

Entry Header

Name:

Aswina Sivaloganathan

Role:

Programmer

Short description:

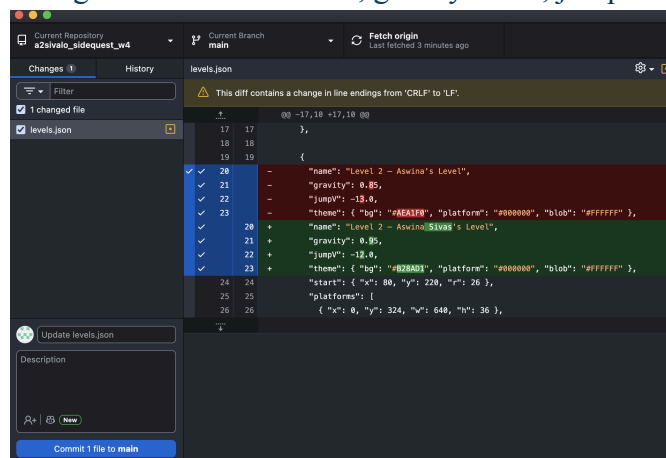
I used GenAI to **understand** how the Week 4 Blob Platformer example generates levels from JSON data, then **modified the levels.json file myself** to create a second level with a new platform layout, gravity + jump values, and visual theme.

Small decision or changes made:

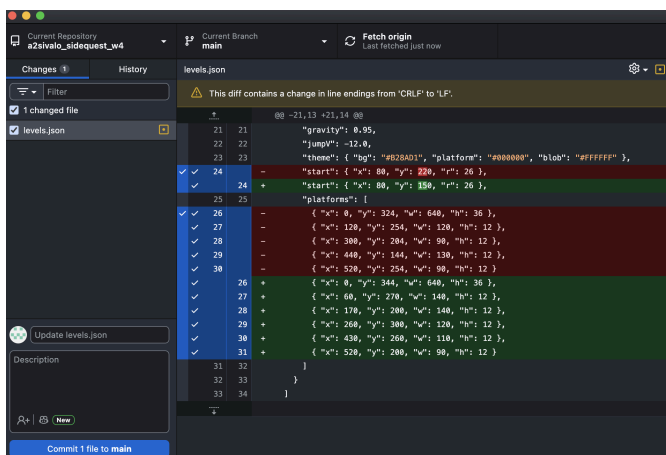
I chose to add a second level by editing the JSON data instead of modifying the core drawing logic. I adjusted background and theme colours, platform positions, gravity, and jump strength to create a more challenging layout to show level progression.

Evidence:

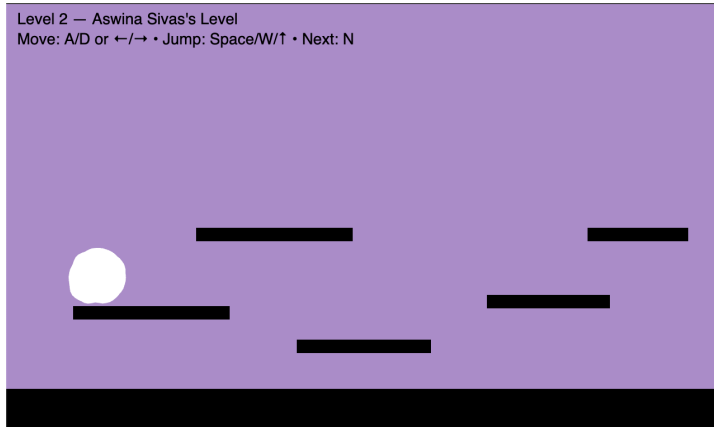
Change #1: The level name, gravity value, jump and theme colours.



Change #2: The platform difficulty, changing the x and y values.



Final Result:



GenAI Documentation

GenAI tools were used for this task.

Date Used:

February 9th, 2026

Tool Disclosure:

ChatGPT 5.2

Appendix

Transcript:

I'm working on a Week 4 Side Quest for a p5.js course. The requirement says: "Generate a level using arrays or JSON data; use loops to dynamically place tiles, obstacles, or words." Can you explain what this means in simple terms?

