

Process & Decision Documentation

Project/Assignment Decisions

Side Quests and A4 (Individual Work)

Keep this section brief, typically 2 to 4 sentences.

Focus on:

- A significant decision I made was to get rid of half my 2d array of platforms and adding a moving platforms that takes up the entirety of the right side of my game.
- I also decided at the beginning I wanted a uniform grid platform level so that it would be easier for the player to get to the finish line.

Examples:

- Adding a moving platform on the right
- Adding a red box on the top right to act as the finish line that ends the level
- Change the layout of my initial grid level idea when I realized that the player could not advance towards the goal

Role-Based Process Evidence

Entry Header

Name: Katherine Chen

Role(s): Main Coder/Editor

Primary responsibility for this work: Using AI to fulfill side quest 4's requirements, as well as being the human decision point of how the level should progress.

Goal of Work Session

Examples:

- Creating a level that uses arrays or JSON data, as well as loops, obstacles, dynamic items (moving platform).
- Automatically loaded up the next level once the current one is done

Tools, Resources, or Inputs Used

- GenAI:
 - Claude Sonnett 4.5
 - VSCode Agent
- VSCode
- GitHub

GenAI Documentation

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

Date Used: Feb 8th, 2026

- **Tool Disclosure:** Claude Sonnett 4.5, VSCode Agent

Purpose of Use: I had an idea of how I wanted to format the level at the beginning, but I did not know the optimal way of feeding prompts to VSCode's Agent so that it would properly

implement my idea into code. Thus, I made Claude and VSCode's Agent work together so that I could turn my vision into reality.

Summary of Interaction: The tool was mainly in charge of implementing my ideas. It did not decide how the level would look, rather I would feed it my vision and it would turn it into code. Of course it also helped me troubleshoot some problems that would have taken me way too long to look for.

Human Decision Point(s): The human decision point in this was that I came up with the level's format idea myself (making it a grid/jungle format, adding moving platforms), as well as doing some small tweeks to AI's code so that certain visual items were displayed the way I wanted them to. I understood most of the implementations that AI made, I just needed them to do it since coding this all myself would have taken way too long. However, knowing meant that I was able to tweak smaller details such as the 0,1 2D array that was in charge of displaying which platforms were visible or not.

Integrity & Verification Note: Since I personally know HTML, JavaScript, and CSS, I was able to verify most of which the code wrote and understood that it was mostly correct. Also I kept on testing some edge cases and well as if there were any accidental infinite loops through live server just to make sure my game was running properly.

Scope of GenAI Use: Like mentioned earlier. GenAI did not contribute to the creative portions of this. The ideas on how the level would look all came from human decisions (me).

Limitations or Misfires: Unfortunately, for things like the blob starting point edge case where the blob kept on spawning at the wrong spot was an AI error that I had to troubleshoot myself. Also AI wouldn't have been able to give me an idea on how to format my level because then it would have been too technical and impractical.

Summary of Process (Human + Tool)

- I utilized AI to make my creative game vision into code
- I utilized it to troubleshoot some errors
- I came up with my own ideas on how the code should look (grid + moving platforms) in order to fulfill the sidequest requirements
- I did some of my own tweaking directly to the code when it was too tedious to ask AI (ect. Changing the y axis of the blob starting point in different levels)

Decision Points & Trade-offs

- I did some tweeks myself such as changing the blob's jumping height after I realized that the base one would not allow my blobs to reach certain platforms
- I also did some tweeks on the starting point. The initial one kept spawning the blob under the first platform (edge case since this only happened if I help the right key arrow after dying) and had to make changes to the y-axis

Verification & Judgement

- Playtesting was a big part of how I made sure the code was running smoothly. Throughout different portions of the process, my interface would do things like glitch, infinite loop on itself, ect. And I had to personally troubleshoot why that happens and fix it.
- I also took into consideration that players may not know what the red square on the top right meant so I wrote it next to the rest of the instructions

