

Process & Decision Documentation

This document is used to make your design and development process visible. At this stage of your academic career, you are expected not only to produce finished work, but to articulate how decisions were made, how ideas changed, and how collaboration (for the assignments that include group work) unfolds

In professional and co-op contexts, employers do not only evaluate your final projects in your portfolio. They often ask candidates to explain their process, justify trade-offs, reflect on iteration, and describe their roles within a team.

Project/Assignment Decisions

Side Quests and A4 (Individual Work)

One important decision I made was to make the player move at a slower speed when it entered the yellow section of the world. I decided to do this because I wanted the player to emulate a happy and relaxed character compared to the chaotic and energetic speed that the player moves at in the red world. This enhanced my design significantly as it showed the contrast between the two players within the game world.

Role-Based Process Evidence

```
// REMOVE json obstacles
this.obstacles = [];

// Generate rectangles across entire map
for (let i = 0; i < 50; i++) {
  this.obstacles.push({
    x: random(0, this.w),
    y: random(0, this.h),
    w: random(40, 120),
    h: random(40, 120),
    r: 10,
  });
}

You, now * Uncommitted changes
```

```

this.obstacles = [];

let attempts = 0;
let maxObstacles = 50;
let maxAttempts = 5000; // prevents infinite loop

while (this.obstacles.length < maxObstacles && attempts < maxAttempts) {
    attempts++;

    let newRect = { this: this
      x: random(0, this.w - 120),
      y: random(0, this.h - 120),
      w: random(40, 120),
      h: random(40, 120),
      r: 10,
    };

    let overlapping = false;

    for (let o of this.obstacles) {
        if (
            newRect.x < o.x + o.w &&
            newRect.x + newRect.w > o.x &&
            newRect.y < o.y + o.h &&
            newRect.y + newRect.h > o.y
        ) {
            overlapping = true;
            break;
        }
    }

    if (!overlapping) {
        this.obstacles.push(newRect);
    }
}

```

You, now * Uncommitted changes

Using ChatGPT I had it produce code to generate rectangles at random to be placed all over the world

```
{
  "schemaVersion": 1,
  "world": {
    "w": 2400,
    "h": 1600,
    "bg": [255, 190, 173],
    "gridStep": 160
  },
}
```

```

noStroke();
fill(178, 34, 34);
for (const o of this.obstacles) rect(o.x, o.y, o.w, o.h, o.r ?? 0);

this.bg = json.world?.bg ?? [255, 190, 173];

```

Using the website Cooler, I adjusted the RGB values of the game background to a light red and then the obstacles colour to a deep red [1]

```
drawWorld() {
  noStroke();

  let mid = this.h / 2;

  // Top half (original background)
  fill(this.bg[0], this.bg[1], this.bg[2]);
  rect(0, 0, this.w, mid);

  // Bottom half (light yellow)
  fill(255, 255, 204); // light yellow
  rect(0, mid, this.w, mid);

  // Grid
  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();
}
```

Code produced by ChatGPT after prompting it to help generate code to make the top half of the world red and the bottom half of the world yellow

```
this.color = [102, 20, 0]; // default top-half color

if (this.y > mid) {
  this.s = this.baseSpeed * 0.3; // slow down
  this.color = [204, 153, 0]; // dark yellow in bottom half
} else {
  this.s = this.baseSpeed; // normal speed
  this.color = [102, 20, 0]; // original color
}

const len = max(1, abs(dx) + abs(dy));
this.x += (dx / len) * this.s;
this.y += (dy / len) * this.s;

draw() {
  player.updateInput(level);
  fill(this.color);
  noStroke();
  rect(this.x - 12, this.y - 12, 24, 24, 5);
}
```

Using ChatGPT I had it produce code for the player to change from red to yellow when it crossed over into the yellow world (the yellow colour was provided by ChatGPT and can be seen in the Appendix

```
if (this.y > mid) {  
    this.s = this.baseSpeed * 0.3; // slow down  
  
if (this.y > mid) {  
    this.s = this.baseSpeed * 0.5;  
} else {  
    this.s = this.baseSpeed;  
}
```

This code was generated by ChatGPT to control the speed settings of the player according to its position on the world

Entry Header

Name: Mercedes Wilson

Role(s): Designer and coder

Primary responsibility for this work: Design and execute my ideas into the code of the example 4 game.

Goal of Work Session

Briefly describe what you were trying to accomplish during this phase of the assignment.

