Process & Decision Documentation

Project/Assignment Decisions

For this week's sidequest, I expanded on the Week 5 Example 4 code by adding collectible balls throughout the world space. By placing multiple balls across the world, I created an objective for players to explore the entire space to find them. This decision, along with a score counter, made the experience more game-like while still working with the sidequest's goal of using camera behaviour and JSON-driven design. I chose to make the world smaller than the original template so that the exploration would be more contained

Role-Based Process Evidence

Name: Queene Calading

Role(s): Designer

Primary responsibility for this work: Implementing collectible balls, collision-based progression, speed-based feedback and end state behaviours

Goal of Work Session:

- Add collectibles across the JSON defined world
- Implement collision detections
- Increasing player speed after each collection
- Track progress with a counter
- End screen

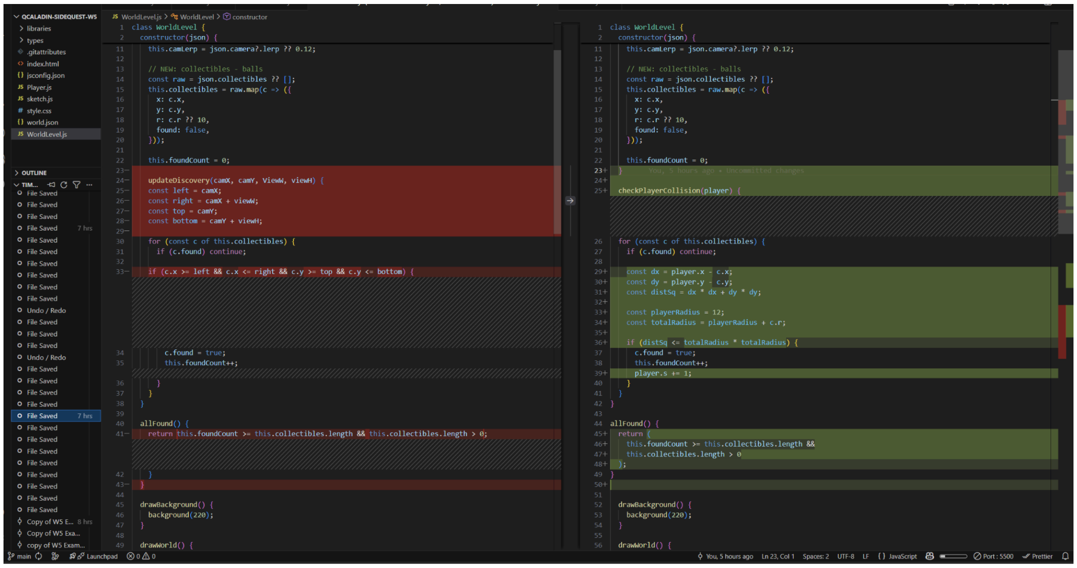
Tools, Resources, or Inputs Used:

ChatGPT 5.2 (Code help, debugging)

VSCode

Github

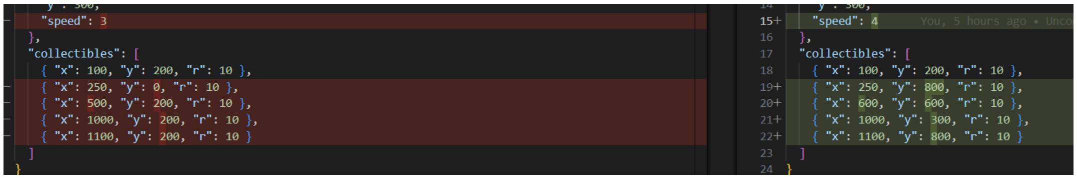
Prior starter code given (W5 Example 4)



```
File Saved
File Saved
File Saved
File Saved    7 hrs
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved
Undo / Redo
File Saved
File Saved
File Saved
Undo / Redo
File Saved
File Saved
File Saved
File Saved
File Saved    7 hrs
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved
Copy of W5 E... 8 hrs
Copy of W5 Exa...
copy of W5 Exam...
```

