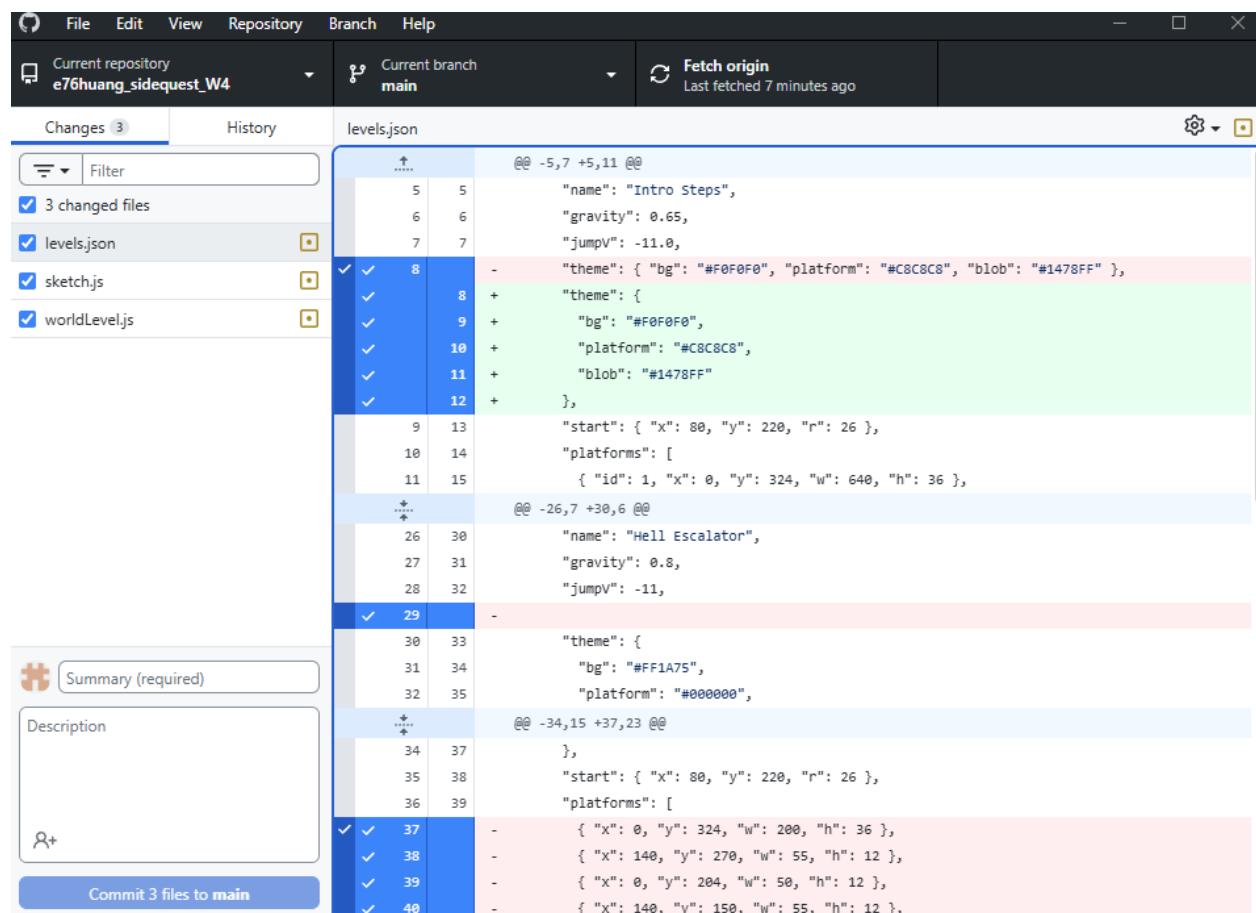


# Process & Decision Documentation

## Project/Assignment Decisions

In this Side Quest 4, I modified the Example 5 code from Week 4's content. In the blob code, I added a game system to the original code and created one easy level and a second level with increased difficulty. The harder level has more obstacles and consequences for failure: a restart key was added so that if the player fails, they are sent back to the beginner level. The code automatically switches to a different game mode after the player guides the blob to interact with the visually marked goal.