- Implement my design ideas for the side quest into the code
- Applying colours and design elements (obstacles and player speed) to enhance the design of the game
- Debugging the code with ChatGPT whenever necessary

Tools, Resources, or Inputs Used

- ChatGPT 5.2
- Tutorial 5 Slides on LEARN
- Cooler website for colour codes [1]

GenAI Documentation

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

Date Used: February 13th, 2026

Tool Disclosure: ChatGPT 5.2

Purpose of Use: Coding and debugging support

Summary of Interaction: The GenAI helped me code my ideas when I was stuck. It gave useful tips and tricks when debugging and figuring out as to why my code wasn't working as well.

Human Decision Point(s): I rejected some of the code ChatGPT was providing as it wasn't working for my code and then I would iterate through its responses asking it to do it again and ensuring it was giving me accurate code and not damaging my code further.

Integrity & Verification Note: I ensured ChatGPT was consistent with the tutorial notes that were learned in class.

Scope of GenAI Use: ChatGPT did not help with the design decisions and did not provide help with the colours chosen for the background of the game and the obstacles. ChatGPT did not make the final decisions on the speed of the player as I tweaked it after receiving the response.

Limitations or Misfires: ChatGPT produced incorrect code at times and was not useful when debugging my code. After a couple of iterations it eventually responded with correct debugging code and strategies, but it initially was not proving useful.

Summary of Process (Human + Tool)

Describe what you did, focusing on process rather than outcome.

- Iterated through different colours using the Cooler website to find the best fitting colours for my game design [1]
- Prompted ChatGPT to give code to add to my game design for the changing of colours in the world
- Iterated through prompts with ChatGPT while debugging my code when it was not appearing properly

Decision Points & Trade-offs

Describe one or two key decisions you made:

- Changing the players colour when it moved from the red background to the yellow background to ensure that the emotion and design was staying consistent throughout the game
- The change in colours for the player enhanced how the user was viewing and understanding the player in the game

Verification & Judgement

Explain how you evaluated whether your decision or change was appropriate:

- I reread the side quest instructions several times to ensure I was on the right track
- I revisited the tutorial slides on LEARN to double check certain parts of my code
- I had the live server open while coding to check my code was always working and that it was aligning with what I envisioned
- After writing a chunk of code I always played my game to ensure it was functioning

Limitations, Dead Ends, or Open Questions

Note anything that:

- ChatGPT gave incorrect code that made my game not work at times, so I had to go through it step by step and prompt it again
- I wanted to have the red obstacles not overlap onto the yellow world but I ran out of time to figure it out as I am leaving tomorrow abroad

Appendix

how do I make it so the rectangles appear all over the world map: 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(178, 34, 34); for (const o of this.obstacles) rect(o.x, o.y, o.w, o.h, o.r ?? 0); } { "schemaVersion": 1, "world": { "w": 2400, "h": 1600, "bg": [255, 190, 173], "gridStep": 160 }, "camera": { "lerp": 0.12 }, "playerStart": { "x": 300, "y": 300, "speed": 10 }, "obstacles": [{ "x": 40, "y": 40, "w": 80, "h": 80, "r": 10 }] }

Right now your rectangles **are only appearing where you define them in `this.obstacles`** — and your JSON only has one obstacle:

```
"obstacles": [
  { "x": 40, "y": 40, "w": 80, "h": 80, "r": 10 }
]
```

So your loop:

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

is only drawing that single rectangle.

✓ Option 1: Add More Rectangles in JSON

If you want rectangles scattered across the entire world map, just add more objects with different x and y values:

```
"obstacles": [
  { "x": 40, "y": 40, "w": 80, "h": 80, "r": 10 },
  { "x": 600, "y": 200, "w": 100, "h": 60 },
  { "x": 1200, "y": 800, "w": 150, "h": 40 },
  { "x": 2000, "y": 1200, "w": 90, "h": 120 }
]
```

Because your world is:

```
w: 2400
h: 1600
```

You can use any x between 0-2400
and any y between 0-1600

✓ Option 2: Auto-Generate Rectangles Across the Whole World

If you want rectangles randomly spread everywhere, generate them in your constructor:

```
this.obstacles = [];

for (let i = 0; i < 50; i++) {
  this.obstacles.push({
    x: random(0, this.w),
    y: random(0, this.h),
    w: random(40, 120),
    h: random(40, 120),
    r: 10
  });
}
```

Now 50 rectangles will appear across the entire 2400×1600 world.