Limitations, Dead Ends, or Open Questions

- There were quite a few things that crashed at the beginning, for example my array was set up in a way where there were platforms on the right while also having a moving platform, causing a major glitch
 - I fixed it by getting rid of the platforms
- Currently I am still trying to figure out how I can not stick with a static array to organize my platforms, but rather make it more “free” looking as it only functions as a 3x4 grid right now.
 - Im thinking that I would have to create a grid with fully custom positions to fix this, however, that way making changes would be quite tedious

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

Prompt	Answer
can you just give me a set of instructions to ask vscode's copilot/agent to code for this idea: platforms arranged in rows and columns (like a jungle gym), where teh player jumps through the grid to reach the top and uses nexted loops	<p>Implementation Plan:</p> <p>What we'll do:</p> <ol style="list-style-type: none">1. Add a <code>generateGrid()</code> function to <code>WorldLevel.js</code> that uses nested loops2. Update the constructor to use it when it sees a pattern3. Add a new grid level to <code>levels.json</code>4. Keep your original level working
	<p>Prompts to Give Copilot (use in order):</p> <p>Prompt 1:</p> <p>In <code>WorldLevel.js</code>, add a new method called <code>generateGrid</code> to the <code>WorldLevel</code> class. It should take these parameters: <code>rows</code>, <code>cols</code>, <code>startX</code>, <code>startY</code>, <code>platformWidth</code>, <code>platformHeight</code>,</p>

	<p>spacingX, spacingY. Use a nested for loop (one loop inside another) to create platforms in a grid pattern. The outer loop should go through rows, the inner loop should go through columns. Return an array of Platform objects.</p>
	<p>Prompt 2:</p> <p>In the WorldLevel constructor, after the line that sets this.start, add an if statement. Check if levelJson.pattern exists. If it does and pattern.type equals "grid", call this.generateGrid() and pass in the pattern's properties. Otherwise, keep the existing code that maps platforms from the JSON array.</p>
	<p>Prompt 3:</p> <p>In levels.json, add a new level object after the first level. Name it "Grid Climb". Instead of a platforms array, give it a pattern object with type "grid", rows 3, cols 4, startX 80, startY 100, platformWidth 100, platformHeight 12, spacingX 60, spacingY 80. Also add a ground platform separately at y: 324.</p>
	<p>Prompt 4:</p> <p>Update the new grid level in levels.json so it has both a pattern for the grid AND a platforms array with just one platform for the ground floor at the bottom.</p>
	<p>Prompt 5 (test it):</p> <p>Make sure the grid level is playable. The player should spawn at x: 80, y: 280 and be able to jump through the grid to reach the top.</p>
	<p>That's it! Use these one at a time and test between each one. Let me know if Copilot gets confused on any of them!</p>
In WorldLevel.js, add a new method called generateGrid to the WorldLevel class. It should	I'll open worldLevel.js to find the <code>WorldLevel</code> class and where to add <code>generateGrid</code> .

take these parameters: rows, cols, startX, startY, platformWidth, platformHeight, spacingX, spacingY. Use a nested for loop (one loop inside another) to create platforms in a grid pattern. The outer loop should go through rows, the inner loop should go through columns. Return an array of Platform objects.

I'll create a short TODO plan for adding `generateGrid` to [worldLevel.js](#).

Created 2 todos

Figuring out the tool call

I need to fix the `apply_patch` call because it failed due to a missing explanation field. I'll make sure to add that. Also, I have to include a preamble before the `apply_patch`, just like we did before with the `manage_todo_list`.

So, I'll send a preamble message like, "Now I'll patch `worldLevel.js` to add the `generateGrid` method." Then, I can call `apply_patch` again with the explanation. I'll use the same patch as before.

Now I'll patch [worldLevel.js](#) to add the `generateGrid` method.

Edited

`worldLevel.js+16-0`

I'll update the TODO list to mark the work complete.

