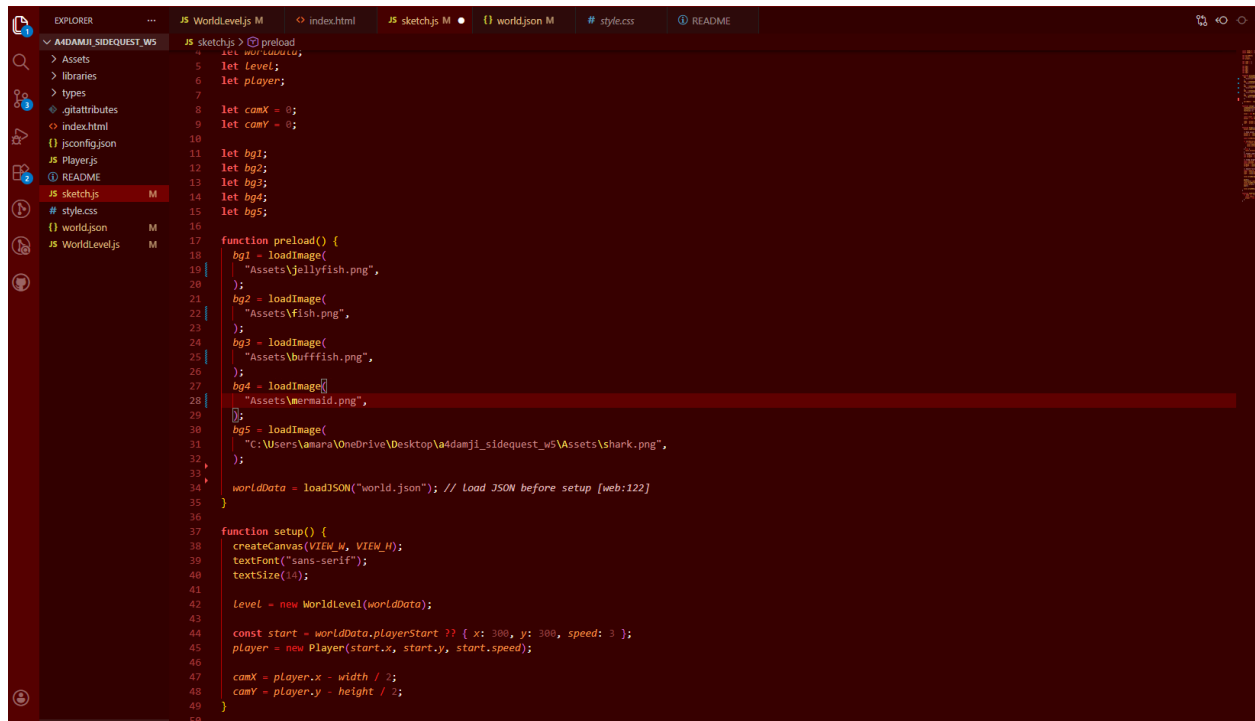


Process & Decision Documentation

Project/Assignment Decisions

For this assignment I decided to have images of various ocean scenes as the background. When implementing this I found nothing would load properly. After some troubleshooting I compared my code to a previous sidequest where I used images to see what I did wrong. That's when I found my error, I had messed up the preload. I used the copy path shortcut to load my images. These paths go from the computer to one drive, to desktop, then to the png. The code wasn't able to follow this path. I realized I had to shorten it to just the folder in the code and the image itself. I also realized the dashes that came with the path were in the wrong direction and messed up with the code's ability to read the file name. Once the right path was in the code started working successfully.

Fixing image errors



```
1  sketch.js > preload
2  let worldData;
3  let level;
4  let player;
5
6  let camX = 0;
7  let camY = 0;
8
9  let bg1;
10 let bg2;
11 let bg3;
12 let bg4;
13 let bg5;
14
15 function preload() {
16   bg1 = loadImage(
17     "Assets\\jellyfish.png",
18   );
19   bg2 = loadImage(
20     "Assets\\fish.png",
21   );
22   bg3 = loadImage(
23     "Assets\\bufffish.png",
24   );
25   bg4 = loadImage(
26     "Assets\\mermaid.png",
27   );
28   bg5 = loadImage(
29     "C:\\Users\\amara\\OneDrive\\Desktop\\a4damji_sidequest_ws\\Assets\\shark.png",
30   );
31   worldData = loadJSON("world.json"); // Load JSON before setup [web:122]
32 }
33
34 function setup() {
35   createCanvas(VIEW_W, VIEW_H);
36   textFont("sans-serif");
37   textSize(16);
38
39   level = new WorldLevel(worldData);
40
41   const start = worldData.playerStart ?? { x: 300, y: 300, speed: 3 };
42   player = new Player(start.x, start.y, start.speed);
43
44   camX = player.x - width / 2;
45   camY = player.y - height / 2;
46 }
```