✓ Option 3: Fill the Whole Map in a Grid Pattern

If you want them evenly covering the world:

```
this.obstacles = [];

for (let x = 0; x < this.w; x += 300) {
  for (let y = 0; y < this.h; y += 300) {
    this.obstacles.push({
      x: x,
      y: y,
      w: 100,
      h: 100,
      r: 10
    });
  }
}
```

This creates a tiled world effect.

⚠ Important Check

If rectangles still only appear in one area, make sure:

- Your **camera is translating properly**
- You are drawing the world relative to camera position

Example camera translation:

```
push();
translate(-camera.x, -camera.y);
world.drawWorld();
pop();
```

Otherwise you'll only see shapes near (0,0).

If you tell me whether you want:

- random world clutter
- platform layout
- game obstacles
- aesthetic background pattern

I can tailor the exact version for your assignment.

where do I put the option 2 code?

Put **Option 2** inside your **World class constructor**, right after you load the JSON values — and replace the JSON obstacles.

Right now your world is probably doing something like this:

```
class World {
  constructor(data) {
    this.w = data.world.w;
    this.h = data.world.h;
    this.bg = data.world.bg;
    this.gridStep = data.world.gridStep;

    this.obstacles = data.obstacles; // ← this is using JSON obstacles
  }
}
```

Replace That Line With This

```
class World {
  constructor(data) {
    this.w = data.world.w;
    this.h = data.world.h;
    this.bg = data.world.bg;
    this.gridStep = data.world.gridStep;

    // REMOVE this:
    // this.obstacles = data.obstacles;

    // ADD THIS instead:
    this.obstacles = [];

    for (let i = 0; i < 50; i++) {
      this.obstacles.push({
        x: random(0, this.w),
        y: random(0, this.h),
        w: random(40, 120),
        h: random(40, 120),
        r: 10
      });
    }
  }
}
```

```
    }  
}
```

Why It Goes in the Constructor

The constructor runs **once when the world is created**, so:

- The rectangles get generated once
- `drawWorld()` just draws them
- They don't regenerate every frame

If you put that loop inside `drawWorld()`, they would:

- Randomize every frame
 - Flicker like crazy
 - Break collisions
-

Quick Check

If `random()` gives you an error, make sure:

- You're using p5.js
 - The code runs after `setup()`
-

If you want, paste your full `World` class and I'll place it exactly where it belongs so nothing breaks.

this is what my `World` class looks like:

```
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 ?? [255, 190, 173]; 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; }
```

Perfect — this makes it super clear.

Right now you have:

```
this.obstacles = json.obstacles ?? [];
```

That means you're pulling obstacles directly from the JSON.

✓ Here's Exactly What To Change

💡 Replace this line:

```
this.obstacles = json.obstacles ?? [];
```

✓ With this:

```
this.obstacles = [];

// Generate 50 random rectangles across the whole world
for (let i = 0; i < 50; i++) {
    this.obstacles.push({
        x: random(0, this.w),
        y: random(0, this.h),
        w: random(40, 120),
        h: random(40, 120),
        r: 10
    });
}
```

💡 So Your Full Class Becomes:

```
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 ?? [255, 190, 173];
        this.gridStep = json.world?.gridStep ?? 160;

        // REMOVE json obstacles
        this.obstacles = [];

        // Generate rectangles across entire map
        for (let i = 0; i < 50; i++) {
            this.obstacles.push({
                x: random(0, this.w),
                y: random(0, this.h),
                w: random(40, 120),
                h: random(40, 120),
                r: 10
            });
        }
    }
}
```

```
        this.camLerp = json.camera?.lerp ?? 0.12;
    }
}
```

Important

Now your JSON `obstacles` section won't be used anymore.

If you want to:

- Use JSON obstacles sometimes
- Use random obstacles other times

Tell me — I can show you how to make it toggleable.

If rectangles still only show near the top-left, your issue is camera translation — and I can help fix that next.

can you change the code to make sure that the obstacles do not overlap on eachother?

Yes  — we just need to check each new rectangle against the ones already placed before adding it.

We'll use simple rectangle collision detection.