Completed: Add `generateGrid` method (1/2)

Done — [WorldLevel](#) now has `generateGrid(rows, cols, startX, startY, platformWidth, platformHeight, spacingX, spacingY)` which returns an array of `Platform` objects arranged in a grid. Want me to add an example usage in [levels.json](#) or wire it into the level parsing?

```
74     background(color(this.theme.bg));
75     for (const p of this.platforms) {
76       p.draw(color(this.theme.platform));
77     }
78   }
79
80   /**
81    * Generate a grid of platforms and return an array of Platform instances.
82    * Outer loop iterates rows, inner loop iterates columns.
83    */
84   generateGrid(rows, cols, startX, startY, platformWidth, platformHeight, spacingX) {
85     const platforms = [];
86     for (let r = 0; r < rows; r++) {
87       for (let c = 0; c < cols; c++) {
88         const x = startX + c * (platformWidth + spacingX);
89         const y = startY + r * (platformHeight + spacingY);
90         platforms.push(new Platform({ x: x, y: y, w: platformWidth, h: platformHeight }));
91       }
92     }
93     return platforms;
94   }
95 }
96 }
```

In the WorldLevel constructor, after the line that sets this.start, add an if statement. Check if levelJson.pattern exists. If it does and pattern.type equals "grid", call this.generateGrid() and pass in the pattern's properties. Otherwise, keep the existing code that maps platforms from the JSON array.

I'll patch `worldLevel.js` constructor to use `levelJson.pattern` (grid) when present.

```
Edited
worldLevel.js+16-2
Added 1 todo
Done — WorldLevel constructor now
uses levelJson.pattern.type === "grid" to
call this.generateGrid(...); otherwise it maps platforms as
before. Want me to add a sample pattern entry
to levels.json for testing?
```

```
// Convert raw platform objects into Platform instances.
this.platforms = (levelJson.platforms || []).map((p) => new Platform(p));
// Convert raw platform objects into Platform instances or generate from a
if (levelJson.pattern && levelJson.pattern.type === "grid") {
  const p = levelJson.pattern;
  this.platforms = this.generateGrid(
    p.rows,
    p.cols,
    p startX,
    p startY,
    p.platformWidth,
    p.platformHeight,
    p.spacingX,
    p.spacingY
  );
} else {
  this.platforms = (levelJson.platforms || []).map((p) => new Platform(p));
}
/*
If you want the canvas to fit the world, you can infer width/height by
finding the maximum x+w and y+h across all platforms.
*/
inferWidth(defaultWidth - 500) /
```

In `levels.json`, add a new level object after the first level. Name it "Grid Climb". Instead of a platforms array, give it a pattern object with type "grid", rows 3, cols 4, startX 80, startY 100, platformWidth 100, platformHeight 12, spacingX 60, spacingY 80. Also add a ground platform separately at y: 324.

I'll read `levels.json` to find the correct insertion point for the new level.

Now I'll insert the Grid Climb level into `levels.json` after the first level.

```
Edited
levels.json+22-0
Created 4 todos
Done — added the Grid Climb level to levels.json (pattern grid +
ground platform). Want me to run the sketch or add a short
demo in index.html that loads this level?
```

```

16   ]
17 }
18 }, Expected comma
19 {
20   "name": "Grid Climb",
21   "gravity": 0.65,
22   "jumpV": -11.0,
23   "theme": { "bg": "#F0F0F0", "platform": "#C8C8C8", "blob": "#1478FF" },
24   "start": { "x": 80, "y": 220, "r": 26 },
25   "pattern": {
26     "type": "grid",
27     "rows": 3,
28     "cols": 4,
29     "startX": 80,
30     "startY": 100,
31     "platformWidth": 100,
32     "platformHeight": 12,
33     "spacingX": 60,
34     "spacingY": 80
35   },
36   "platforms": [
37     { "x": 0, "y": 324, "w": 640, "h": 36 }
38   ]
39 }
40 ]
41

```

Update the new grid level in levels.json so it has both a pattern for the grid AND a platforms array with just one platform for the ground floor at the bottom.