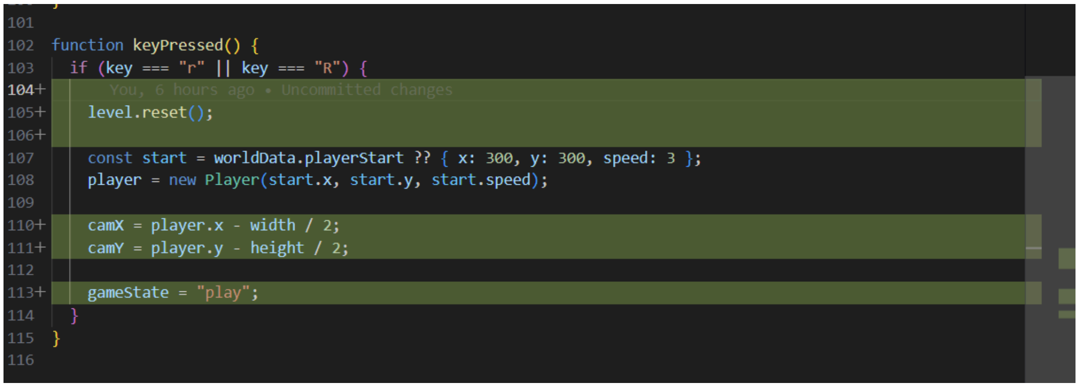
```javascript
class WorldLevel {
  constructor(json) {
    this.camLerp = json.camera?.lerp ?? 0.12;

    // NEW: collectibles - balls
    const raw = json.collectibles ?? [];
    this.collectibles = raw.map(c => ({
      x: c.x,
      y: c.y,
      r: c.r ?? 10,
      found: false,
    }));

    this.foundCount = 0;

    updateDiscovery(camX, camY, ViewW, viewH) {
    const left = camX;
    const right = camX + viewW;
    const top = camY;
    const bottom = camY + viewH;

    for (const c of this.collectibles) {
      if (c.found) continue;

    if (c.x >= left && c.x <= right && c.y >= top && c.y <= bottom) {
        c.found = true;
        this.foundCount++;
      }
    }
  }

  allFound() {
    return this.foundCount >= this.collectibles.length && this.collectibles.length > 0;
    }
  }

  drawBackground() {
    background(220);
  }

  drawWorld() {
```

```javascript
class WorldLevel {
  constructor(json) {
    this.camLerp = json.camera?.lerp ?? 0.12;

    // NEW: collectibles - balls
    const raw = json.collectibles ?? [];
    this.collectibles = raw.map(c => ({
      x: c.x,
      y: c.y,
      r: c.r ?? 10,
      found: false,
    }));

    this.foundCount = 0;
  }

  checkPlayerCollision(player) {
  for (const c of this.collectibles) {
    if (c.found) continue;

    const dx = player.x - c.x;
    const dy = player.y - c.y;
    const distSq = dx * dx + dy * dy;

    const playerRadius = 12;
    const totalRadius = playerRadius + c.r;

    if (distSq <= totalRadius * totalRadius) {
      c.found = true;
      this.foundCount++;
      player.s += 1;
    }
  }
}

allFound() {
  return (
    this.foundCount >= this.collectibles.length &&
    this.collectibles.length > 0
  );
}


  drawBackground() {
    background(220);
  }

  drawWorld() {
```

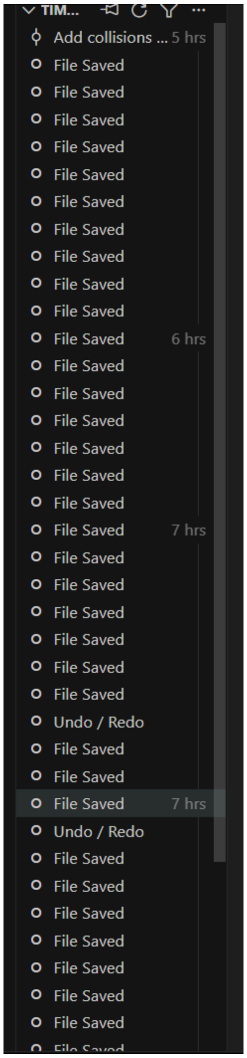


```json
    "y": 300,
    "speed": 3
  },
  "collectibles": [
    { "x": 100, "y": 200, "r": 10 },
    { "x": 250, "y": 0, "r": 10 },
    { "x": 500, "y": 200, "r": 10 },
    { "x": 1000, "y": 200, "r": 10 },
    { "x": 1100, "y": 200, "r": 10 }
  ]
}
```

```json
    "y": 300,
    "speed": 4
  },
  "collectibles": [
    { "x": 100, "y": 200, "r": 10 },
    { "x": 250, "y": 800, "r": 10 },
    { "x": 600, "y": 600, "r": 10 },
    { "x": 1000, "y": 300, "r": 10 },
    { "x": 1100, "y": 800, "r": 10 }
  ]
}
```



```javascript
function keyPressed() {
  if (key === "r" || key === "R") {

    level.reset();

    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;

    gameState = "play";
  }
}
```



```
Add collisions ... 5 hrs
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved    6 hrs
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved    7 hrs
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved
Undo / Redo
File Saved
File Saved
File Saved    7 hrs
Undo / Redo
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved
File Saved
```

