

Process & Decision Documentation

GBDA 302: Side Quest 4

Student Name: Min Htet Naing, dmnaing (21008098)

Project/Assignment Decisions

Description

For Side Quest 4, I modified the provided Week 4 JSON-driven platformer example to demonstrate data-driven level generation using classes and loops. My goal was to make the level structure and win condition configurable through JSON rather than hard coded in the sketch.

Key Decision / Change

I chose to build on **Example 5** because it already integrated JSON-based level data, platform objects, and a blob player with physics. Instead of progressing levels when the player reached the edge of the screen, I added a door object defined in the JSON level data. When the player overlaps with the door, the sketch automatically loads the next level. This made the win condition clearer and tied progression directly to the level structure.

Another key decision was to keep all gameplay logic separate from visual polish. Level structure and progression are handled through JSON and class-based logic, while visual improvements were kept cosmetic to avoid changing the learning focus of the assignment.

GenAI Documentation (Used)

Date Used: Feb 10th, 2026

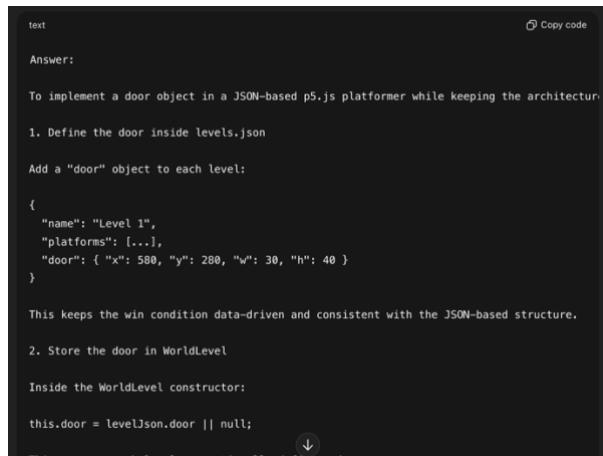
Tool Disclosure: ChatGPT (GPT-5.2 Thinking)

ChatGPT was used to clarify how to implement a door-based win condition within the existing class structure and to confirm that the modification aligned with the Week 4 JSON + class architecture. It also assisted with minor refactoring suggestions for sketch.js and WorldLevel.js. All final code was reviewed, implemented, and tested manually in VS Code and on GitHub Pages.

Appendix

Prompt 1

“How can I implement a door object in a JSON-based p5.js platformer and detect player overlap to automatically load the next level?”



The screenshot shows a text-based AI response. At the top left is a 'text' button, and at the top right is a 'Copy code' button. The text starts with 'Answer:' followed by a detailed explanation and code snippets. The code shows how to define a 'door' object in a JSON file and store it in a constructor. A note at the bottom indicates that this ensures each level can optionally define a door.

```
text
Copy code

Answer:

To implement a door object in a JSON-based p5.js platformer while keeping the architecture clean, follow these steps:

1. Define the door inside levels.json

Add a "door" object to each level:



```
{
 "name": "Level 1",
 "platforms": [...],
 "door": { "x": 580, "y": 280, "w": 30, "h": 40 }
}
```



This keeps the win condition data-driven and consistent with the JSON-based structure.

2. Store the door in WorldLevel

Inside the WorldLevel constructor:



```
this.door = levelJson.door || null;
```



↓



This ensures each level can optionally define a door.


```

Answer:

2. Store the door in WorldLevel

```
Inside the WorldLevel constructor:  
this.door = levelJson.door || null;  
  
This ensures each level can optionally define a door.
```

3. Draw the door in drawWorld()

```
Inside WorldLevel.drawWorld():  
  
if (this.door) {  
    fill(120, 80, 40);  
    rect(this.door.x, this.door.y, this.door.w, this.door.h);  
}  
  
Rendering stays inside WorldLevel because it is part of the world geometry.
```

4. Detect player overlap in sketch.js

```
Win-condition logic should NOT go in WorldLevel. It belongs in sketch.js since that file handles the game loop.
```

Add a helper function:

```
function playerTouchesDoor(player, door) {  
    return (  
        player.x + player.r > door.x &&  
        player.x - player.r < door.x + door.w &&  
        player.y + player.r > door.y &&  
        player.y - player.r < door.y + door.h  
    );  
}  
  
Then inside draw():  
  
if (world.door && playerTouchesDoor(player, world.door)) {  
    const next = (levelIndex + 1) % data.levels.length;  
    loadLevel(next);  
}  
  
This keeps:  
- Data in JSON  
- Rendering in WorldLevel  
- Gameplay flow in sketch.js
```

This structure aligns with the Week 4 JSON + class-based architecture.

Prompt 2

“Where should squash/stretch animation be added inside a Blob Player class without affecting collision or physics logic?”

Answer:

text

Answer:

```
Squash and stretch animation should be added inside the draw() method of the BlobPlayer class.  
  
Collision detection and physics calculations rely on the player's position (x, y), radius and velocity (vx, vy).  
  
To avoid affecting physics:
```

- Do NOT modify:
 - this.x
 - this.y
 - this.r
 - this.vx / this.vy
 - collision calculations
- Apply squash/stretch using p5.js transforms inside draw() only.

```
Example structure:  
draw(colourHex) {  
    const squash = this.onGround ? 1.1 : 0.95;
```

```
push();  
translate(this.x, this.y);  
scale(1, squash);  
  
fill(color(colourHex));  
beginShape();  
  
for (let i = 0; i < this.points; i++) {  
    const a = (i / this.points) * TAU;  
    const rr = this.r; // keep radius logic unchanged  
    vertex(cos(a) * rr, sin(a) * rr);  
}  
  
endShape(CLOSE);  
pop();  
}  
  
Explanation:  
- push() and pop() isolate the transformation.  
- translate() moves the origin to the blob's center.  
- scale() modifies only the drawing transform.  
- Physics and collision remain unaffected because no state values are changed.
```

This keeps animation purely visual and aligns with the Week 4 class-based architecture.

Prompt 3

“Refactor my sketch.js to use door-based level progression instead of screen-edge detection while keeping the architecture modular.”

Answer:

text

Answer:

To refactor sketch.js to use door-based level progression instead of screen-edge detection:

- JSON defines the door data
- WorldLevel renders the door
- sketch.js controls game flow

1. Remove screen-edge detection

Delete or comment out any logic such as:

```
if (player.x > width) {
    loadLevel(next);
}
```

This logic is tied to screen position rather than level structure.

2. Ensure door data is defined in levels.json

Each level should include:

```
"door": { "x": 580, "y": 280, "w": 30, "h": 40 }
```

3. Confirm WorldLevel stores door data

In the constructor:

```
this.door = levelJson.door || null;
```

Rendering of the door remains inside drawWorld().

4. Add a collision helper in sketch.js

```
function playerTouchesDoor(player, door) {
    return (
        player.x + player.r > door.x &&
        player.x - player.r < door.x + door.w &&
        player.y + player.r > door.y &&
        player.y - player.r < door.y + door.h
    );
}
```

5. Trigger level progression inside draw()

```
if (world.door && playerTouchesDoor(player, world.door)) {
    const next = (levelIndex + 1) % data.levels.length;
    loadLevel(next);
}
```

This approach keeps:

- Level structure in JSON
- Rendering logic in WorldLevel
- Progression logic in sketch.js
- Physics and collision inside BlobPlayer

The architecture remains modular and aligned with the Week 4 JSON + class-based design pa