I'll open [levels.json](#) to confirm the Grid Climb level structure and edit it if needed.

Patching [levels.json](#) to fix a stray brace and ensure the Grid Climb level remains with both [pattern](#) and a single ground [platform](#).

Edited
levels.json+0-1

Edited
levels.json+1-3

Done — [levels.json](#) now contains the Grid Climb level with a [pattern](#) grid and a single ground [platform](#) at y:324. Want me to run the sketch to verify it loads?

```

35   35 |   spacing: .60
34   34 |   ],
35   -   "platforms": [
36   -     { "x": 0, "y": 324, "w": 640, "h": 36 }
37   -   ]
35+  |   "platforms": [{ "x": 0, "y": 324, "w": 640, "h": 36 }]
38   36   }
39   37   ]
40   38 }

```

Make sure the grid level is playable. The player should spawn at x: 80, y: 280 and be able to jump through the grid to reach the top.

I'll update [levels.json](#) to set the Grid Climb spawn to x:80, y:280 and track the change in the todo list.

Created 2 todos

Edited

levels.json+1-1

Completed: *Update Grid Climb spawn (1/2)*

Done — Grid Climb now spawns at x:80, y:280. Want me to run the sketch locally to verify playability?

```
  "gravity": 0.5,
  "jumpV": -11.0,
  "theme": { "bg": "#F0F0F0", "platform": "#C8C8C8", "blob": "#1478FF" },
  "start": { "x": 80, "y": 220, "r": 26 },
  "start": { "x": 80, "y": 280, "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 },
```

Add a goal on the top right so that when the blob reaches it it automatically starts the 'N' next level. also make it so that when the blob falls off the lowest level of platforms i have to restart since i "died"

I'll inspect [blobPlayer.js](#) and related files to find where to add goal detection and death handling.
I'll add a `setGoal` and goal drawing to [worldLevel.js](#), and update [sketch.js](#) to set the goal after resizing, detect goal collision to advance level, and detect falling off-screen to restart.

Edited
[worldLevel.js+25-0](#)

Read [sketch.js](#)

Edited
[sketch.js+27-1](#)

Checked [sketch.js](#), [worldLevel.js](#), [blobPlayer.js](#): no problems found

Progress update: added a top-right goal and death handling.

What I changed:

- [worldLevel.js](#): added `setGoal(canvasW, canvasH)` and draw logic for a goal rectangle.
- [sketch.js](#): after resizing the canvas I call `world.setGoal(W,H)`. In `draw()` I now:
 - restart the level if the blob falls below the canvas (`player.y - player.r > height`),
 - advance to the next level automatically when the blob overlaps the goal rectangle.

Next steps — pick one:

- I can make the goal visually fancier (icon/text) or change its size/margin.
- I can add a visible respawn/death message or a short delay before restarting.

Which would you like?

```
88+    backgroundColor(this.theme.bg),
89+    for (const p of this.platforms) {
90+      p.draw(color(this.theme.platform));
91+    }
92+    // Draw goal if present
93+    if (this.goal) {
94+      push();
95+      // Goal colour contrasts with platforms
96+      fill(255, 204, 0);
97+      noStroke();
98+      rect(this.goal.x, this.goal.y, this.goal.w, this.goal.h, 6);
99+      pop();
100+
101+  }
102+
103+ /**
104+ * Place a rectangular goal in the top-right of the level.
105+ * Call after canvas is resized so width/height are known.
106+ */
107+ setGoal(canvasW, canvasH, opts = {}) {
108+   const gw = opts.w ?? 36;
109+   const gh = opts.h ?? 36;
110+   const margin = opts.margin ?? 12;
111+   this.goal = {
112+     x: canvasW - gw - margin,
113+     y: margin,
114+     w: gw,
115+     h: gh,
116+   };
117+ }
118+
119+ // 2) Update and draw the player on top of the world
120+ // 2) Update the player (physics + collisions)
121+ player.update(world.platforms);
122+
123+ // 2a) Death: fell off bottom -> restart current level
124+ if (player.y - player.r > height) {
125+   loadLevel(levelIndex);
126+   return;
127+ }
128+
129+ // 2b) Goal: if player reaches goal, advance to next level
130+ if (world.goal) {
131+   const playerBox = {
132+     x: player.x - player.r,
133+     y: player.y - player.r,
134+     w: player.r * 2,
135+     h: player.r * 2,
136+   };
137+   if (overlapAABB(playerBox, world.goal)) {
138+     const next = (levelIndex + 1) % data.levels.length;
139+     loadLevel(next);
140+     return;
141+   }
142+ }
143+
144+ // 3) Draw player on top of world
145+ player.draw(world.theme.blob);
146+
147+ // 3) HUD
```

