

Process & Decision Documentation

Name: Oma Tahir – o3tahir

Role: Coder

Primary responsibility for this work: Making changes to existing example code

Goal of work session: Taking the example code from 04, and adding an obstacle into one of the levels

Project/Assignment Decisions

For side quest 4, I wanted to add a moving obstacle into the third level. I wanted to keep the initial levels the same, but once the player reaches the third level, they face two moving obstacles. These obstacles move up and down in two different spots, that the player is trying to avoid while getting to the end point. If they fail to do, then they must re-start the level.

GenAI Documentation

GenAi was used

Date Used: Feb 8, 2026

Tool Disclosure: Copilot (built into VS code) ChatGPT 5.2

Purpose of Use: I used ChatGPT 5.2 to help me create a clear and concise prompt to give to the built in AI copilot in VS code to get it to do what I wanted. I used the copilot Ai to make the actual changes to the example code 04 I was provided to use.

Summary of interaction: I used ChatGPT 5.2 to help me phrase a clear and specific prompt for the VS code copilot. After providing that, Copilot then made the actual changes to the example code. I tested it to it completed the task needed to and then clicked keep for the changes.

Human Decision Points: I was the one who came up with the actual idea of what I wanted to add to fulfill the requirements for this side quest. I knew I wanted to add a moving obstacle to one of the levels to make the round more difficult for the player. I wanted the obstacle to move up and down making it hard to move without being touched.

Integrity & Verification Note: I reviewed the changes from copilot within my code to ensure the changes that were made aligned with what was asked in the side quest. I mainly just relied on running the game and seeing whether the moving and game start over what did they were intended to.

Limitations or Misfires: With the initial prompt I gave to copilot it was not specific enough to the code it would actually change, this caused the game to just stop working.

Summary of Process (human + tool): I reviewed the starter code that was provided to me to see how the initial example to set up and where what code could be found. I used ChatGPT to help me refine the prompt I was putting into copilot, which was initially not specific enough and making the game crash. ChatGPT 5.2 started by just giving me html code, which I did not find useful. So, I then refined what I was asking and said to give me a specific prompt to put into the built in copilot that would help me make the necessary changes. Once I inputted the prompt into copilot I tested to see if the changes did what I needed them to do. Once I was pleased with the results, I chose to keep the changes to the code and save it.

Scope of GenAI: I was the one who came up with the idea of what I wanted the obstacle to be and how I wanted it to move. Once having that sorted I then went in and asked ChatGPT to help me refine and come up with a better clearer and concise prompt to give to copilot that would help avoid the game crashing again. I was responsible for seeing the output of the changes and seeing what changes I wanted to keep in the code or not.

Verification & Judgement: I confirmed that the obstacle was behaving the way I wanted it to as well as ensuring that the level re-starts if the obstacle collides with the player. Going back to the goal for the side quest for this week to ensure what my code did aligned with the goal for this assignment.

Role-Based process Evidence

```
@@ -17,28 +17,108 @@ Responsibilities:
17 17 */
18 18
19 19 class Level {
20 20   /*
21 21   * MovingObstacle (inner class)
22 22   * - created when parsing 'M' symbols in the level grid
23 23   * - moves vertically between minRow and maxRow (pixel bounds computed)
24 24   * - reverses direction at bounds
25 25   */
26 26   static MovingObstacle = class {
27 27   constructor(col, row, minRow, maxRow, ts, speed = 1.0) {
28 28     this.col = col; // column index in grid
29 29     this.row = row; // initial row index in grid (used to reset)
30 30     this.minRow = minRow; // inclusive min row index
31 31     this.maxRow = maxRow; // inclusive max row index
32 32     this.ts = ts;
33 33     this.speed = speed;
34 34
35 35     // pixel center y position
36 36     this.y = row * ts + ts / 2;
37 37     this.x = col * ts + ts / 2;
38 38     this.dir = 1; // 1 = down, -1 = up
39 39     this.w = ts * 0.8;
40 40     this.h = ts * 0.8;
41 41     this.minY = minRow * ts + ts / 2;
42 42     this.maxY = maxRow * ts + ts / 2;
43 43   }
44 44
45 45   update() {
46 46     // Move and reverse at bounds
47 47     this.y += this.speed * this.dir;
48 48     if (this.y >= this.maxY) {
49 49       this.y = this.maxY;
50 50       this.dir = -1;
51 51     } else if (this.y <= this.minY) {
52 52       this.y = this.minY;
53 53       this.dir = 1;
54 54     }
55 55   }
56 56
57 57   draw() {
```

```
@@ -1,22 +1,40 @@
1 1 {
2 2   "levels": [
3 3     [
4 4       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
5 5       [1, 2, 0, 0, 0, 1, 0, 0, 0, 1, 1],
6 6       [1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1],
7 7       [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1],
8 8       [1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1],
9 9       [1, 0, 0, 0, 0, 0, 0, 0, 3, 1, 1],
10 10      [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
11 11     ],
12 12     [
13 13       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
14 14       [1, 2, 0, 0, 1, 0, 0, 0, 3, 1, 1],
15 15       [1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1],
16 16       [1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1],
17 17       [1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1],
18 18       [1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1],
19 19       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
20 20     ]
21 21   ],
22 22   "grid": [
23 23     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
24 24     [1, 2, 0, 0, 1, 0, 0, 0, 3, 1, 1],
25 25     [1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1],
26 26     [1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1],
27 27     [1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1],
28 28     [1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1],
29 29     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
30 30   ],
31 31   "grid": [
32 32     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
33 33     [1, 2, 0, 0, 1, 0, 0, 0, 3, 1, 1],
34 34     [1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1],
35 35     [1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1],
36 36     [1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1],
37 37     [1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1],
38 38     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
39 39   ]
40 40 }
```