Totally — this requirement sounds scarier than it actually is. Let's break it down very simply, without code jargon overload 😊

What the requirement is asking (in plain English)

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

This means:

- 👉 **Don’t hard-code everything manually**
- 👉 **Store level info in data (arrays or JSON)**
- 👉 **Use a loop to draw things based on that data**

Instead of saying:

“Draw a square here. Draw another square there. Draw another one...”

You say:

**“Here’s a list that describes my level.
Now loop through it and draw whatever it says.”**

Step 1: What is “generating a level”?

A level can be *anything*, depending on your project:

- **A grid of tiles**
- **Floating blobs**
- **Words on the screen**
- **Obstacles the player avoids**
- **UI elements that appear in positions**

💡 **For your Side Quest, a “level” does NOT need to be a full game level.
It can be as simple as:**

- **A screen layout**
 - **A mood visualization**
 - **A pattern that reacts or animates**
-

Step 2: What does “using arrays or JSON data” mean?

Instead of drawing things one by one, you store the instructions in data.

Example idea (conceptual, not code)

You might store:

- **Positions**
- **Sizes**
- **Types**
- **States (calm vs anxious, active vs inactive)**

Array example (mental model)

Think of an array like a list:

- **Shape 1 goes here**
- **Shape 2 goes here**
- **Shape 3 goes here**

Each item in the list represents one thing in your level.

JSON example (mental model)

JSON is like a labeled object:

- **This thing is a “tile”**
- **It has a size**
- **It has a position**
- **It has a mood/state**

You do not need JSON unless you want structure — arrays are totally fine for Side Quest.

Step 3: What does “use loops to dynamically place things” mean?

This is the most important part.

Instead of:

```
draw tile at (100, 100)
draw tile at (200, 100)
draw tile at (300, 100)
```

You do:

“Loop through my list and draw whatever’s inside.”

So:

- **The loop reads your data**
- **p5 draws things based on the data**
- **If you add/remove data, the level changes automatically**

That’s what “dynamically” means.

Step 4: What counts as “tiles, obstacles, or words”?

You can choose any one of these (or something similar):

Tiles

- **Repeating shapes in a grid**
- **Patterned blobs**
- **Visual blocks**

Obstacles

- **Shapes that block movement**
- **Visual noise**

- Objects that jitter or feel “in the way”

Words

- Text placed around the canvas
- Repeating phrases
- Words that change position or mood

💡 Words count just as much as shapes.
This is great if your project is more expressive or emotional.

Very simple example idea (no code)

If you’re doing anxiety (which you mentioned earlier 🙄):

- Store a list of “blobs”
- Each blob has:
 - x position
 - y position
 - jitter amount
- Use a loop to draw them
- Slight randomness makes them shake

✅ That already satisfies the requirement:

- Level = emotional layout
- Data = array of blobs
- Loop = draws them all

- Dynamic = easy to change
-

What your TA is actually checking for

They're asking:

- ✓ Did you store your level in data?
- ✓ Did you use a loop to create visuals from that data?
- ✓ Did you avoid manually placing everything?

They are NOT asking for:

- Advanced AI
 - Perfect game logic
 - Complex systems
-

TL;DR (one sentence)

Put your level info in an array (or JSON), then use a loop to draw things based on that data instead of placing everything manually.

If you want, tell me:

- what your Side Quest concept is
- whether you're using shapes or text

and I'll help you map it exactly to this requirement without overcomplicating it 🧡

I am using the Week 4 “Blob Platformer” lecture example, which loads level data from a levels.json file. How does this example generate a level using JSON data and loops?

Great question — this lecture example is actually a perfect, textbook match for that requirement. I'll explain it step-by-step in conceptual terms, not code-heavy, so you can also *describe it confidently* in a submission or README.

Big picture: what the Blob Platformer is doing

The Blob Platformer does not hard-code the level layout in the sketch.

Instead, it:

1. Stores the level layout in a **levels.json** file
2. Loads that data into p5
3. Uses loops to read the data
4. Draws platforms, obstacles, and elements based on what the data says

That is *exactly* what “generate a level using arrays or JSON data; use loops” means.

