

Process & Decision Documentation

Name: Giang Tran

Role(s): Sole Coder

Primary responsibility for this work: All tasks for Week 4 Side Quest

Goal of Work Session

Implement a tile-based level generator using 2D arrays and loops to dynamically place platforms and obstacles, then add level progression mechanics with goal zones.

Tools, Resources, or Inputs Used

- Claude Code (for coding assistance)
- Claude (for technical consultation)
- Lecture Notes (Week 4)
- Existing platformer codebase (Week 2 blob example)

GenAI Documentation

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

Date Used: Feb 9 2026

Tool Disclosure: tool name + model/version (example, ChatGPT 5.2)

Purpose of Use: Technical implementation assistance for translating conceptual designs into working code, debugging collision detection, and structuring class-based architecture.

Summary of Interaction: The tool helped implement the levelGenerator.js file structure, suggested collision detection logic for goal zones, and provided syntax for array manipulation in the tile map system. It also assisted with integrating the new generator into the existing class structure.

Human Decision Point(s):

- Rejected initial suggestion to use separate collision arrays for obstacles; instead kept unified Platform class for simpler code maintenance

- Modified tile size from suggested 32px to 40px after testing showed better visual proportion with blob size
- Redesigned goal zone positioning to right side of levels rather than centered, improving flow
- Chose to remove manual level-switching (key N) instruction from HUD after implementing auto-progression
- Decided to use minimal tile types (0-3) rather than expanded set suggested by tool, keeping complexity appropriate for assignment scope

Integrity & Verification Note: Verified collision detection by playtesting edge cases (corners, fast movement). Checked that tile-to-pixel conversion matched existing platform coordinate system. Confirmed goal detection worked reliably across different blob velocities. Cross-referenced array structure with Week 3 lecture examples on nested loops.

Scope of GenAI Use: GenAI did not contribute to: level design decisions (tile layouts, platform placement patterns), physics tuning values, theme color choices, the conceptual approach to using 2D arrays for level generation, or determining which features to prioritize. The creative and architectural decisions were human-driven.

Limitations or Misfires: Tool initially suggested complex pathfinding for obstacle placement, which was unnecessary for assignment requirements. Recommended adding more tile types (ice, bounce pads, enemies) when simplicity was preferred. Provided goal collision using separate AABB function when reusing existing overlapAABB was cleaner.

Summary of Process (Human + Tool)

- Started by reviewing the existing platformer code structure and sketching a tile map grid on paper to plan level layouts. Experimented with different TILE_SIZE values (32, 40, 48px) by hardcoding platforms and playtesting blob movement at each scale, ultimately settling on 40px.
- Created the initial tile map array with ground platforms and floating platforms, then iteratively adjusted positions after testing revealed gaps that were too wide for jumping. Implemented nested loops to convert the 2D array into platform objects, debugging coordinate calculations when platforms appeared offset.
- Added obstacles (type 2) with distinct red coloring and integrated the generator with the existing level system. Experimented with goal zone placement (center, left, right side) before settling on right-side positioning for better flow.

Implemented auto-progression with collision detection, refining the timing after initial tests showed it triggered too quickly.

- Tested the complete gameplay loop across both levels to verify smooth transitions without memory issues.

Decision Points & Trade-offs

Decision 1: Unified Platform Class for Obstacles

- Options considered: Create separate Obstacle class with different collision behavior, or reuse Platform class
- What changed: Kept Platform class for both platforms and obstacles, differentiating only by color and storage array
- Why: Obstacles needed identical collision physics. Separate class would duplicate code without functional benefit. Simpler architecture easier to debug and maintain. Aligns with DRY principle from lecture.

Decision 2: Goal Zone as Auto-Trigger vs. Key Press

- Options considered: Require player to press key at goal (like "E to exit"), or automatic progression on contact
- What changed: Implemented automatic progression with immediate level load on goal collision
- Why: Smoother gameplay flow—no interruption to momentum. Matches modern platformer conventions. Goal zone already provides clear visual target, extra input redundant. Testing showed automatic felt more satisfying.

Verification & Judgement

- Playtesting: Ran through both levels multiple times at different speeds to test goal detection reliability. Verified blob couldn't skip goal zone by jumping over. Tested edge cases like landing partially on goal.
- Code Review: Compared tile-to-pixel conversion logic against Week 3 lecture examples on 2D array indexing. Confirmed nested loop structure matched course material.
- Comparison with Course Concepts: Verified use of arrays and loops satisfied "dynamically place tiles" requirement. Checked that level data structure aligned with JSON loading examples from Week 4 lecture

- Visual Testing: Confirmed color contrast between platforms (gray), obstacles (red), and goal (gold) provided clear gameplay readability.