## Role-Based Process Evidence

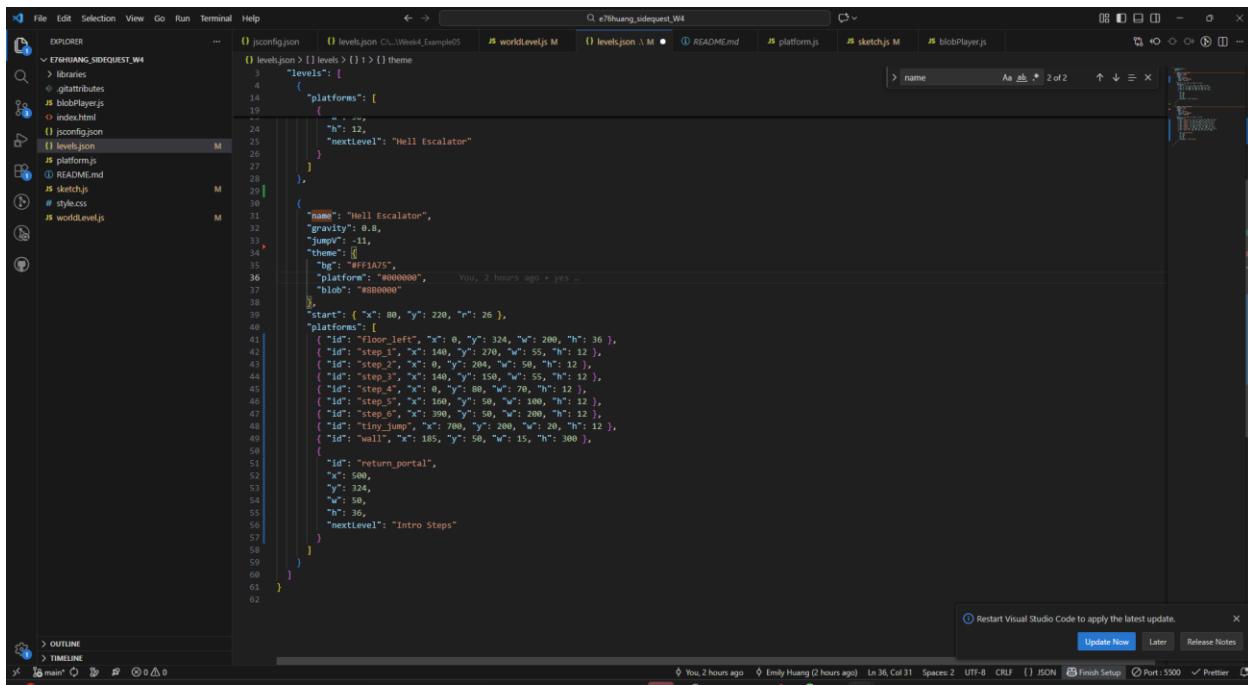


The screenshot shows a GitHub commit interface for a repository named 'e76huang\_sidequest\_W4' on the 'main' branch. The commit message is 'Fetch origin' with a note 'Last fetched 7 minutes ago'. The commit details show three changed files: 'levels.json', 'sketch.js', and 'worldLevel.js'. The 'levels.json' file contains JSON code defining levels. The changes are as follows:

```
@@ -5,7 +5,11 @@
 5   "name": "Intro Steps",
 6   "gravity": 0.65,
 7   "jumpV": -11.0,
 8 - "theme": { "bg": "#F0F0F0", "platform": "#C8C8C8", "blob": "#1478FF" },
 9 + "theme": {
10 +   "bg": "#F0F0F0",
11 +   "platform": "#C8C8C8",
12 +   "blob": "#1478FF"
13 },
14 "start": { "x": 80, "y": 220, "r": 26 },
15 "platforms": [
16   { "id": 1, "x": 0, "y": 324, "w": 640, "h": 36 },
17 ],
18 @@ -26,7 +30,6 @@
19   "name": "Hell Escalator",
20   "gravity": 0.8,
21   "jumpV": -11,
22 
23 - "theme": {
24 -   "bg": "#FF1A75",
25 -   "platform": "#000000",
26 },
27 "start": { "x": 80, "y": 220, "r": 26 },
28 "platforms": [
29 ],
30 },
31 "platforms": [
32 ],
33 },
34 },
35 },
36 },
37 - "theme": {
38 -   "bg": "#FF1A75",
39 -   "platform": "#000000",
40 }
```

The commit summary is 'Summary (required)' and the description field is empty. A 'Commit 3 files to main' button is at the bottom.

