If you open `file:///.../index.html`, browsers often block file fetching.

✅ Fix:

- Use **VS Code Live Server** (Right click `index.html` → “Open with Live Server”)
- Or upload to GitHub Pages / use p5 editor

(Once scripts load correctly, GitHub Pages will also serve `levels.json` fine.)

---

## 3) Your `levels.json` currently only has ONE level

Your JSON has only one item in `"levels": [ ... ]`

`levels`

So even if you press **N**, it just reloads the same level because there's nothing else to switch to.

✅ Fix: add a second level object inside the `levels` array.

---

## 4) Quick check: your JSON keys match what your code expects

Your level data uses:

- `jumpV`
- `gravity`
- `theme.bg`, `theme.platform`, `theme.blob`
- `start`
- `platforms`

`levels`

That matches the structure your prof's setup is clearly aiming for, so your JSON format is fine.

---

## What I'd do next (fastest path)

1. **Fix the script filenames in `index.html`** (case mismatch)

`index`

2. Run with **Live Server** (don't double click the html)
3. Add **Level 2** to `levels.json`

`levels`

4. (Bonus) Add a “finish condition” (like reaching the far right or a goal object) →  
`auto loadLevel (levelIndex + 1)`

If you want, paste your GitHub Pages URL (or tell me if you want “finish” to mean *touch a goal, reach the right edge, or collect something*), and I'll tell you the cleanest 5-line change to make Level 2 auto-load.

### Sources

oh ok it works when I run the html in live server, so ig that's what I gotta do? now say I wanted to add levels, I can either use arrays or a json file I have the choice, what do you think?

Yup — **that's exactly it** 👍

For this project, **Live Server is the correct way** to run it. `loadJSON()` needs a server, so your prof will almost certainly do the same (or run it via p5 editor / GitHub Pages). You're good.

Now to the real question 📌

---

## Arrays vs JSON for levels — what should *you* use?

Short answer (but I'll explain):

👉 **Stick with JSON** for this assignment.

Not because arrays are bad — but because **JSON scores you more points with less risk** given your prof's examples.

---

## Why JSON is the smarter move *here*

### 1. Your prof is already teaching JSON

Your current setup literally says:

```
data = loadJSON("levels.json");  
world = new WorldLevel(data.levels[levelIndex]);
```

That tells the marker:

- “I understand external data”
- “I can separate data from logic”
- “I can scale levels cleanly”

Markers **love** that.

---

### 2. JSON = clearer evidence you used loops + data-driven design

Your assignment says:

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

With JSON:

- The **data** is visible in `levels.json`
- The **loop** is visible in `WorldLevel`
- The separation is obvious

With arrays in `sketch.js`, it can blur together and look more “hardcoded.”

---

### 3. JSON makes adding levels stupidly easy

To add a new level, you just copy-paste:

```
{  
  "name": "Level 2",  
  "gravity": 0.9,  
  "jumpV": -16,
```

```
"start": { "x": 60, "y": 200 },
"theme": {
  "bg": "#fff3e0",
  "platform": "#222",
  "blob": "#4caf50"
},
"platforms": [
  { "x": 0, "y": 300, "w": 640, "h": 60 },
  { "x": 200, "y": 250, "w": 120, "h": 20 }
]
}
```

Drop it into "levels": [ ... ] and you're done.

No JS logic touched = less chance of breaking stuff.

---

## When arrays *would* make sense (but not needed here)

Arrays are great if:

- you're doing a tile grid ( [ ['#', '.', '#'], ... ] )
- you're generating patterns procedurally
- you're NOT already using JSON

But your prof **already gave you a JSON-based architecture**, so switching would actually be more work.

---

## What I'd submit if I were you (max marks, minimal stress)

- ✓ Keep **levels.json**
- ✓ Add **at least 2 levels**
- ✓ Let `WorldLevel` loop through platforms
- ✓ Run with **Live Server**
- ✓ (Bonus) Auto-load next level

That hits **every line** of the assignment description.

---

## Bonus: auto-load next level (very clean, prof-friendly)

Inside `draw()` in `sketch.js`, add:

```

if (player.pos.x > width) {
  const next = levelIndex + 1;
  if (next < data.levels.length) {
    loadLevel(next);
  }
}

```

Now:

- Finish level by reaching the right edge
- Level 2 loads automatically
- You get the **bonus requirement** without extra UI

If you want, next we can:

- make a tiny “goal platform” instead of screen edge
- add words/collectibles (also from JSON 🙄)
- sanity-check `WorldLevel` so your loops are crystal-clear for grading

You’re honestly in a really good spot right now.

ok I have an idea, basically, I want to make it so that the blob needs to touch some kind of activator for the level to be over, so for the first level, it’ll be like a cold circle/oval and when the blob makes contact with it it’ll go to the next level

Thought for 1m 5s

Yeahhh that idea is perfect for this setup 🔥 — you can add a “**goal/activator**” object to your JSON, have `WorldLevel` read it, **draw it as a cold ellipse**, and then in `draw()` check collision: if the blob touches it → **load next level**.