Replace Your Random Generator With This

Put this inside your constructor where the loop was:

```
this.obstacles = [];

let attempts = 0;
let maxObstacles = 50;
let maxAttempts = 5000; // prevents infinite loop

while (this.obstacles.length < maxObstacles && attempts < maxAttempts) {
    attempts++;

    let newRect = {
```

```

x: random(0, this.w - 120),
y: random(0, this.h - 120),
w: random(40, 120),
h: random(40, 120),
r: 10
};

let overlapping = false;

for (let o of this.obstacles) {
  if (
    newRect.x < o.x + o.w &&
    newRect.x + newRect.w > o.x &&
    newRect.y < o.y + o.h &&
    newRect.y + newRect.h > o.y
  ) {
    overlapping = true;
    break;
  }
}

if (!overlapping) {
  this.obstacles.push(newRect);
}
}

```

What This Does

For every new rectangle:

- Generates a random position
- Checks it against all existing rectangles
- Only adds it if it **does NOT overlap**
- Stops after 50 successful placements

`maxAttempts` prevents freezing if the world gets too full.

Optional: Add Spacing Between Them

If you want breathing room (so they don't even touch), add padding:

Change the collision check to this:

```
let padding = 20;
```

```
if (
    newRect.x < o.x + o.w + padding &&
    newRect.x + newRect.w + padding > o.x &&
    newRect.y < o.y + o.h + padding &&
    newRect.y + newRect.h + padding > o.y
)
```

Now they'll have space between them 

If you'd like, I can also:

- Make them avoid spawning near the player start
- Create clustered “biomes”
- Make them snap to your gridStep
- Generate platforms instead of random blocks

Just tell me the vibe you're going for  

I want to make the bottom half of the world background light yellow and then the rectangle obstacles on the bottom half yellow. The player I want to slow down the speed at which it moves for when it crosses over into the yellow section.

Love this — you're basically creating a **zone-based world with environmental effects**  

You need 3 things:

1.  Split background color
2.  Change obstacle color if in bottom half
3.  Slow player when inside yellow zone

I'll walk you through each cleanly.



1

Make Bottom Half Light Yellow

Modify your `drawWorld()` background section.

Replace your background code with:

```
drawWorld() {
    noStroke();

    let mid = this.h / 2;
```

```

// Top half (original background)
fill(this.bg[0], this.bg[1], this.bg[2]);
rect(0, 0, this.w, mid);

// Bottom half (light yellow)
fill(255, 255, 204); // light yellow
rect(0, mid, this.w, mid);

// Grid
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();

// Draw obstacles (we'll modify next)
for (const o of this.obstacles) {
  if (o.y > mid) {
    fill(255, 204, 0); // yellow for bottom half obstacles
  } else {
    fill(178, 34, 34); // red for top half
  }

  rect(o.x, o.y, o.w, o.h, o.r ?? 0);
}
}

```

Now:

- Top world = pink
 - Bottom world = light yellow
 - Obstacles change color depending on zone
-

✓ 2 Slow Player in Yellow Zone

In your **player update function**, you need to check:

```

let mid = world.h / 2;

if (this.y > mid) {
  this.currentSpeed = this.baseSpeed * 0.5; // slow to 50%
} else {
  this.currentSpeed = this.baseSpeed;
}

```

IMPORTANT

Your player must have:

```
this.baseSpeed = 10;  
this.currentSpeed = this.baseSpeed;
```

Then use `this.currentSpeed` in your movement code instead of hardcoded values.

Example:

```
this.vx += this.accel * this.currentSpeed;
```

OR

```
this.x += this.currentSpeed;
```

(depending on your movement system)

What's Happening Conceptually

You created a:

-  Visual biome change
-  Mechanical gameplay change
-  Emotional shift (top = free, bottom = heavy/slow)

This is actually really strong for your reflective world assignment.

If You Want It More Subtle

Instead of hard switching at mid:

```
let zoneStrength = map(this.y, mid, this.h, 1, 0.5, true);  
this.currentSpeed = this.baseSpeed * zoneStrength;
```

Now it gradually slows as you move downward.

If you paste your Player class, I can adjust the exact movement line so nothing breaks.

this is my code for the player section

It looks like the player code didn't come through — I don't see it attached.

Please paste your **Player class (or at least the movement/update section)** so I can modify it properly without breaking your physics.

Specifically, I need to see:

- Your constructor (where speed is defined)
- Your update() or move() function
- How you apply velocity (vx / vy)

Once I see that, I'll:

- Add a baseSpeed + slow-zone system cleanly
- Make sure it works with your existing acceleration/friction
- Avoid breaking collisions or jump physics

Paste it and we'll wire it up correctly 

```
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(102, 20, 0);
noStroke(); rect(this.x - 12, this.y - 12, 24, 24, 5); }}
```

Perfect — your movement system is very clean 
We'll modify it safely without breaking anything.