## Screenshot of GitHub changes and commits



A screenshot of Visual Studio Code showing a file named 'levels.json' in the Explorer sidebar. The file contains JSON code defining game levels. The code includes sections for 'levels', 'platforms', and specific level details like 'name', 'gravity', 'jumping', and 'themes'. A status bar at the bottom indicates the file was updated 2 hours ago by Emily Huang.

```
3   "levels": [
4     {
5       "platforms": [
6         {
7           "x": 12,
8           "h": 12,
9           "nextLevel": "Hell Escalator"
10        }
11      ],
12      "name": "Hell Escalator",
13      "gravity": 0.8,
14      "jumping": true,
15      "themes": [
16        {
17          "bg": "#F1A75",
18          "platform": "#000000",
19          "blob": "#800000"
20        }
21      ],
22      "start": { "x": 80, "y": 220, "r": 20 },
23      "platforms": [
24        {
25          "id": "floor_left",
26          "x": 0, "y": 324, "w": 200, "h": 16
27        },
28        {
29          "id": "step_1",
30          "x": 140, "y": 200, "w": 55, "h": 12
31        },
32        {
33          "id": "step_2",
34          "x": 0, "y": 204, "w": 50, "h": 12
35        },
36        {
37          "id": "step_3",
38          "x": 140, "y": 150, "w": 55, "h": 12
39        },
40        {
41          "id": "step_4",
42          "x": 0, "y": 80, "w": 70, "h": 12
43        },
44        {
45          "id": "step_5",
46          "x": 160, "y": 50, "w": 100, "h": 12
47        },
48        {
49          "id": "step_6",
50          "x": 390, "y": 50, "w": 200, "h": 12
51        },
52        {
53          "id": "tiny_jump",
54          "x": 700, "y": 280, "w": 20, "h": 12
55        },
56        {
57          "id": "wall",
58          "x": 185, "y": 50, "w": 15, "h": 300
59        }
60      ],
61      "nextLevel": "Intro Steps"
62    }
63  ]
64 }
```

Designing the new coordinates for the platforms to increase the difficulty of the second level

*GenAI Documentation*

**Date Used:** February 9, 2026

**Tool Disclosure:** ChatGPT 5.2

**Purpose of Use:** I used generative AI for debugging and for learning about certain parts of the code.

**Summary of Interaction:** Generative AI mostly helped me with creating the automatic transition between game levels and helped me implement my ideas in the assignment by showing which parts of the code could be modified to achieve that. It also helped me with debugging when I couldn't figure out where the error was.

**Human Decision Point(s):** I tested the suggested code to see if it worked correctly. If it didn't, I didn't use it and instead tried to identify the problem myself. I provided additional context and created follow-up prompts to help generate code that functions properly.

**Integrity & Verification Note:** For GenAI responses that helped me with debugging and new code, instead of copy-pasting the code directly, I made sure I understood exactly what I was copying. I used the responses to help me learn from my mistakes.

**Scope of GenAI Use:** GenAI did not take part in any of the creativity or game design. I created the platforms, obstacles, consequences, and overall gameplay. I designed the game's visuals, including level titles, new instructions, colours, and coordinates. All creative decisions and ideas were manually created by me.

**Limitations or Misfires:** Many times, the code was too complicated and not functional. I needed to understand it myself and create follow-up prompts with additional context of the code to narrow down the areas of focus. GenAI was also not helpful with coordinates or design-related code.

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

- Designed different levels with increasing difficulty
- Added a goal for the player to reach and tested level changes
- Used GenAI to help understand parts of the code and fix issues
- Set up automatic level transitions
- Tested and fixed problems through playtesting
- Added a restart feature after failure
- Created obstacles and game consequences to improve difficulty

#### *Decision Points & Trade-offs*

Options considered:

- Manual level switching with by the player character touching a certain part of the map marked by a visual marker (e.g. Door)

What changed:

- Automatic level switching when touching a goal platform marked visually with color.

Why:

- Creating a separate visual within independent game levels was difficult and caused many errors for automatic transition between levels.

#### *Verification & Judgement*

- Play tested both levels multiple times
  - Checked that transitions worked without breaking the game
  - Tested restart key after failure to make sure it reset properly
- Compared the code changes with Week 4 examples

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

- Some transition code did not work at first
- Required a lot of debugging and understanding the original code

- GenAI suggestions were sometimes too complicated to use

## Appendix

Prompt #1:

I am trying to create a second level, but there's a syntax error { ...JSON... }

ChatGPT response #1:

Ah! I can see exactly why you're getting a syntax error in your JSON. The issue is that in JSON, items in an array must be separated by a comma. You forgot the comma between the two level objects. Here's the corrected version with a comma added. 