GenAI Documentation

Date Used: Feb 22, 2026

Tool Disclosure: ChatGPT 5.2

Purpose of Use: Coding guidance for implementing ball collision, play and end screen, and debugging issues

Summary of Interaction:

ChatGPT helped by providing step-by-step instructions in implementing collectible balls, and how to do the collisions. It also assistant in debugging issues like missing variables, incorrect method placements, and detecting spelling errors that caused the code to not work, These instructions served as starting frameworks that I adapted into the overall code

Human Decision Point(s):

- Accepted debugging guidance and structural code suggestions, but adjusted parameters like collectible placements, world size, and player speed to better match the pace I want
- Rejected suggested “meditative pulsing glow” for collectibles, keeping it as simple static circles. I instead opted to make it speedy for fun and fast gameplay
- I rejected camera based discovery mechanic and implemented player collision instead with the balls
- I decided to reward the players with increased speed after collecting each ball

Integrity & Verification Note:

I verified that the given ideas were compatible with p5.js physics and drawing by checking the online p5.js library of what each parameter means in a line of code, and ensured no AI-generated code was copied without understanding (I asked questions and made it so that I was using AI to show me how / what I could do, rather than telling it what needs to be done and copy paste). I also adjusted values (speed increases, world size, collectible positions) based on playtesting results

Scope of GenAI Use:

GenAI contributed to debugging guidance and example implementations for collision detection, state switching, and reset behavior. However, the overall gameplay concept (collectible exploration, speed reward system, and pacing choices) was designed and refined by me. I determined the number and placement of collectibles, the progression logic, and the final user experience through testing and manual adjustments. The AI did not decide the gameplay goals or emotional tone of the experience.

Limitations or Misfires:

- Some AI code examples included unnecessary complexity for my skill level, so I simplified them to maintain readability and ensure I fully understood each mechanic
- Early AI suggestions used a camera-visibility discovery system, which did not match my intention for direct player interaction. I replaced this with collision-based collection logic
- Initial reset logic examples caused the end overlay to persist due to incorrect state resets. I debugged this by implementing a dedicated reset() method inside WorldLevel and calling it in keyPressed() to properly clear collected states

Appendix

Transcript

Prompt:

week 5 sidequest-- example 4 code

index.html:

```
<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>Week 5 — Example 4 (JSON World + Smooth Camera)</title>

<link rel="stylesheet" href="style.css" />

<script src="libraries/p5.min.js"></script>



<!-- Classes first -->

<script src="WorldLevel.js"></script>

<script src="Player.js"></script>

<!-- Then sketch -->

<script src="sketch.js"></script>

</head>

<body>

<main></main>

</body>

</html>
```

config.json:

```
{
```

```
"include": [
"*.js",
"libraries/*.js",
"c:\\Users\\david\\.vscode\\extensions\\wmcicompsci.cs30-p5-1.9.3\\p5types\\global.d.ts"
]
}
```

player.js:

```
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);
}
}
```

sketch.js:

```
/*
Week 5 — Example 4: Data-driven world with JSON + Smooth Camera
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);
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);
}
}

style.css:
html, body {
```

```
margin: 0;
padding: 0;
}
canvas {
display: block;
}
```

world.json:

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

worldLevel.js:

```
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("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,
);
}
}
```

Using this sample code, I want to create a mini game of the player moving across the world with the camera, adding circles in fixed parameters for the player to “discover” and interact with. Colliding with the circles will add to a counter like “Balls found: 1/ 5”, and adding an end screen in the end

Help show me how to do this

Response:

Below is one clean way to add **5 hidden circles (“balls”) that the camera can discover**, increment a **counter in the top-right**, and show an **end screen** when all are found — using your existing Example 4 structure (JSON-driven world + smooth camera).