Right now your player always moves at:

this.s

We'll:

- Keep `this.s` as the **base speed**
- Add a dynamic `currentSpeed`
- Slow it only when in the yellow (bottom) half

✓ Step 1 — Update Constructor

Change your constructor to this:

```
class Player {  
    constructor(x, y, speed) {  
        this.x = x;  
        this.y = y;  
  
        this.baseSpeed = speed ?? 3; // original speed  
        this.currentSpeed = this.baseSpeed;  
    }  
}
```

✓ Step 2 — Modify updateInput()

Replace your movement section with this:

```
updateInput(world) {  
    const dx =  
        (keyIsDown(RIGHT_ARROW) || keyIsDown(68)) -  
        (keyIsDown(LEFT_ARROW) || keyIsDown(65));  
  
    const dy =  
        (keyIsDown(DOWN_ARROW) || keyIsDown(83)) -  
        (keyIsDown(UP_ARROW) || keyIsDown(87));  
  
    // 🔥 Check if player is in bottom half  
    const mid = world.h / 2;  
  
    if (this.y > mid) {  
        this.currentSpeed = this.baseSpeed * 0.5; // slow down in yellow  
    } else {  
        this.currentSpeed = this.baseSpeed;  
    }  
  
    const len = max(1, abs(dx) + abs(dy));  
    this.x += (dx / len) * this.currentSpeed;  
    this.y += (dy / len) * this.currentSpeed;  
}
```

✓ Step 3 — Pass the World In

Wherever you call updateInput, change:

```
player.updateInput();
```

to:

```
player.updateInput(world);
```

Otherwise it won't know where the halfway line is.

What Happens Now

- Top half → normal speed
 - Bottom half → 50% speed
 - Clean, no physics break
 - Works with camera
-

If You Want It More Emotional (Gradual Slow)

Replace the speed section with:

```
let zoneStrength = map(this.y, mid, world.h, 1, 0.4, true);
this.currentSpeed = this.baseSpeed * zoneStrength;
```

Now the deeper you go into yellow, the slower you move.

That creates a much more reflective/meditative feeling.

If you'd like, we can also:

- Change player color in yellow zone
- Add subtle screen tint
- Add friction effect instead of raw speed drop
- Add sound dampening effect

Tell me the vibe you're designing for 

That didn't work

Good — thank you for saying that. Let's debug properly instead of guessing.

Your player system is very simple:

```
this.x += (dx / len) * this.s;  
this.y += (dy / len) * this.s;
```

So the most likely reason it didn't work is:

- 👉 updateInput(world) is not receiving world
- OR
- 👉 world is undefined in that scope

Let's fix this in the safest way possible.

✓ Clean Working Version (No Parameters Needed)

Instead of passing `world` in, we'll reference it globally (since you're clearly using a global `world` variable already for drawing).

Step 1 — Modify Constructor

```
class Player {  
    constructor(x, y, speed) {  
        this.x = x;  
        this.y = y;  
  
        this.baseSpeed = speed ?? 3;  
        this.s = this.baseSpeed; // keep original movement variable  
    }  
}
```

Step 2 — Modify updateInput()

Replace the entire function with this:

```
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));  
  
    // 🔥 Check if in yellow zone
```

```
const mid = world.h / 2;

if (this.y > mid) {
    this.s = this.baseSpeed * 0.5; // slow down
} else {
    this.s = this.baseSpeed;
}

const len = max(1, abs(dx) + abs(dy));
this.x += (dx / len) * this.s;
this.y += (dy / len) * this.s;
}
```

Why This Will Work

- No need to pass parameters
 - Uses your existing `this.s`
 - Keeps your movement math unchanged
 - Only modifies speed before movement
-

Quick Debug Check

If it STILL doesn't work, add this temporarily:

```
console.log(this.y, mid);
```

If the values never cross `mid`, then:

- Your world height is different
- OR your camera is translating but player position is not world-based

If that happens, show me:

- How you draw the world
- How you translate the camera

Because then this becomes a coordinate space issue.