GenAI Documentation

Date Used: Saturday February 21st

Tool Disclosure: ChatGPT

Purpose of Use: I used Chatgpt to assist with creating the changing background feature

Summary of Interaction: I asked Chatgpt how I would go about to create a background that changes with every interaction with the obstacle in the code

Human Decision Point(s): I had chatgpt use colours to test if the code it makes works. After I added the images and the final details myself.

Integrity & Verification Note: I reviewed generated code to ensure it didn't clash with pre existing code and ensured it assisted in reaching my final goal.

Limitations or Misfires: Because it didn't have a full picture of the entire code its implementation instructions were confusing to follow. I would go through and manually undo and edit to get a working product,

Summary of Process (Human + Tool) The process involved getting a base from chatgpt. There I edited and adjusted components to fit my vision. After I figured out how to add the code I added and removed code accordingly myself till I was satisfied with the results.

Verification & Judgement: When using pre-existing code chatGPT never gets the full picture especially when it is connected by different pages. It makes up new variables and repeats code. This can make implementation confusing and hard to understand what it tries to make.

Appendix

Me: I have a code divided into different pages and json files

right now the code is a blank world with a moveable blue character and a square in the middle

I want to do something where when the character hits the square the background changes

I plan to have images but as the moment to make sure it works I wish to use different colours as well make the square appear in a different location when hit

How would I go about this?

I have a json for the world, a js for the player and for the world level, and for drawing everything

ChatGPT: This is a great place to add a simple state + collision trigger system. Since your project is already modular (world JSON, player JS, world level JS, draw file), you'll want the *world* to control:

1. Background color
 2. Square position
 3. What happens when collision occurs
-



Core Idea

When:

player collides with square

Then:

- change background color
- move square to new position

You do this by:

- Detecting collision
- Calling a world method like `world.onSquareHit()`
- Updating world properties



Step 1: Track Background in World

In your WorldLevel.js, add:

```
class WorldLevel {
  constructor(data) {
    this.backgroundColor = data.backgroundColor || "lightgray";

    this.square = {
      x: data.square.x,
      y: data.square.y,
      size: data.square.size
    };

    this.colors = ["lightblue", "lightgreen", "lightpink", "lavender"];
  }

  draw() {
    background(this.backgroundColor);

    fill("black");
    rect(this.square.x, this.square.y, this.square.size, this.square.size);
  }
}
```



Step 2: Add a Collision Check Function

Still inside `WorldLevel`:

```
checkCollision(player) {
  return (
    player.x < this.square.x + this.square.size &&
    player.x + player.size > this.square.x &&
    player.y < this.square.y + this.square.size &&
    player.y + player.size > this.square.y
  );
}
```

This assumes your player is a square. If it's a circle, tell me and I'll adjust it.

Step 3: What Happens When Hit

Add this method:

```
onSquareHit() {  
  // Change background color randomly  
  let randomIndex = floor(random(this.colors.length));  
  this.backgroundColor = this.colors[randomIndex];  
  
  // Move square to new random position  
  this.square.x = random(50, width - 50);  
  this.square.y = random(50, height - 50);  
}
```

Step 4: Trigger It in draw()

In your main `draw()` file:

```
function draw() {  
  world.draw();  
  player.update();  
  player.draw();  
  
  if (world.checkCollision(player)) {  
    world.onSquareHit();  
  }  
}
```

Important Problem You'll Run Into

Without protection, this will trigger EVERY FRAME while touching.

That means background will flicker wildly.



Fix: Add a Cooldown Flag

Inside `WorldLevel`:

```
this.justHit = false;
```

Update collision logic:

```
if (world.checkCollision(player)) {  
  if (!world.justHit) {  
    world.onSquareHit();  
    world.justHit = true;  
  }  
} else {  
  world.justHit = false;  
}
```

Now it only triggers once per touch.