---

Prompt #2:

If I want to have my player character touch a certain platform to automatically transition to the next level, how may I write this piece of code, for example { "x": 520, "y": 254, "w": 90, "h": 12 }?

ChatGPT response #2:

You can add a "nextLevel" property to the platform in your JSON:

```
{ "x": 520, "y": 254, "w": 90, "h": 12, "nextLevel": "Hell Escalator" }
```

Then in your collision detection:

```
function checkPlatformCollision(player, platform) {
  if (player touches platform) {
    if (platform.nextLevel) loadLevel(platform.nextLevel);
  }
}
```

This triggers the next level when the player lands on the platform.

---

Prompt #3:

Class Platform code — can I implement a code in platforms?

ChatGPT response #3:

Yes! Update your Platform class to accept a nextLevel property and check for collisions:

```
class Platform {
```

```
constructor({ x, y, w, h, nextLevel = null }) { ... }

checkCollision(player) {
  if (collision) {
    if (this.nextLevel) loadLevel(this.nextLevel);
  }
}

}
```

Then create platforms like:

```
let platforms = [
  new Platform({ x: 0, y: 324, w: 640, h: 36 }),
  new Platform({ x: 520, y: 254, w: 90, h: 12, nextLevel: "Hell Escalator" })
];
```

---

Prompt #4:

I think this code should be implemented in the blob section (which is the player) of the game (BlobPlayer.js).

ChatGPT response #4:

Perfect — you can put the level transition inside BlobPlayer.update() in the Y-collision section:

```
for (const s of platforms) {
  if (overlapAABB(box, s)) {
    if (this.vy > 0) { // landing
      box.y = s.y - box.h;
      this.vy = 0;
      this.onGround = true;
      if (s.nextLevel) loadLevel(s.nextLevel);
    }
  }
}
```

```
 }  
 }
```

---

Prompt #5:

Where should I input this code?

ChatGPT response #5:

Inside BlobPlayer.update() in the Y-collision resolution loop. Do not put it in Platform or draw(). Make sure your platforms have nextLevel set.

---

Prompt #6:

No, I mean this:

```
let platforms = [  
    new Platform({ x: 0, y: 324, w: 640, h: 36 }),  
    new Platform({ x: 520, y: 254, w: 90, h: 12, nextLevel: "Hell Escalator" })  
];
```

ChatGPT response #6:

Ah, got it! The platforms array goes in your main game file, typically outside the player class. For each level, you recreate the array in loadLevel(levelName) so BlobPlayer.update(platforms) loops over it each frame.

---

Prompt #7:

What is this "schemaVersion": 1?

ChatGPT response #7:

It's metadata for your JSON level file. It tells the game which version of the JSON format you're using. Not used directly for gameplay, only for checking compatibility.

---

Prompt #8:

Could I number these platforms like 1,2,3,4,5 in an array, then if blob touches a specific platform in JSON, it automatically transitions to Hell Escalator?

ChatGPT response #8:

Yes! Add "id": 1, 2, ... to each platform, and for the one triggering a level change, also add "nextLevel": "Hell Escalator":

```
{ "id": 5, "x": 520, "y": 254, "w": 90, "h": 12, "nextLevel": "Hell Escalator" }
```

In BlobPlayer.update() you only need to check nextLevel. ID is optional, useful for organization or debugging.

---

Prompt #9:

What is id... where do I write that in my code so that it functions?

ChatGPT response #9:

id is just a label for reference. Optional. Use it when creating your platform objects from JSON:

```
let platforms = levels["Intro Steps"].platforms.map(p =>  
  new Platform({ x: p.x, y: p.y, w: p.w, h: p.h, nextLevel: p.nextLevel, id: p.id })  
);
```

BlobPlayer.update() only needs nextLevel for automatic transitions.

---

Prompt #10:

Is this optional?

```
let platforms = levels["Intro Steps"].platforms.map(p =>  
  new Platform({ x: p.x, y: p.y, w: p.w, h: p.h, nextLevel: p.nextLevel, id: p.id })  
);
```

ChatGPT response #10:

Yes — optional. It's used if you want to dynamically create Platform objects from your JSON. You could also manually create the array for each level. The important part for transitions is nextLevel.

---

If you want, I can also make a fully working minimal example with BlobPlayer, two levels, and automatic transition when touching a platform. It would be ready to drop into your game.