Reply with:

- “Still not working”
or
- Paste your draw() loop

We’ll nail it.

```
this is my WorldLevel.json file: 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 ?? [255, 190, 173]; this.gridStep = json.world?.gridStep ?? 160; this.obstacles = []; let attempts = 0; let maxObstacles = 50; let maxAttempts = 5000; // prevents infinite loop while (this.obstacles.length < maxObstacles && attempts < maxAttempts) { attempts++; let newRect = { x: random(0, this.w - 120), y: random(0, this.h - 120), w: random(40, 120), h: random(40, 120), r: 10, }; let overlapping = false; for (let o of this.obstacles) { if (newRect.x < o.x + o.w && newRect.x + newRect.w > o.x && newRect.y < o.y + o.h && newRect.y + newRect.h > o.y) { overlapping = true; break; } } if (!overlapping) { this.obstacles.push(newRect); } } this.camLerp = json.camera?.lerp ?? 0.12; }

drawBackground() { background(191, 239, 255); } drawWorld() { noStroke(); let mid = this.h / 2; // Top half (original background) fill(this.bg[0], this.bg[1], this.bg[2]); rect(0, 0, this.w, mid); // Bottom half (light yellow) fill(255, 255, 204); // light yellow rect(0, mid, this.w, mid); // Grid 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(); // Draw obstacles (we'll modify next) for (const o of this.obstacles) { if (o.y > mid) { fill(255, 204, 0); // yellow for bottom half obstacles } else { fill(178, 34, 34); // red for top half } rect(o.x, o.y, o.w, o.h, o.r ?? 0); } }

drawHUD(player, camX, camY) { noStroke(); fill(20); text("Example 4 — JSON world + smooth camera (lerp).", 12, 20); text("camLerp(JSON): " + this.camLerp + " Player: " + (player.x | 0) + "," + (player.y | 0) + " Cam: " + (camX | 0) + "," + (camY | 0), 12, 40, ); }

} } This is my world.json file: { "schemaVersion": 1, "world": { "w": 2400, "h": 1600, "bg": [255, 190, 173], "gridStep": 160 }, "camera": { "lerp": 0.12 }, "playerStart": { "x": 300, "y": 300, "speed": 10 }, "obstacles": [ { "x": 40, "y": 40, "w": 80, "h": 80, "r": 10 } ] }

This is my player.json file: class Player { constructor(x, y, speed) { this.x = x; this.y = y; this.baseSpeed = speed ?? 3; this.s = this.baseSpeed; // keep original movement variable }

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)); // Check if in yellow zone const mid = world.h / 2; if (this.y > mid) { this.s = this.baseSpeed * 0.5; // slow down } else { this.s = this.baseSpeed; }

const len = max(1, abs(dx) + abs(dy)); this.x += (dx / len) * this.s; this.y += (dy / len) * this.s; }

draw() { fill(102, 20, 0); noStroke(); rect(this.x - 12, this.y - 12, 24, 24, 5); }

}

} }

This is my sketch file: /* Week 5 — Example 4: Data-driven world with JSON + Smooth Camera Course: GBDA302 | Instructors: Dr. Karen Cochrane & David Han Date: Feb. 12,
```

2026 Move: WASD/Arrows Learning goals: - Extend the JSON-driven world to include camera parameters - Implement smooth camera follow using interpolation (lerp) - Separate camera behavior from player/world logic - Tune motion and feel using external data instead of hard-coded values - Maintain player visibility with soft camera clamping - Explore how small math changes affect “game feel” */ 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); textAlign(CENTER); 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(world); // 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); } } Using this information and code tell me what is the problem with my code.

Good — now we can see the real issue clearly.

The Problem

In your `draw()` function you wrote:

```
player.updateInput(world);
```

But:

1. There is **no variable called `world`**
2. Your world object is named **`level`**
3. Your `updateInput()` function does **NOT take a parameter**

So JavaScript is either:

- Throwing an error silently
- Or breaking before your speed logic runs

The Fix (One Line Change)

Change this in `draw()`:

```
player.updateInput(world);
```

 To this:

```
player.updateInput();
```

That's it.

Why This Works

Inside `Player.updateInput()` you already reference:

```
const mid = world.h / 2;
```

But your actual world object is named:

```
level
```

So this line is ALSO wrong.

Second Problem (Very Important)

Inside your Player class you wrote:

```
const mid = world.h / 2;
```

There is no `world`. Your world is called `level`.

Fix Inside Player Class

Change this line:

```
const mid = world.h / 2;
```

 To this:

```
const mid = level.h / 2;
```

Now it references the correct object.