Current Repository
o3tahir_sidequest4_W4

Current Branch
main

Fetch origin
Last fetched 19 minutes ago

Changes 4

History

Filter

4 changed files

level.js

levels.json

obstacle.js

sketch.js

Summary (required)

Description

A+ | New

Commit 4 files to main

sketch.js

@@ -1,4 +1,3 @@

1 -

2 const TS = 32;

3

4 // Raw JSON data (from levels.json).

5 @@ -20,18 +19,29 @@ function preload() {

6

7 function setup() {

8 //

9 - Convert raw JSON grids into Level objects.

10 - levelsData.levels is an array of 2D arrays.

11

12 + Convert raw JSON into Level objects.

13 + levelsData.levels is an array where each entry can be either:

14 + - a plain 2D array (legacy format) OR

15 + - an object with a 'grid' property and optional metadata.

16 + The loop below reads the level data and constructs Level objects.

17

18 +} //

19

20 - levels = levelsData.levels.map((grid) => new Level(copyGrid(grid), TS));

21

22 + levels = levelsData.levels.map((levelEntry) => {

23 + // (Array.isArray(levelEntry)) {

24 + // legacy: levelEntry is the grid itself

25 + return new Level(copyGrid(levelEntry), TS);

26 + } else if (levelEntry && levelEntry.grid) {

27 + return new Level(copyGrid(levelEntry.grid), TS);

28 + } else {

29 + // fallback guard

30 + return new Level({

31 + //

32 + // [1, 1,

33 + // [1, 1,

34 + //],

35 + // TS,

36 + //];

37 + }

38 + });

39

40

41

42

43

44

45

46

47 // Create a player.

48 player = new Player(TS);

49 @@ -39,18 +57,32 @@ function setup() {

50

51 function draw() {

52 background(240);

53

54

55

56

57

58

59

60

Current Repository
o3tahir_sidequest4_W4

Current Branch
main

Fetch origin
Last fetched 19 minutes ago

Changes 4

History

Filter

4 changed files

level.js

levels.json

obstacle.js

sketch.js

Summary (required)

Description

A+ | New

Commit 4 files to main

sketch.js

@@ -59,10 +59,10 @@

60 + // Update moving obstacles for the current level (they move continuously)

61 + levels[l].updateObstacles();

62 +

63 // Draw current level then player on top.

64 levels[l].draw();

65 player.draw();

66

67 + // Collision logic (rectangle-rectangle): if player rect intersects any obstacle, restart level

68 + // Compute player's AABB (we use a box around the avatar circle)

69 + const px = player.pixelX();

70 + const py = player.pixelY();

71 + const pw = player.ts * 0.6; // same diameter used when drawing the player

72 + const ph = pw;

73 + const playerRect = {

74 + left: px - pw / 2,

75 + top: py - ph / 2,

76 + right: px + pw / 2,

77 + bottom: py + ph / 2,

78 + };

79 +

80 + if (levels[l].checkObstacleCollision(playerRect)) {

81 + // restart current level: reset player pos and obstacle positions

82 + loadLevel(l);

83 + return; // skip HUD this frame so restart is immediate

84 + }

85 +

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134 + // Reset obstacles to their initial positions whenever a level is loaded.

135 + if (typeof level.resetObstacles === "function") level.resetObstacles();

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

Appendix :

Copilot transcript

My prompt:

i was instructed to Generate a level using arrays or JSON data; use loops to dynamically place tiles, obstacles, or words. I want to add in a level with a moving element that goes up and down and you need to ensure you are not touching it when trying to get past or you have to restart the level. if you pass successfully the screen will shift to the different screen/obstacle

Prompt Outcome:

I'll help you create a level with a moving obstacle and dynamic level generation. Let me first examine the current code structure.

