

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?”

text  Copy code

Answer:

To implement a door object in a JSON-based p5.js platformer while keeping the architecture:

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.

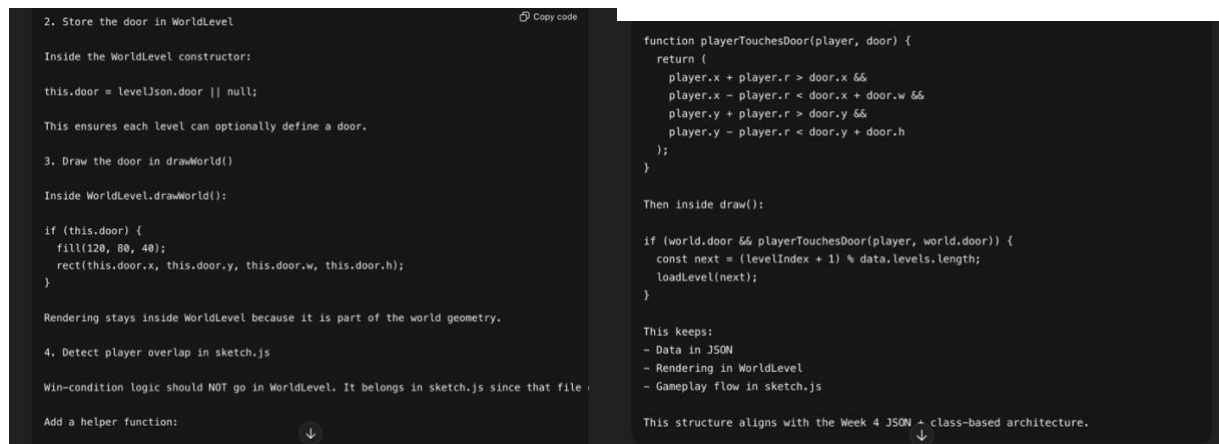
2. Store the door in WorldLevel

Inside the WorldLevel constructor:

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

 This ensures each level can optionally define a door.

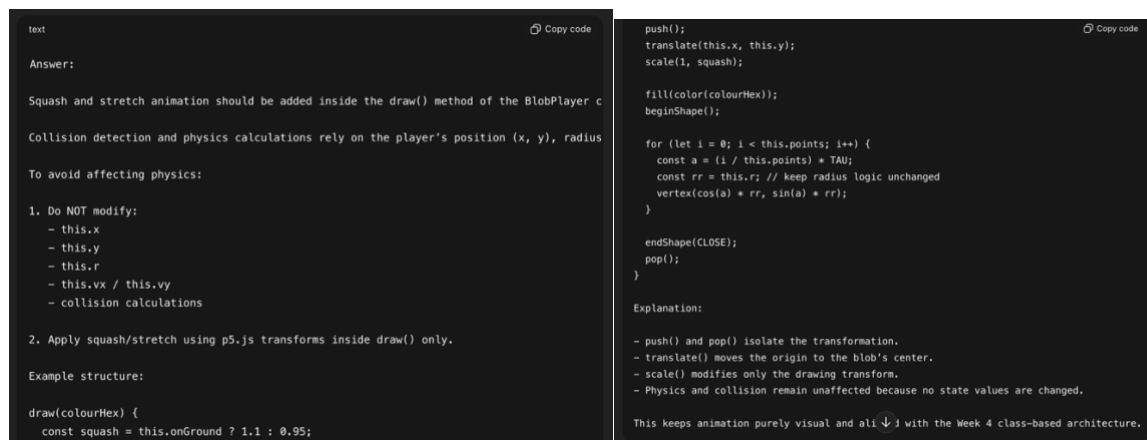
Answer:



Prompt 2

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

Answer:



Prompt 3

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

Answer:

text

Copy code

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": 500, "y": 200, "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