Later When You Add Images

Instead of:

```
background(this.backgroundColor);
```

You can do:

```
image(this.backgroundImage, 0, 0, width, height);
```

And swap images inside `onSquareHit()`.



Optional: Control It from JSON

In your JSON:

```
{  
  "backgroundColor": "lightblue",  
  "square": {  
    "x": 200,  
    "y": 200,  
    "size": 50  
  }  
}
```

Then your world stays data-driven.



Architecture Tip (Important)

Keep:

- Player → movement only
- World → environment + triggers
- Main draw → connects them

You're building this correctly for scaling into a real game.

If you'd like, send:

- Your player structure (square or blob?)
- Your world JSON format

And I'll tailor it exactly to your code structure.

Me: current code

player code:

```
class Player {  
  constructor(x, y, speed) {  
    this.x = x;  
    this.y = y;  
    this.s = speed ?? 3;  
  }  
  
  updateInput() {  
    const dx =  
      (keyIsDown(RIGHT_ARROW) || keyIsDown(68)) -  
      (keyIsDown(LEFT_ARROW) || keyIsDown(65));  
  
    const dy =  
      (keyIsDown(DOWN_ARROW) || keyIsDown(83)) -  
      (keyIsDown(UP_ARROW) || keyIsDown(87));  
  
    const len = max(1, abs(dx) + abs(dy));  
    this.x += (dx / len) * this.s;  
    this.y += (dy / len) * this.s;  
  }  
}
```



```
draw() {  
  fill(50, 110, 255);  
  noStroke();  
  rect(this.x - 12, this.y - 12, 24, 24, 5);  
}  
}
```

world json (the obstacle is the square)

```
{  
  "schemaVersion": 1,  
  "world": {  
    "w": 2400,  
    "h": 1600,  
    "bg": [235, 235, 235],  
    "gridStep": 160  
  },  
  "camera": {  
    "lerp": 0.12  
  },  
  "playerStart": {  
    "x": 300,  
    "y": 300,  
    "speed": 3  
  }  
}
```

```
},  
  "obstacles": [{ "x": 1200, "y": 800, "w": 40, "h": 40, "r": 10 }]  
}
```

wordlevel

```
class WorldLevel {  
  constructor(json) {  
    this.schemaVersion = json.schemaVersion ?? 1;  
  
    this.w = json.world?.w ?? 2400;  
    this.h = json.world?.h ?? 1600;  
    this.bg = json.world?.bg ?? [235, 235, 235];  
    this.gridStep = json.world?.gridStep ?? 160;  
  
    this.obstacles = json.obstacles ?? [];  
  
    // NEW: camera tuning knob from JSON (data-driven)  
    this.camLerp = json.camera?.lerp ?? 0.12;  
  }  
  
  drawBackground() {  
    background(220);  
  }  
}
```

```

drawWorld() {
  noStroke();

  fill(this.bg[0], this.bg[1], this.bg[2]);

  rect(0, 0, this.w, this.h);


  stroke(245);

  for (let x = 0; x <= this.w; x += this.gridStep) line(x, 0, x, this.h);
  for (let y = 0; y <= this.h; y += this.gridStep) line(0, y, this.w, y);


  noStroke();

  fill(170, 190, 210);

  for (const o of this.obstacles) rect(o.x, o.y, o.w, o.h, o.r ?? 0);
}

```

```

drawHUD(player, camX, camY) {
  noStroke();

  fill(20);

  text("Ocean Exploration", 12, 20);

  text(
    "camLerp(JSON): " +
    this.camLerp +
    " Player: " +
    (player.x | 0) +
    ", " +

```

```
(player.y | 0) +  
" Cam: " +  
(camX | 0) +  
"," +  
(camY | 0),  
12,  
40,  
);  
}  
}
```