Here’s the clean way to do it with *your exact codebase*.

## 1) Add a goal to each level in `levels.json`

Right now you only have 1 level. Change it to have **two levels**, and add a `goal` rectangle for each one (we’ll draw it as an ellipse, but collide as a rectangle — super common + easy).

```

{
  "schemaVersion": 1,
  "levels": [
    {
      "name": "Intro Steps",

```



```

    "gravity": 0.65,
    "jumpV": -11.0,
    "theme": { "bg": "#F0F0F0", "platform": "#C8C8C8", "blob": "#1478FF" },
    "start": { "x": 80, "y": 220, "r": 26 },

    "goal": { "x": 560, "y": 110, "w": 50, "h": 70 },

    "platforms": [
      { "x": 0, "y": 324, "w": 640, "h": 36 },
      { "x": 120, "y": 254, "w": 120, "h": 12 },
      { "x": 300, "y": 204, "w": 90, "h": 12 },
      { "x": 440, "y": 144, "w": 130, "h": 12 },
      { "x": 520, "y": 254, "w": 90, "h": 12 }
    ]
  },
  {
    "name": "Level 2: Cold Switch",
    "gravity": 0.75,
    "jumpV": -12.0,
    "theme": { "bg": "#EAF6FF", "platform": "#B7D3E6", "blob": "#FF4D6D" },
    "start": { "x": 80, "y": 220, "r": 26 },

    "goal": { "x": 60, "y": 90, "w": 50, "h": 70 },

    "platforms": [
      { "x": 0, "y": 324, "w": 640, "h": 36 },
      { "x": 140, "y": 270, "w": 90, "h": 12 },
      { "x": 260, "y": 230, "w": 90, "h": 12 },
      { "x": 380, "y": 190, "w": 90, "h": 12 },
      { "x": 500, "y": 150, "w": 90, "h": 12 }
    ]
  }
]
}

```

---

## 2) Update `worldLevel.js` to store + draw the goal

Add this in the constructor **after** `this.start`:

```

// Goal / activator (rectangle for collision; drawn as ellipse)
this.goal = {
  x: levelJson.goal?.x ?? 560,
  y: levelJson.goal?.y ?? 120,
  w: levelJson.goal?.w ?? 50,
  h: levelJson.goal?.h ?? 70,
};

```

Then add a method to draw it (put this below `drawWorld()`):

```

drawGoal() {
  // "cold" look
  push();

```

```

noStroke();

// icy outer glow
fill(180, 225, 255, 180);
ellipse(this.goal.x + this.goal.w / 2, this.goal.y + this.goal.h / 2,
        this.goal.w * 1.2, this.goal.h * 1.2);

// inner core
fill(120, 200, 255, 220);
ellipse(this.goal.x + this.goal.w / 2, this.goal.y + this.goal.h / 2,
        this.goal.w, this.goal.h);

pop();
}

```

---

### 3) In `sketch.js`, draw the goal + detect touch to load next level

In `draw()`, right after `world.drawWorld();`, add:

```
world.drawGoal();
```

Then after `player.update(...)` add the win check:

```

// Win condition: touch the goal (activator)
const playerBox = {
  x: player.x - player.r,
  y: player.y - player.r,
  w: player.r * 2,
  h: player.r * 2,
};

if (overlapAABB(playerBox, world.goal)) {
  const next = levelIndex + 1;
  if (next < data.levels.length) loadLevel(next);
  else console.log("All levels complete!");
}

```

✅ That's it. Now level ends when the blob touches the cold oval.

---

### 4) IMPORTANT for GitHub Pages later (so it doesn't go blank)

Your `index.html` currently loads `Platform.js`, `WorldLevel.js`, `BlobPlayer.js`.

But your files are named lowercase (`platform.js`, `worldLevel.js`, `blobPlayer.js`).

**Live Server can still work on your machine**, but **GitHub Pages will break** unless cases match.

So change your script tags to:

```
<script src="platform.js"></script>  
<script src="worldLevel.js"></script>  
<script src="blobPlayer.js"></script>  
<script src="sketch.js"></script>
```

---

If you want, I can also help you make the activator **animate** (pulsing / icy shimmer) so it's obvious to the player where to go, and that'll look really good for grading.