[paste worldLevel code] [paste levels code]
currently my issue is that my blob cannot jump around freely it is always blocked by the upper platforms and cannot progress from the lower ones without dying.
Give me some prompts to fix that with a 2D array.

Okay, starting fresh from your code! Here's what we're doing:

Goal: Use a 2D array to design your level layout where you control exactly which platforms appear.

Prompt 1:

In `levels.json`, in the Grid Climb level, change the pattern object to include a "layout" property. The layout should be a 2D array (array of arrays) where 1 means a platform exists and 0 means empty space. Make it look like this:

```
"layout": [
  [1, 0, 0, 1],
  [0, 1, 1, 0],
  [1, 0, 0, 1]
]
```

Remove the rows and cols properties since the layout array will determine the size.

Prompt 2:

In `WorldLevel.js`, modify the `generateGrid` method. Add a parameter called `layout` at the end. If `layout` exists, ignore the `rows` and `cols` parameters. Instead, loop through `layout` using nested for loops - outer loop goes through `layout.length` (rows), inner loop goes through `layout[r].length` (columns). Only create a platform when `layout[r][c]` equals 1.

Prompt 3:

In `WorldLevel.js` constructor, where it calls `this.generateGrid`, pass `p.layout` as the last argument so the layout array gets used.

That's it! After this, you can edit the 2D array in `levels.json` however you want to design your level.

In levels.json, in the Grid Climb level, change the pattern object to include a "layout" property. The layout should be a 2D array (array of arrays) where 1 means a platform exists and 0 means empty space. Make it look like this:

```
"layout": [
  [1, 0, 0, 1],
  [0, 1, 1, 0],
  [1, 0, 0, 1]
]
```

Remove the rows and cols properties since the layout array will determine the size.

```
23   "start": { "x": 80, "y": 280, "r": 26 },
24   "pattern": {
25     "type": "grid",
26     "rows": 3,
27     "cols": 4,
28     "startX": 80,
29     "startY": 100,
30     "platformWidth": 100,
31     "platformHeight": 12,
32     "spacingX": 60,
33     "spacingY": 80,
34     "spacingY": 80,
35     "layout": [
36       [1, 0, 0, 1],
37       [0, 1, 1, 0],
38       [1, 0, 0, 1]
39     ],
40     "platforms": [ { "x": 0, "y": 324, "w": 640, "h": 36 } ]
41   }
```

Updated — Grid Climb's `pattern` now includes the requested layout 2D array and no longer contains `rows` or `cols`. Want me to update `WorldLevel.generateGrid` to support reading a layout (so empty cells are skipped) and use it when present?

In WorldLevel.js, modify the generateGrid method. Add a parameter called layout at the end. If layout exists, ignore the rows and cols parameters. Instead, loop through layout using nested for loops - outer loop goes through `layout.length` (rows), inner loop goes through `layout[r].length` (columns). Only create a platform when `layout[r][c]` equals 1.

I'll update `worldLevel.js`: modify the constructor call to pass `p.layout`, and change `generateGrid` to accept `layout` and use it when present.