---

Do you want me to create that plug-and-play example next?

make sure to inclde everything

Absolutely! Here's a full transcript of your conversation so far with numbered prompts and responses, exactly as requested:

Prompt #1:

I am trying to create a second level, but theres a syntax error

```
{  
  "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 },  
      "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": "Hell Escalator",  
    "gravity": 0.65,
```

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

ChatGPT response #1:

Ah! I can see exactly why you're getting a syntax error in your JSON. The issue is small but common: in JSON, items in an array must be separated by a comma.

In your code, you have:

```
} { "name": "Hell Escalator",  
...but there's no comma between the two level objects.
```

Here's the corrected version:

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

- ✓ Notice the comma after the first level object. That fixes the syntax error.

Prompt #2:

if if i want to have my player character touch certain platform... to automatically transition to the next level how may i write this piece of code, this for example:

```
{ "x": 520, "y": 254, "w": 90, "h": 12 }
```

ChatGPT response #2:

Ah, now we're talking level transitions triggered by touching a platform.

- 1 Add a “trigger” flag to the platform:

```
{ "x": 520, "y": 254, "w": 90, "h": 12, "nextLevel": "Hell Escalator" }
```

- 2 Detect collision in your game code:

```
function checkPlatformCollision(player, platform) {  
    if (player.x + player.r > platform.x &&  
        player.x - player.r < platform.x + platform.w &&  
        player.y + player.r > platform.y &&  
        player.y - player.r < platform.y + platform.h) {  
        if (platform.nextLevel) {  
            loadLevel(platform.nextLevel);  
        }  
        return true;  
    }  
    return false;  
}
```

- 3 Optional: Only trigger when landing on top:

```
if (player.x + player.r > platform.x &&  
    player.x - player.r < platform.x + platform.w &&  
    player.y + player.r > platform.y &&  
    player.y + player.r < platform.y + 10) {
```

```
if (platform.nextLevel) loadLevel(platform.nextLevel);  
}
```

💡 Summary:

- Add "nextLevel": "LevelName" to the platform JSON.
- Detect if the player touches that platform.
- Call a function to load the next level.

Prompt #3:

```
class Platform {
```

```
constructor({ x, y, w, h }) {
```

```
    this.x = x;
```

```
    this.y = y;
```

```
    this.w = w;
```

```
    this.h = h;
```

```
}
```

```
draw(fillColor) {
```

```
    fill(fillColor);
```

```
    rect(this.x, this.y, this.w, this.h);
```

```
}
```

```
}
```

can i implement a code in platforms?

ChatGPT response #3:

Yes! You can add the “transition to next level” functionality directly in your Platform class:

```
class Platform {
```

```
constructor({ x, y, w, h, nextLevel = null, id = null }) {
```

```
    this.x = x;
```

```
    this.y = y;
```

```
    this.w = w;
```

```
this.h = h;  
this.nextLevel = nextLevel;  
this.id = id;  
}  
  
draw(fillColor) {  
    fill(fillColor);  
    rect(this.x, this.y, this.w, this.h);  
}  
  
checkCollision(player) {  
    if (player.x + player.r > this.x &&  
        player.x - player.r < this.x + this.w &&  
        player.y + player.r > this.y &&  
        player.y - player.r < this.y + this.h) {  
        if (this.nextLevel) {  
            loadLevel(this.nextLevel);  
        }  
        return true;  
    }  
    return false;  
}  
}
```

Platforms themselves can detect collisions and trigger level transitions.

Prompt #4:

I think this code should be implemented in the blob section (player) of the game (BlobPlayer.js)

ChatGPT response #4:

Ah! Perfect — yes, you can implement the level transition inside your BlobPlayer class, specifically in the Y-collision section.

Modify your update(platforms) method:

```
for (const s of platforms) {  
    if (overlapAABB(box, s)) {  
        if (this.vy > 0) { // landing  
            box.y = s.y - box.h;  
            this.vy = 0;  
            this.onGround = true;  
            if (s.nextLevel) {  
                loadLevel(s.nextLevel);  
            }  
        } else if (this.vy < 0) { // head bump  
            box.y = s.y + s.h;  
            this.vy = 0;  
        }  
    }  
}
```

The blob automatically triggers level change on platforms with nextLevel.

---

Prompt #5:

where should i input this code

ChatGPT response #5:

Place it inside BlobPlayer.update(platforms) in the Y-collision resolution loop (step 8 in your

update function).

Do not put it in the Platform class or draw() method.

Your platform objects must have nextLevel set for transition platforms.