Limitations, Dead Ends, or Open Questions

- Limitation: Goal collision sometimes triggers when blob edge barely touches goal edge. Could add minimum overlap threshold, but current implementation works adequately for assignment scope.
- Dead End: Tried implementing collectible items in tile map (type 4) but collision detection became complex with multiple check types. Removed feature to keep code focused on core requirements.

Appendix

Prompt	Output
I need to add a level generator to my platformer game. I have existing Platform, BlobPlayer, and WorldLevel classes. Can you show me how to structure a 2D tile map array for level generation?	<p>I can help you create a tile map structure. First, let me look at your existing code to understand the architecture.</p> <p>[Reviews existing files]</p> <p>Based on your code, here's a basic tile map structure:</p> <pre>```javascript const TILE_SIZE = 32; const levelMap = [[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [1, 1, 1, 1, 1, 1]]; // 0 = empty // 1 = platform ``` </pre> <p>You'll need nested loops to convert this to platform objects:</p>

```
```javascript
for (let row = 0; row < levelMap.length;
row++) {
 for (let col = 0; col <
levelMap[row].length; col++) {
 const tileType = levelMap[row][col];
 if (tileType === 1) {
 const x = col * TILE_SIZE;
 const y = row * TILE_SIZE;
 // Create platform
 }
 }
}
```
```

```

USER:

the tile size seems small. my blob radius is 26, what would work better?

CLAUDE:

Good point. For a blob with radius 26, I'd recommend TILE\_SIZE between 32-48 pixels. Let's try 40px - it gives you enough space for the blob to move through while maintaining visual clarity.

USER:

ok let me test that. also can you help me create the actual levelGenerator.js file? I want it to have different tile types - platforms, obstacles, and floating platforms

CLAUDE:

I'll create a levelGenerator.js file with multiple tile types:

```
```javascript
const TILE_SIZE = 40;

function generateLevelFromMap(tileMap)
{
  const platforms = [];
```
```

```

```

const obstacles = [];

for (let row = 0; row < tileMap.length; row++) {
    for (let col = 0; col < tileMap[row].length; col++) {
        const tileType = tileMap[row][col];
        const x = col * TILE_SIZE;
        const y = row * TILE_SIZE;

        if (tileType === 1) {
            platforms.push({ x, y, w: TILE_SIZE,
h: TILE_SIZE });
        } else if (tileType === 2) {
            obstacles.push({ x, y, w: TILE_SIZE,
h: TILE_SIZE });
        } else if (tileType === 3) {
            platforms.push({ x, y, w: TILE_SIZE,
h: TILE_SIZE });
        }
    }
}

return { platforms, obstacles };
}
...

```

Would you like me to create a sample level map as well?

```

```javascript
function buildGeneratedLevel() {
 const levelMap = [
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 3, 3, 0, 0, 0, 0, 0, 3, 3, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
];
}

```

```
[3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3,
0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0],
[1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1,
1]
];

const generated =
generateLevelFromMap(levelMap);

return {
 name: "Generated Level",
 gravity: 0.65,
 jumpV: -12.0,
 theme: { bg: "#2C3E50", platform:
"#27AE60", blob: "#E74C3C" },
 start: { x: 60, y: 240, r: 26 },
 platforms: generated.platforms,
 obstacles: generated.obstacles
};
}
```
=====
```

SESSION 2: Testing and Debugging Tile Size

```
```javascript
function preload() {
 data = loadJSON("levels.json");
}

function setup() {
 player = new BlobPlayer();
```

	<pre>// Add generated level data.levels.push(buildGeneratedLevel());  loadLevel(0); // ... } ``` </pre>
the obstacles aren't showing up differently from platforms. they're both gray	<p>You need to update WorldLevel.js to render obstacles separately. Add this to the drawWorld() method:</p> <pre>```javascript drawWorld() {   background(color(this.theme.bg));   for (const p of this.platforms) {     p.draw(color(this.theme.platform));   }   // Draw obstacles in red   for (const o of this.obstacles) {     o.draw(color("#E74C3C"));   } } ``` </pre> <p>Also make sure WorldLevel constructor stores obstacles:</p> <pre>```javascript this.obstacles = (levelJson.obstacles    []).map((o) =&gt; new Platform(o)); ``` </pre>
how should I handle level progression? I'm thinking a goal zone the player has to reach	<p>Good idea. You can add a goal object to each level:</p> <pre>```javascript goal: { x: 580, y: 100, w: 50, h: 50 } ``` </pre> <p>Then in sketch.js draw(), check for collision:</p>

```
```javascript
function playerReachedGoal() {
  const g = world.goal;
  const px = player.x - player.r;
  const py = player.y - player.r;
  const pw = player.r * 2;
  const ph = player.r * 2;

  return px < g.x + g.w && px + pw > g.x
  &&
    py < g.y + g.h && py + ph > g.y;
}

function draw() {
  world.drawWorld();
  player.update(world.platforms);
  player.draw(world.theme.blob);

  if (playerReachedGoal()) {
    loadNextLevel();
  }
}
```

```

USER:

wait I already have an overlapAABB function in blobPlayer.js. can I reuse that instead of writing a new collision check?

CLAUDE:

Good catch! Yes, you can create a simple box object and use the existing function:

```
```javascript
function playerReachedGoal() {
  const playerBox = {
    x: player.x - player.r,
    y: player.y - player.r,
    w: player.r * 2,
    h: player.r * 2
  };
  return overlapAABB(playerBox,
world.goal);
}
```

	<pre> } ``` </pre> <p>This is cleaner and reuses your tested code.</p>
Add a second level that loads automatically when the first is finished	<p>I'll add a goal zone to each level that triggers the next level when reached.</p> <p>[Read 2 files]</p> <p>I'll add goal zones to both levels and auto-load the next level when reached.</p> <p>[Update(levels.json) - Added 2 lines, removed 1 line]</p> <pre> 2 "schemaVersion": 1, 3 "levels": [4 { 5 "name": "Intro Steps", 5 + "name": "Level 1 - Intro Steps", 6 "gravity": 0.65, 7 "jumpV": -11.0, 8 "theme": { "bg": "#F0F0F0", "platform": "#C8C8C8", "blob": "#1478FF" }, 9 "start": { "x": 80, "y": 220, "r": 26 }, 10 "goal": { "x": 560, "y": 94, "w": 50, "h": 50 }, 11 "platforms": [12 { "x": 0, "y": 324, "w": 640, "h": 36 }, 13 { "x": 120, "y": 254, "w": 120, "h": 12 }, </pre> <p>[Update(levelGenerator.js) - Added 2 lines, removed 1 line]</p> <pre> 61 const generated = generateLevelFromMap(levelMap); 62 63 return { 64 name: "Generated Level", 64 + name: "Level 2 - Generated", </pre>

```
65     gravity: 0.65,
66     jumpV: -12.0,
67     theme: { bg: "#2C3E50",
platform: "#27AE60", blob: "#E74C3C" },
68     start: { x: 60, y: 240, r: 26 },
69 +   goal: { x: 560, y: 80, w: 50, h: 50
},
70     platforms: generated.platforms,
71     obstacles: generated.obstacles,
72   };
```

[Update(worldLevel.js) - Added 3 lines]

```
53
54 // Convert obstacle objects into
Platform instances (for collision +
drawing).
55   this.obstacles =
(levelJson.obstacles || []).map((o) => new
Platform(o));
56 +
57 + // Goal zone (finish area).
58 +   this.goal = levelJson.goal || { x:
580, y: 100, w: 50, h: 50 };
59 }
60
61 /*
```

[Update(worldLevel.js) - Added 3 lines]

```
85   for (const o of this.obstacles) {
86     o.draw(color("#E74C3C"));
87   }
88 + // Draw goal zone in gold
89 +   fill(color("#FFD700"));
90 +   rect(this.goal.x, this.goal.y,
this.goal.w, this.goal.h);
91 }
92 }
```

Now update sketch.js to check for goal collision and auto-load the next level.

[Update(sketch.js) - Added 24 lines, removed 2 lines]

```
50   player.update(world.platforms);
```

```

51   player.draw(world.theme.blob);
52
53 - // 3) HUD
53 + // 3) Check if player reached the
goal
54 + if (playerReachedGoal()) {
55 +   loadNextLevel();
56 +
57 +
58 + // 4) HUD
59   fill(0);
60   text(world.name, 10, 18);
61 - text("Move: A/D or ←/→ • Jump:
Space/W/↑ • Next: N", 10, 36);
61 + text("Move: A/D or ←/→ • Jump:
Space/W/↑", 10, 36);
62 }
63
64 +// Check if player overlaps the goal
zone
65 +function playerReachedGoal() {
66 + const g = world.goal;
67 + const px = player.x - player.r;
68 + const py = player.y - player.r;
69 + const pw = player.r * 2;
70 + const ph = player.r * 2;
71 +
72 + return px < g.x + g.w && px + pw
> g.x && py < g.y + g.h && py + ph > g.y;
73 +}
74 +
75 +// Load next level (loops back to
first)
76 +function loadNextLevel() {
77 + const next = (levelIndex + 1) %
data.levels.length;
78 + loadLevel(next);
79 +}
80 +
81 function keyPressed() {
82   // Jump keys
83   if (key === " " || key === "W" || key
=== "w" || keyCode === UP_ARROW) {

```

Done! Refresh your browser at
<http://localhost:8000>

Changes:

- Gold goal zone appears on the right side of each level
- Auto-loads next level when the player touches the goal
- Levels loop: Level 1 → Level 2 → Level 1 → ...

Reach the gold square to advance to the next level.