Edited
`worldLevel.js+21-7`
 Done — `generateGrid` now accepts a `layout` parameter and creates platforms only where `layout[r][c] === 1`. I also updated the constructor to pass `pattern.layout` into `generateGrid`.
 Next recommended step: run the sketch and press N to load the Grid Climb level and confirm the layout appears as intended.
 Want me to change the default loaded level to the grid for quick verification?

```

132 |     spacingY = 0,
133+    layout,
134  } {
135  const platforms = [];
136+ // If a layout (2D array) is provided, use it to determine platform locations
137+ if (layout && Array.isArray(layout)) {
138+   for (let r = 0; r < layout.length; r++) {
139+     const row = layout[r] || [];
140+     for (let c = 0; c < row.length; c++) {
141+       if (row[c] === 1) {
142+         const x = startX + c * (platformWidth + spacingX);
143+         const y = startY + r * (platformHeight + spacingY);
144+         platforms.push(new Platform({ x: x, y: y, w: platformWidth, h: platformHeight }));
145+       }
146+     }
147+   }
148+ } else {
149+   for (let r = 0; r < rows; r++) {
150+     for (let c = 0; c < cols; c++) {
151+       const x = startX + c * (platformWidth + spacingX);
152+       const y = startY + r * (platformHeight + spacingY);
153+       platforms.push(
154+         new Platform({ x: x, y: y, w: platformWidth, h: platformHeight }),
155+       );
156+     }
157+   }

```

In `WorldLevel.js` constructor, where it calls `this.generateGrid`, pass `p.layout` as the last argument so the layout array gets used.

I'll open [worldLevel.js](#) to confirm the `generateGrid` call includes `p.layout` as the last argument.
 Read [worldLevel.js](#)
 Confirmed — [worldLevel.js](#) now calls `this.generateGrid(..., p.layout)` (layout passed as last arg). Want me to make the grid level the default (`loadLevel(1)` in [sketch.js](#)) so it opens immediately?

i want. to be more like you can climb up using the left platforms and on the right you can also take an "elevator" moving platform

Ohhh! You want **moving platforms** too! That's a cool addition.

Let me break this down into steps:

What you need:

1. **Static platforms on the left** (checkerboard pattern for climbing)
2. **A moving platform on the right** (goes up and down like an elevator)

Here are more prompts:

Prompt 4:

Create a new file called MovingPlatform.js. Copy the Platform class code but add these extra properties in the constructor: startY, endY, speed (default 1), and direction (default 1). Add an update() method that moves the platform: `this.y += this.speed * this.direction`. When y reaches endY or startY, reverse direction by multiplying this.direction by -1.

Prompt 5:

In index.html, add this line after the Platform.js script tag:
`<script src="MovingPlatform.js"></script>`

Prompt 6:

In levels.json Grid Climb level, add a new property called "movingPlatforms" (separate from the pattern). Make it an array with one object: { "x": 500, "startY": 280, "endY": 120, "w": 80, "h": 12, "speed": 1.5 }

Prompt 7:

	In WorldLevel.js constructor, after the platforms are created, add code to check if levelJson.movingPlatforms exists. If it does, loop through it and create MovingPlatform instances, then add them to the this.platforms array using push.
	<p>Prompt 8:</p> <p>In sketch.js in the draw() function, before player.update(), add a loop: <code>for (const p of world.platforms) { if (p.update) p.update(); }</code> This will update any moving platforms.</p>
	Now you'll have both the 2D array layout AND a moving elevator platform!

In index.html, add this line after the Platform.js script tag:

```
<script src="MovingPlatform.js"></script>
```

I'll insert the [MovingPlatform.js](#) script tag right after the existing [Platform.js](#) script tag in [index.html](#).

Read [index.html](#)

Edited

index.html+1-0

Inserted `<script`

`src="MovingPlatform.js"></script>` after [platform.js](#) in [index.html](#).

Need me to convert any existing platforms in [levels.json](#) to [MovingPlatform](#) instances or update [WorldLevel](#) to instantiate moving platforms when pattern specifies movement?