---

1) Update `world.json` (add collectible circles)

Add a new array like this (positions are **world coordinates**, not screen coordinates):

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

 "collectibles": [
   { "x": 220,  "y": 260,  "r": 10 },
   { "x": 900,  "y": 420,  "r": 12 },
   { "x": 1400, "y": 1200, "r": 11 },
   { "x": 2000, "y": 800,  "r": 12 },
   { "x": 2250, "y": 1450, "r": 10 }
 ]
}
```

Change the positions however you want (hide them near corners, behind obstacles, etc.).

---

2) Update `WorldLevel.js` (store + draw + “discover” logic)

Replace your `WorldLevel` with this version (or copy-paste the added parts):

```
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 ?? [];


  // Camera tuning knob from JSON

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


  // NEW: collectibles (balls)

  const raw = json.collectibles ?? [];

  this.collectibles = raw.map(c => ({

   x: c.x,

   y: c.y,

   r: c.r ?? 10,

   found: false,

  }));


  this.foundCount = 0;

}


// Called every frame (in sketch.js) to detect "camera discovery"

updateDiscovery(camX, camY, viewW, viewH) {

  // View rectangle in world coords

  const left = camX;

  const right = camX + viewW;
```

```
    const top = camY;

    const bottom = camY + viewH;


    for (const c of this.collectibles) {

      if (c.found) continue;


      // If the ball is visible on screen, count it as "discovered"

      if (c.x >= left && c.x <= right && c.y >= top && c.y <= bottom) {

        c.found = true;

        this.foundCount++;

      }

    }

  }


  allFound() {

    return this.foundCount >= this.collectibles.length && this.collectibles.length > 0;

  }


  drawBackground() {

    background(220);

  }


  drawWorld() {

    // World fill

    noStroke();

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

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


// 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);


// Obstacles

noStroke();

fill(170, 190, 210);

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


// NEW: draw collectibles that are not found yet

// (simple “symbol”: a circle + small ring)

for (const c of this.collectibles) {

  if (c.found) continue;


  // subtle pulse so they feel “alive”

  const pulse = 1 + 0.08 * sin(frameCount * 0.06);


  noStroke();

  fill(60, 60, 60, 180);

  circle(c.x, c.y, c.r * 2 * pulse);


  noFill();

  stroke(60, 60, 60, 120);
```

```
    strokeWeight(2);
    circle(c.x, c.y, c.r * 3.2 * pulse);
    strokeWeight(1);
    noStroke();
  }
}

drawHUD(player, camX, camY) {
  // Left HUD (your original)
  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
  );
  // NEW: Top-right counter

  const total = this.collectibles.length;

  const label = `Ball Found ${this.foundCount} / ${total}`;


  const pad = 10;

  const boxW = 170;

  const boxH = 28;


  fill(255, 240);

  rect(width - boxW - pad, pad, boxW, boxH, 8);


  fill(20);

  textAlign(RIGHT, CENTER);

  text(label, width - pad - 10, pad + boxH / 2);

  textAlign(LEFT, BASELINE);

}


drawEndScreen() {

  // Overlay end screen

  push();

  fill(0, 180);

  rect(0, 0, width, height);


  fill(255);

  textAlign(CENTER, CENTER);
```

```
    textSize(28);
    text("You found them all!", width / 2, height / 2 - 20);

    textSize(16);
    text("Press R to restart", width / 2, height / 2 + 20);
    pop();
  }
}
```

---

3) Update `sketch.js` (add a simple “end state”)

Key idea:

- While playing: update input, camera, then call `level.updateDiscovery(...)`
- If all found: switch to end screen

Here’s the minimal change version:

```
const VIEW_W = 800;
const VIEW_H = 480;

let worldData;
let level;
let player;

let camX = 0;
let camY = 0;
let gameState = "play"; // NEW


function preload() {

 worldData = loadJSON("world.json");

}


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() {

 if (gameState === "play") {

   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
  const camLerp = level.camLerp;
  camX = lerp(camX, targetX, camLerp);
  camY = lerp(camY, targetY, camLerp);

  // NEW: check if camera "discovers" any balls this frame
  level.updateDiscovery(camX, camY, width, height);

  // If done, switch state
  if (level.allFound()) {
    gameState = "end";
  }
}
```

```
  // Draw (always)
  level.drawBackground();

  push();
  translate(-camX, -camY);
  level.drawWorld();
  player.draw();
  pop();

  level.drawHUD(player, camX, camY);