Step 1: What the **levels.json file represents**

Think of **levels.json as a blueprint for the level.**

Inside the file, the level is described using data like:

- Rows and columns
- Numbers or symbols that represent different tiles
- Separate arrays for platforms, hazards, or goals

So instead of saying:

“Draw a platform at x=100, y=300”

The file says something like:

“At this grid position, place a platform”

The JSON doesn’t draw anything — it only *describes* the level.

Step 2: How JSON becomes usable in the sketch

When the sketch runs:

- p5 loads **levels.json**
- The JSON turns into a JavaScript object
- That object contains arrays (lists) of level data

At this point, the program has:

- A list of rows
- Each row has a list of tiles
- Each tile value represents a type of object

This is where the “arrays or JSON data” requirement is fulfilled.

Step 3: How loops generate the level

This is the key idea.

The sketch uses loops to:

- Go through each row of the level data
- Then go through each tile in that row
- Check what type of tile it is
- Draw the correct object in the correct place

Conceptually, it does:

For every row
→ for every column
→ look at the value
→ draw the matching tile

Because of this:



- Adding a new row to the JSON automatically extends the level
- Changing numbers in the JSON changes the level layout
- No drawing code has to be rewritten

That's what "dynamically place tiles or obstacles" means.

Step 4: Why this counts as "generating" a level

The level is generated at runtime, not manually placed.

Important distinctions:

-  Manually placing platforms = *not* generation
-  Reading data + looping = *generation*

The Blob Platformer:

- Does not know the level shape ahead of time
- Builds the level entirely from the JSON
- Uses logic to decide *what to place and where*

So the level is:

- Data-driven
 - Scalable
 - Reusable
-

Why your instructor included this example

This example exists to show you:

- How separating data from visuals works
- How professional games manage levels
- How loops + data = flexible design

If you're using or modifying this example, you are 100% aligned with the Week 4 requirement.

If you want, next I can help you:

- Identify which exact parts of the Blob Platformer code you can point to
- Adapt this logic to something more expressive (like emotion-based blobs)

Just tell me 👍

What does it mean in this context to add a level?

In this specific context (the Week 4 Blob Platformer), “adding a level” has a very concrete, simple meaning.

Short answer (plain English)

Adding a level means creating another layout in the `levels.json` file that the game can load and build automatically using loops.

You are not manually drawing a new level in code.

What a “level” actually is here

In the Blob Platformer, a level = data, not visuals.

A level is:

- A set of rows and columns

- Stored in `levels.json`
- Made of values that represent platforms, empty space, hazards, etc.

Each level is just:

“Here’s another grid of numbers/symbols”

Nothing more.

What changes when you “add” a level

When you add a level, you:

- Do not touch the drawing logic
- Do not rewrite loops
- Do not place platforms manually

You only add new data.

Conceptually:

- Level 1 → grid layout A
- Level 2 → grid layout B
- Level 3 → grid layout C

The same loops read whichever level is active and generate it.

Why this counts as “generating a level”

The important idea your instructor cares about:

The program doesn’t care what the level looks like — it just reads data and builds it.

That means:

- New levels are data-driven
- The system is reusable
- This is how real games scale

So “adding a level” proves that:

- ✓ You understand data-driven design
 - ✓ You’re using JSON correctly
 - ✓ You’re relying on loops, not manual placement
-

What adding a level does *not* mean

- ✗ Adding new drawing code
- ✗ Copy-pasting platform objects
- ✗ Creating a new JS file
- ✗ Making a whole new game

If all you did was:

- Edit **levels.json**
- Add another level entry
- Let the existing code load it

You did it right.

One mental model to remember

Think of it like Minecraft:

- The engine = your p5 sketch
- The world seed / map = **levels.json**
- Adding a level = adding a new map file

Same engine. Different data.

If you want, next I can:

- **Help you describe how many levels you should add (minimum vs bonus)**
- **Help you tweak a level so it clearly shows “dynamic generation”**

Just say the word 👍