In levels.json Grid Climb level, add a new property called "movingPlatforms" (separate from the pattern). Make it an array with one object: { "x": 500, "startY": 280, "endY": 120, "w": 80, "h": 12, "speed": 1.5 }

Added `movingPlatforms` to the Grid Climb level in [levels.json](#):

- One entry: { "x": 500, "startY": 280, "endY": 120, "w": 80, "h": 12, "speed": 1.5 }.

Next, would you like me to wire these into [WorldLevel](#) so they become [MovingPlatform](#) instances and are updated each frame?

```
32     "layout": [
33       [1, 0, 0, 1],
34       [0, 1, 1, 0],
35       [1, 0, 0, 1]
36     ],
37   },
38+   "movingPlatforms": [
39+     { "x": 500, "startY": 280, "endY": 120, "w": 80, "h": 12, "speed": 1.5 }
40+   ],
41   "platforms": [{ "x": 0, "y": 324, "w": 640, "h": 36 }]
42 }
43 ]
44 }
```

In `WorldLevel.js` constructor, after the platforms are created, add code to check if `levelJson.movingPlatforms` exists. If it does, loop through it and create `MovingPlatform` instances, then add them to the `this.platforms` array using `push`.

I'll patch `worldLevel.js` to construct `MovingPlatform` instances from `levelJson.movingPlatforms` and append them to `this.platforms`.

Edited
`worldLevel.js+17-0`

Done — `WorldLevel` now creates `MovingPlatform` instances from `levelJson.movingPlatforms` and appends them to `this.platforms`.

Next I can:

- Update the game loop to call `update()` on moving platforms each frame so they move, or
- Convert existing static platforms to moving ones for demo.

Which should I do?

```
    this.platforms = [...levelJson.platforms || []].map((p) => new Platform(p));
}

// Create any moving platforms defined in the level and append them
if (levelJson.movingPlatforms && Array.isArray(levelJson.movingPlatforms)) {
  for (const m of levelJson.movingPlatforms) {
    const mp = new MovingPlatform({
      x: m.x,
      y: m.startY ?? m.y ?? 0,
      w: m.w,
      h: m.h,
      startY: m.startY ?? m.y ?? 0,
      endY: m.endY ?? m.y ?? 0,
      speed: m.speed ?? 1,
      direction: m.direction ?? 1,
    });
    this.platforms.push(mp);
  }
}

/*
If you want the canvas to fit the world, you can infer width/height by
finding the maximum x+w and y+h across all platforms.
*/
```

In sketch.js in the draw() function, before player.update(), add a loop: for (const p of world.platforms) { if (p.update) p.update(); } This will update any moving platforms.

I'm going to patch [sketch.js](#) to update moving platforms each frame before the player updates.

Edited
sketch.js+4-1

Done — [sketch.js](#) now updates moving platforms each frame before the player update. Want me to run a quick lint/error check or start the sketch so you can test the moving platform?

```
41
42  function draw() {
43    // 1) Draw the world (background + platforms)
44    world.drawWorld();
45
46-   // 2) Update the player (physics + collisions)
46+   // 2) Update moving platforms (if any) then the player (physics + collisions)
47+   for (const p of world.platforms) {
48+     if (p.update) p.update();
49+
50     player.update(world.platforms);
51
52     // 2a) Death: fell off bottom -> restart current level
53     if (player.y - player.r > height) {
54       loadLevel(levelIndex);
55     }
56   }
```

right now my moving platforms are glitching up and down at the speed of light. make it so that it gradually moves up and down (aka takws 3 seconds to get to the top, 3 seconds to go back down, repeat)

I'm going to update [MovingPlatform.js](#) to use time-based movement (pixels per millisecond) so platforms take 3 seconds to travel between bounds and reverse smoothly.

Edited
MovingPlatform.js+28-9

Updated [MovingPlatform.js](#) to use time-based movement:

- Platforms now compute pixels/ms so travel from `startY` to `endY` in `travelTime` seconds (default 3s).
- `update()` uses `deltaTime` (p5's ms since last frame) for smooth motion and reverses at bounds.
- Backwards-compatible `speed` remains but is no longer used for motion.