Sketch

```
const VIEW_W = 800;
```

```
const VIEW_H = 480;
```

```
let worldData;
```

```
let level;
```

```
let player;
```

```
let camX = 0;
```

```
let camY = 0;
```

```
function preload() {
```

```
worldData = loadJSON("world.json"); // load JSON before setup [web:122]
}
```

```
function setup() {
  createCanvas(VIEW_W, VIEW_H);
  textFont("sans-serif");
  textSize(14);

  level = new WorldLevel(worldData);

  const start = worldData.playerStart ?? { x: 300, y: 300, speed: 3 };
  player = new Player(start.x, start.y, start.speed);

  camX = player.x - width / 2;
  camY = player.y - height / 2;
}
```

```
function draw() {
  player.updateInput();

  // Keep player inside world
  player.x = constrain(player.x, 0, level.w);
  player.y = constrain(player.y, 0, level.h);
```

```
// Target camera (center on player)

let targetX = player.x - width / 2;

let targetY = player.y - height / 2;


// Clamp target camera safely

const maxCamX = max(0, level.w - width);

const maxCamY = max(0, level.h - height);

targetX = constrain(targetX, 0, maxCamX);

targetY = constrain(targetY, 0, maxCamY);


// Smooth follow using the JSON knob

const camLerp = level.camLerp; // ← data-driven now

camX = lerp(camX, targetX, camLerp);

camY = lerp(camY, targetY, camLerp);


level.drawBackground();


push();

translate(-camX, -camY);

level.drawWorld();

player.draw();

pop();


level.drawHUD(player, camX, camY);
```

```
}
```

```
function keyPressed() {  
  if (key === "r" || key === "R") {  
    const start = worldData.playerStart ?? { x: 300, y: 300, speed: 3 };  
    player = new Player(start.x, start.y, start.speed);  
  }  
}
```

Chatgpt:

Perfect — your structure is **very clean and scalable** already. We'll integrate this properly into your architecture without breaking your camera system.

Right now:

- Player is center-based (`rect(this.x - 12, this.y - 12, 24, 24)`)
- Obstacles are top-left based (`rect(o.x, o.y, o.w, o.h)`)

So collision needs to account for that difference.

Step 1 — Add Collision + Reaction to WorldLevel

Add these properties in your `WorldLevel` constructor:

```
this.hitCooldown = false;
```

```
// Some background presets to cycle through
```

```
this.bgVariants = [
```

```
[235, 235, 235],  
[200, 230, 255],  
[255, 220, 220],  
[220, 255, 220],  
[240, 220, 255]  
];
```

Step 2 — Add Collision Function (AABB vs Player)

Add this method to `WorldLevel`:

```
checkPlayerObstacleCollision(player) {  
  const playerSize = 24;  
  
  const px = player.x - playerSize / 2;  
  const py = player.y - playerSize / 2;  
  
  for (const o of this.obstacles) {  
    const hit =  
      px < o.x + o.w &&  
      px + playerSize > o.x &&  
      py < o.y + o.h &&  
      py + playerSize > o.y;
```



```
    if (hit) return o;
  }

  return null;
}
```

Step 3 — Add What Happens When Hit

Still inside `WorldLevel`:

```
handleObstacleHit(obstacle) {
  // Change background to random preset
  const index = floor(random(this.bgVariants.length));
  this.bg = this.bgVariants[index];

  // Move obstacle to new random position
  obstacle.x = random(100, this.w - 100);
  obstacle.y = random(100, this.h - 100);
}
```

Step 4 — Trigger It in `draw()`

In your `draw()` function (inside Sketch), add this AFTER constraining player but BEFORE drawing:

```
const hitObstacle = level.checkPlayerObstacleCollision(player);
```

```
if (hitObstacle) {  
  if (!level.hitCooldown) {  
    level.handleObstacleHit(hitObstacle);  
    level.hitCooldown = true;  
  }  
} else {  
  level.hitCooldown = false;  
}
```



What This Does

When player touches square:

- Background changes
- Square jumps somewhere else in world
- Only triggers once per contact
- Camera system stays intact



Why This Works With Your Camera

Because:

- Collision uses world coordinates
- Obstacle position updates in world space

- Drawing happens inside `translate(-camX, -camY)`

So nothing breaks visually.

Later: If You Want Smooth Color Transition

Instead of instantly switching `this.bg`, you can lerp toward a target color. But for now this confirms your logic works.

Optional Upgrade (Cleaner Architecture)

If later you want different obstacles to do different things, change JSON to:

```
"obstacles": [  
  { "x": 1200, "y": 800, "w": 40, "h": 40, "r": 10, "type": "colorShift" }  
]
```

Then switch behavior inside `handleObstacleHit()` based on `obstacle.type`.

You're Building This Correctly

Your:

- JSON → world config
- WorldLevel → logic
- Player → movement only
- Sketch → glue

That's exactly how scalable games are structured.

If you'd like next, we can:

- Add a flash effect when hit
- Add a sound trigger
- Add a smooth background transition
- Or convert the obstacle into a trigger zone instead of a solid object

What direction are you building this project toward?