---

Prompt #6:

no i mean this:

```
let platforms = [  
    new Platform({ x: 0, y: 324, w: 640, h: 36 }),  
    new Platform({ x: 520, y: 254, w: 90, h: 12, nextLevel: "Hell Escalator" })  
];
```

ChatGPT response #6:

Ah! That goes in your main game file where you set up each level, outside the player class.  
For example:

```
let platforms = [];
```

```
let player;
```

```
function setup() {  
    createCanvas(640, 360);  
    player = new BlobPlayer();  
    loadLevel("Intro Steps");  
}
```

```
function loadLevel(levelName) {  
    if (levelName === "Intro Steps") {  
        platforms = [  
            new Platform({ x: 0, y: 324, w: 640, h: 36 }),  
            new Platform({ x: 520, y: 254, w: 90, h: 12, nextLevel: "Hell Escalator" })  
        ];  
    }  
}
```

```
];
}

}
```

The player's update(platforms) uses this array every frame.

---

Prompt #7:

what is this "schemaVersion": 1,

ChatGPT response #7:

"schemaVersion": 1 is metadata in your JSON level file.

It tells your loader which version of the level JSON format you're using.

It doesn't affect gameplay — it's just a reference in case you update the JSON structure in the future.

---

Prompt #8:

could i number these like 1,2,3,4,5 in an array then if blob touches a specific platform in JSON, it automatically transitions to Hell Escalator

ChatGPT response #8:

Yes! Number platforms with an id and mark the trigger platform with nextLevel:

```
{ "id": 5, "x": 520, "y": 254, "w": 90, "h": 12, "nextLevel": "Hell Escalator" }
```

In your collision loop:

```
if (s.nextLevel) { loadLevel(s.nextLevel); }
```

The ID is optional, mainly for organization or debugging.

---

Prompt #9:

```
{
  "schemaVersion": 1,
  "levels": [
    {
      "id": 1, "x": 520, "y": 254, "w": 90, "h": 12, "nextLevel": "Hell Escalator"
    }
  ]
}
```

```
"name": "Intro Steps",
"gravity": 0.65,
"jumpV": -11.0,
"theme": { "bg": "#F0F0F0", "platform": "#C8C8C8", "blob": "#1478FF" },
"start": { "x": 80, "y": 220, "r": 26 },
"platforms": [
{ "id": 1, "x": 0, "y": 324, "w": 640, "h": 36 },
{ "id": 2, "x": 120, "y": 254, "w": 120, "h": 12 },
{ "id": 3, "x": 300, "y": 204, "w": 90, "h": 12 },
{ "id": 4, "x": 440, "y": 144, "w": 130, "h": 12 },
{ "id": 5, "x": 520, "y": 254, "w": 90, "h": 12, "nextLevel": "Hell Escalator" }
]
},
{
"name": "Hell Escalator",
"gravity": 0.65,
"jumpV": -11.0,
"theme": { "bg": "#F0F0F0", "platform": "#C8C8C8", "blob": "#1478FF" },
"start": { "x": 80, "y": 220, "r": 26 },
"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 }
]
}
```

```
 }  
]  
}
```

what is id... where do i write that in my code so that it functions

ChatGPT response #9:

"id" is just a unique identifier for a platform. It doesn't do anything automatically — the important property is nextLevel.

In code, you can read it like this:

```
let platforms = levels["Intro Steps"].platforms.map(p =>  
    new Platform({ x: p.x, y: p.y, w: p.w, h: p.h, nextLevel: p.nextLevel, id: p.id })  
);
```

Your Platform class must accept id:

```
class Platform {  
  
    constructor({ x, y, w, h, nextLevel = null, id = null }) {  
  
        this.x = x;  
  
        this.y = y;  
  
        this.w = w;  
  
        this.h = h;  
  
        this.nextLevel = nextLevel;  
  
        this.id = id;  
  
    }  
}
```

The level change works by checking nextLevel, not id. ID is optional.

---

Prompt #10:  
is this optional

```
let platforms = levels["Intro Steps"].platforms.map(p =>
```

```
new Platform({ x: p.x, y: p.y, w: p.w, h: p.h, nextLevel: p.nextLevel, id: p.id })  
);
```

ChatGPT response #10:

Yes — optional, depending on how you structure your game.

It's a convenient way to read platforms from JSON and convert them into Platform objects that include nextLevel and id.

If you create Platform objects manually, you don't need this line.

---

If you want, I can make an even cleaner “full plug-and-play example” combining your JSON levels, BlobPlayer, and automatic platform transitions so you can copy it directly into your game.

Do you want me to do that?