Test by running the sketch — the moving platform should take ~3 seconds up and ~3 seconds down. Want me to make `travelTime` configurable from [levels.json](#) (e.g., add `travelTime` to the [movingPlatforms](#) entry)?

```

3  */
4
5  class MovingPlatform {
6    constructor({ x, y, w, h, startY = y, endY = y, speed = 1, direction }){ Keep Undo ⌂
7      constructor(x, y, w, h, startY = y, endY = y, speed = 1, direction, travelTime = 3){ Keep Undo ⌂
8        // Position is the top-left corner.
9        this.x = x;
10       this.y = y;
11
12       // Size (width/height).
13       this.w = w;
14       this.h = h;
15
16       // Movement bounds and state.
17       this.startY = startY;
18       this.endY = endY;
19
20       // Duration in seconds to travel from startY -> endY (default 3s).
21       this.travelTime = travelTime;
22
23       // Compute travel distance and speed in pixels per millisecond.
24       const distance = Math.abs(this.endY - this.startY);
25       this._speedPerMs = distance > 0 ? distance / (this.travelTime * 1000) : 0;
26
27       // Direction: if provided use it, otherwise choose sign so platform moves toward endY.
28       if (typeof direction !== "undefined") this.direction = direction;
29       else this.direction = this.endY >= this.startY ? 1 : -1;
30
31       // Keep a legacy numeric 'speed' property for compatibility (not used for movement).
32       this.speed = speed;
33       this.direction = direction;
34     }
35
36     // Move vertically between startY and endY.
37     update() { Keep Undo ⌂
38       this.y += this.speed * this.direction;
39     }

```

ok maybe dont make it so taht the blob immediately dies after i falls off platform, but make it so that if it falls like 5px distance off screen it restarts?

I'll update the death detection in [sketch.js](#) to add a 5-pixel grace buffer before restarting.

Edited
sketch.js+2-2

Done — blob now gets a 5-pixel grace buffer. It restarts only after falling 5 pixels past the canvas bottom, giving you a brief window to recover if you almost miss a platform.

once the moving platform reaches the bottom, it teleports my blob off and kills it. I don't want it to do that I want my blob to have the ability to ride the moving platform all the way to the finish line.

The issue is that the blob isn't moving with the platform. I'll add code in [sketch.js](#) to "carry" the blob when it's standing on a moving platform.

Edited

sketch.js+29-0

Fixed — the blob now moves with the platform by tracking the platform's Y position before/after update and applying the delta to the player if it's standing on a moving platform. The blob should now ride the platform smoothly to the bottom and back up.

```
45 // 2) Update moving platforms (if any) then the player (physics + collisions)
46 const platformYBefore = {};
47+ for (const p of world.platforms) {
48+   platformYBefore[world.platforms.indexOf(p)] = p.y;
49+
50+
51+
52+ for (const p of world.platforms) {
53+   if (p.update) p.update();
54+
55+
56+ // Carry blob along with moving platforms
57+ for (let i = 0; i < world.platforms.length; i++) {
58+   const p = world.platforms[i];
59+   if (p.update && typeof platformYBefore[i] !== "undefined") {
60+     const yDelta = p.y - platformYBefore[i];
61+     // Check if blob is on this platform (approximately)
62+     const playerBox = {
63+       x: player.x - player.r,
64+       y: player.y - player.r,
65+       w: player.r * 2,
66+       h: player.r * 2,
67+     };
68+     // If player overlaps platform and is roughly on top, move with it
69+     if (
70+       overlapAABB(playerBox, p) &&
71+       player.y + player.r >= p.y &&
72+       player.y - player.r <= p.y + p.h + 5
73+     ) {
74+       player.y += yDelta;
75+     }
76+
77+
78+
79+   }
80+ }
81+
82+ player.update(world.platforms);
```