  // NEW: end overlay
  if (gameState === "end") {
    level.drawEndScreen();
  }
}

function keyPressed() {
  if (key === "r" || key === "R") {
    // Restart everything cleanly
    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;


  gameState = "play";
 }
}
```

Prompt:

How do I make the overall world smaller?

Response:

To make the overall world smaller, you only need to change the dimensions in world.json. Your camera, constraints, and grid will automatically adapt because they already read from the JSON

Since the world is smaller, update your `collectibles` so they stay inside bounds.

Example for the 1400×900 world:

```
"collectibles": [
 { "x": 150,  "y": 200, "r": 10 },
 { "x": 500,  "y": 300, "r": 12 },
 { "x": 900,  "y": 600, "r": 11 },
 { "x": 1200, "y": 200, "r": 12 },
 { "x": 1100, "y": 750, "r": 10 }
]
```

Prompt:

```
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;

// NEW: camera tuning knob from JSON (data-driven)
this.camLerp = json.camera?.lerp ?? 0.12;

// NEW: collectibles - balls
const raw = json.collectibles ?? [];
this.collectibles = raw.map(c => ({
x: c.x,
y: c.y,
r: c.r ?? 10,
found: false,
}));

this.foundCount = 0;

updateDiscovery(camX, camY, ViewW, viewH) {
const left = camX;
const right = camX + viewW;
const top = camY;
const bottom = camY + viewH;

for (const c of this.collectibles) {
if (c.found) continue;

if (c.x >= left && c.x <= right && c.y >= top && c.y <= bottom) {
c.found = true;
this.foundCount++;
}
}
}


allFound() {
return this.foundCount >= this.collectibles.length && this.collectibles.length > 0;
}
}

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);

}

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,
);
}
}
```

Help me debug, it says i'm missing a ";"

Response:

You’re getting the “missing ;” error because your `updateDiscovery` function is accidentally written **inside the constructor**, which breaks the class syntax.

In JavaScript classes, methods must be defined **outside** the constructor block.

Right now your structure is like this (❌ incorrect):

```
constructor(json) {
 ...
 this.foundCount = 0;


 updateDiscovery(camX, camY, ViewW, viewH) {  // ← inside constructor (syntax error)
```

```
  ...
 }
}
```

That’s why the parser thinks something is wrong and throws a “missing ;” error.

Step-by-step fix

Step 1 — Close the constructor properly

Move `updateDiscovery()` **outside** the constructor.

Step 2 — Also fix the typo in parameters

You wrote:

ViewW (capital V)

viewH

but later you use `viewW`. Keep them consistent.

Prompt:

3B) Add a function that “discovers” balls when camera sees them

Add this method anywhere inside the class:

```
updateDiscovery(camX, camY, viewW, viewH) {
const left = camX;
const right = camX + viewW;
const top = camY;
const bottom = camY + viewH;

for (const c of this.collectibles) {
```

```
if (c.found) continue;

// discovered when visible in the camera view
if (c.x >= left && c.x <= right && c.y >= top && c.y <= bottom) {
c.found = true;
this.foundCount++;
}
}
}

allFound() {
return this.foundCount >= this.collectibles.length && this.collectibles.length > 0;
}
```

3C) Draw circles in the world (only if not found)

Inside drawWorld() (after obstacles is fine), add:

```
for (const c of this.collectibles) {
if (c.found) continue;

const pulse = 1 + 0.08 * sin(frameCount * 0.06);

noStroke();
fill(60, 60, 60, 180);
circle(c.x, c.y, c.r * 2 * pulse);

noFill();
stroke(60, 60, 60, 120);
strokeWeight(2);
circle(c.x, c.y, c.r * 3.2 * pulse);
strokeWeight(1);
noStroke();
}
```

instead of this, can I just do collision check so if my square hits the discovered ball, it'll add a +1 on the counter??
Like this:

```
checkPlayerCollision(player) {
for (const c of this.collectibles) {
if (c.found) continue;

// distance between player center and ball center
const dx = player.x - c.x;
const dy = player.y - c.y;
const distSq = dx * dx + dy * dy;

// player is a 24x24 square → approx radius = 12
const playerRadius = 12;
const totalRadius = playerRadius + c.r;

if (distSq <= totalRadius * totalRadius) {
c.found = true;
this.foundCount++;
}
}
}
```

Response:

Yes — that’s actually a great idea 👍
Instead of “camera sees the ball”, you can do **player collision detection** with each circle.

This makes it feel more intentional: you *physically collect* the balls rather than passively discovering them.

In your `draw()` loop, AFTER player movement but BEFORE drawing:

Find this section:

```
camX = lerp(camX, targetX, camLerp);

camY = lerp(camY, targetY, camLerp);
```

Directly under it, add:

```
// NEW: collect balls when player touches them

level.checkPlayerCollision(player);
```

So it becomes:

```
camX = lerp(camX, targetX, camLerp);

camY = lerp(camY, targetY, camLerp);



// collect balls on collision

level.checkPlayerCollision(player);
```

Prompt:

sketch.js:

```
const VIEW_W = 800;

const VIEW_H = 480;



let worldData;

let level;

let player;
let camX = 0;
let camY = 0;

let gameState = "play";

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() {
if (gameState === "play") {
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.checkPlayerCollision(player);

if (level.allFound()) {
gameState = "end";
}
}

level.drawBackground();

push();
translate(-camX, -camY);
level.drawWorld();
```

```
player.draw();
pop();

level.drawHUD(player, camX, camY);


if (gameState === "end") {
fill(0, 180);
rect(0, 0, width, height);
fill(255);
textAlign(CENTER, CENTER);
textSize(30);
text("You found them all! YAY!", width / 2, height / 2);
textSize(14);
text("Press R to reset", width / 2, height / 2 + 30);
textAlign(LEFT, BASELINE);
}
}

function keyPressed() {
if (key === "r" || key === "R") {

levell.reset();

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;
gameState = "play";
}
}
```

world.json:

```
{
"schemaVersion": 1,
"world": {
"w": 1400,
"h": 900,
"bg": [235, 235, 235],
"gridStep": 120
},
"camera": {
"lerp": 0.10
},
"playerStart": {
"x": 300,
"y": 300,
"speed": 3
},
"collectibles": [
{ "x": 100, "y": 200, "r": 10 },
{ "x": 250, "y": 800, "r": 10 },
{ "x": 600, "y": 600, "r": 10 },
{ "x": 1000, "y": 300, "r": 10 },
{ "x": 1100, "y": 800, "r": 10 }
]
}
```

worldLevels.js:

```
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;

// NEW: camera tuning knob from JSON (data-driven)
this.camLerp = json.camera?.lerp ?? 0.12;

// NEW: collectibles - balls
const raw = json.collectibles ?? [];
this.collectibles = raw.map(c => ({
x: c.x,
y: c.y,
r: c.r ?? 10,
found: false,
}));

this.foundCount = 0;
}

checkPlayerCollision(player) {
```

```
for (const c of this.collectibles) {
if (c.found) continue;

const dx = player.x - c.x;
const dy = player.y - c.y;
const distSq = dx * dx + dy * dy;

const playerRadius = 12;
const totalRadius = playerRadius + c.r;

if (distSq <= totalRadius * totalRadius) {
c.found = true;
this.foundCount++;
}
}
}

allFound() {
return (
this.foundCount >= this.collectibles.length &&
this.collectibles.length > 0
);
}


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);

for (const c of this.collectibles) {
if (c.found) continue;

noStroke();
fill(60, 60, 60, 180);
circle(c.x, c.y, c.r * 2);
}
}

drawHUD(player, camX, camY) {
noStroke();
fill(20);
text("Sidequest W5 - Find and Collect", 12, 20);
text(
"camLerp(JSON): " +
this.camLerp +
"  Player: " +
(player.x | 0) +
"," +
(player.y | 0) +
"  Cam: " +
(camX | 0) +
"," +
(camY | 0),
12,
40,
);
}

reset() {
this.foundCount = 0;
for (const c of this.collectibles) c.found = false;
}
}
```

debug, not resetting properly after pressing r / R in end screen

Response:

Your reset isn't working because of a **typo in `keyPressed()`**:

levell.reset();

You have **two L's** (`levell`) — so the reset function never runs (it should actually throw "levell is not defined" in the console). Since the balls stay found, `level.allFound()` stays true and you immediately stay on the end overlay.

Also: your filename is `worldLevels.js` but in your `index.html` earlier it was `WorldLevel.js`. If the filename doesn't match exactly (case-sensitive on some systems), the class might not update when you think it does. But the main issue is definitely `levell`.

Minimal fix (keep everything else the same)

✅ In `sketch.js`, replace `levell.reset()` with `level.reset()`

```
function keyPressed() {
 if (key === "r" || key === "R") {
  level.reset(); // ✅ fix typo

  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;

  gameState = "play";
 }
}
```

That alone should make it reset properly.

Optional (but recommended) tiny debug line

If you want to confirm reset is happening, add this temporarily:

```
console.log("reset:", level.foundCount);
```

Right after `level.reset()`.