Final Working Setup

In sketch.js

```
player.updateInput();
```

In Player.js

```
const mid = level.h / 2;
```

Why It Broke

You accidentally mixed:

- world
- level
- Passing a parameter
- Not passing a parameter

So the slow-zone condition never properly executed.

After This Fix

Your player will:

- Move normally in top half

- Move 50% slower in yellow zone
 - Camera will still work
 - No refactoring needed
-

If it STILL doesn't slow down after this fix, tell me — but I'm 99% sure this is the exact bug.

that still didn't work. my sketch is not showing up

Good — this tells us something important.

If your **entire sketch is not showing up**, that means you have a **runtime error** stopping execution.

And I can see exactly where it is.

The Real Problem

In your `draw()` function you have:

```
player.updateInput(world);
```

But:

- There is NO variable named `world`
- Your world object is named `level`
- Your Player function does NOT accept a parameter

This will immediately throw:

```
ReferenceError: world is not defined
```

When that happens in p5.js:

 The whole sketch stops rendering.

THE EXACT FIX

In your `draw()` function change this:

```
player.updateInput(world);
```

 To this:

```
player.updateInput();
```

BUT THERE'S ONE MORE ERROR

Inside your Player class you currently have:

```
const mid = level.h / 2;
```

That will ALSO crash.

Why?

Because `level` is defined in sketch.js, but **Player.js does not automatically know that variable exists** unless it's global and loaded in correct order.

If script load order is wrong, you get:

```
ReferenceError: level is not defined
```

SAFEST FIX (Bulletproof Version)

We will pass the level in properly.

Step 1 — Change Player function to accept level

```
updateInput(level) {
```

Step 2 — Use it inside:

```
const mid = level.h / 2;
```

Step 3 — Call it correctly in draw()

```
player.updateInput(level);
```

✓ FINAL CORRECT VERSION

Player.js

```
updateInput(level) {
  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 mid = level.h / 2;

  if (this.y > mid) {
    this.s = this.baseSpeed * 0.5;
  } else {
    this.s = this.baseSpeed;
  }

  const len = max(1, abs(dx) + abs(dy));
  this.x += (dx / len) * this.s;
  this.y += (dy / len) * this.s;
}
```

sketch.js

```
player.updateInput(level);
```

⌚ Why This Works

- No undefined variables
 - No reliance on global scope
 - Clean data flow
 - No runtime crash
-

If It STILL Doesn't Show

Open your browser console (Right Click → Inspect → Console) and tell me the exact red error message.

That message will pinpoint it instantly.

We're very close.

now how do i make it so the player changes to a dark yellow colour when it passes over into the bottom half of the world?

Perfect — now that the speed zone is working, changing the player's **color in the bottom half** is straightforward.

You already know where the bottom half starts (`mid = level.h / 2`). We can just check that and set a variable for color in the Player class.

Step 1 — Add a color property

In your **Player constructor**, add:

```
this.color = [102, 20, 0]; // default top-half color
```

Step 2 — Update color based on zone

Inside `updateInput(level)` (or any update function), after calculating `mid`:

```
if (this.y > mid) {  
    this.s = this.baseSpeed * 0.5; // slow down  
    this.color = [204, 153, 0]; // dark yellow in bottom half  
} else {  
    this.s = this.baseSpeed; // normal speed  
    this.color = [102, 20, 0]; // original color  
}
```

- `[204, 153, 0]` is a nice dark yellow
 - You can tweak the RGB if you want it brighter or darker
-

✓ Step 3—Use `this.color` when drawing

In your `draw()` function of the Player class, change:

```
fill(102, 20, 0);
```

to:

```
fill(this.color);
```

✓ Final Player.js updateInput & draw

```
updateInput(level) {
  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 mid = level.h / 2;

  if (this.y > mid) {
    this.s = this.baseSpeed * 0.5;
    this.color = [204, 153, 0]; // dark yellow
  } else {
    this.s = this.baseSpeed;
    this.color = [102, 20, 0]; // default top-half color
  }

  const len = max(1, abs(dx) + abs(dy));
  this.x += (dx / len) * this.s;
  this.y += (dy / len) * this.s;
}

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

✓ Now when your player crosses into the yellow zone:

- Speed slows down
 - Player changes to dark yellow
-

If you want, I can also make it **gradually transition color** as the player moves deeper into the bottom half, which looks much more polished and reflective.

Do you want me to do that?