Now I'll create a comprehensive solution with a moving obstacle system. I'll:

1. Create a `MovingObstacle` class for vertical moving elements
2. Add a new level with moving obstacles to your JSON
3. Update the `Level` class to manage obstacles
4. Update `sketch.js` to handle collision detection and level restart

Now I'll update the `Level` class to support moving obstacles:

Now I'll update `sketch.js` to handle moving obstacles and collision detection:

Now I'll update `levels.json` to add a new level with moving obstacles and use a more structured format:

Let me also update `sketch.js` to handle the new JSON structure properly:

Now I need to update `index.html` to include the new obstacle script:

Perfect! I've successfully implemented a complete solution for your level with moving obstacles. Here's what I added:

What's New:

1. MovingObstacle Class (`obstacle.js`)

- Creates vertically moving obstacles that bounce between two grid rows
- Renders as a red circle with a warning glow
- Detects collision with the player using distance calculation
- Configurable speed for different difficulty levels

2. Updated Level Class (`level.js`)

- Now accepts obstacle data in the constructor

- `updateObstacles()` - updates all obstacles each frame
- `checkObstacleCollision()` - detects if player touched an obstacle
- `draw()` - renders obstacles alongside maze tiles

3. Enhanced Sketch (`sketch.js`)

- Loads obstacle data from `levels.json`
- Updates obstacles every frame in `draw()`
- Detects collisions and automatically restarts the level with visual feedback
- Red flash overlay and warning message when collision occurs
- Smooth 300ms transition before restarting

4. New Level with Obstacles (`levels.json`)

- Restructured to support both grid and obstacle data
- Added Level 3 with a moving obstacle in column 4 that travels vertically
- Use loops/arrays to dynamically place obstacles; just add more entries to the `obstacles` array

5. Updated HTML (`index.html`)

- Added `obstacle.js` script reference
- Fixed file casing to match your filenames

How It Works:

- Navigate using **WASD or Arrow Keys**
- The red circle moves up and down in the corridor
- **Touch it = instant level restart** (visual feedback with red flash)
- **Reach the goal (orange tile)** to advance to the next level
- Level 3 demonstrates the moving obstacle mechanic

You can add more obstacles to any level by adding entries to the `obstacles` array in `levels.json` with `col`, `startRow`, `endRow`, and optional `speed` properties!

My prompt #2:

The game just crashes after the third level, i need you to include an obstacle so the user moving has to avoid touching

Prompt outcome #2:

Let me check the current state of the files to understand what's happening:

I'll add a complete obstacle system. Let me create the necessary files and updates:

Now let me update the `levels.json` to add a third level with obstacles:

Let me recreate `levels.json` properly:

My prompt #3:

I am working on a simple JavaScript game in VS Code. Constraints (must follow exactly): Use arrays or JSON data to define levels Use loops to dynamically generate the level from that data Do not hardcode obstacles or hazards in the draw function What I need implemented: Add a new level definition stored in an array or JSON object Include a moving obstacle that moves up and down continuously The moving obstacle must be generated from the level data (e.g., a symbol like "M") If the player touches the moving obstacle, the level restarts If the player reaches the exit, the game loads the next level Technical requirements: Use a loop to read the level data and create walls, player start, exit, and moving obstacle The moving obstacle should reverse direction when reaching min/max Y bounds Collision detection should be handled using rectangles Add comments explaining how the level data is parsed and how the obstacle movement works

Output format: Modify my existing code (do not rewrite the entire project) Clearly label: level data level loading loop moving obstacle update logic collision logic Generate code that fits cleanly into a beginner-level course project.

Prompt outcome #3 (Changes saved to code from this output) :

I'll add a short plan and then patch three files: `level.js`, `sketch.js`, and `levels.json`.

Deleted `levels.json`

Done — I added a moving obstacle generated from level data and rectangle collision restart.

Summary of changes:

- `level.js`: added an inner `MovingObstacle` class; parsing of "M" cells in the level grid (level data); obstacle update/draw/reset and rectangle collision logic.
 - Level parsing (level loading loop): scans grid for 2 (start) and "M" obstacle symbols; obstacles are created from the symbol and the tile is normalized to floor.
 - Moving obstacle update logic: obstacles move vertically between computed min/max rows and reverse direction at bounds.
 - Collision logic: rectangle-rectangle intersection between player's AABB and obstacle bounds.
- `sketch.js`: load loop updated to accept both legacy grid arrays and new `{"grid": [...]}` entries; updates obstacles each frame and computes the player's AABB for rectangle collision; restarts the level via `loadLevel(ii)` on collision; `loadLevel` resets obstacles.
- `levels.json`: replaced with a structured format and added a third level that includes two "M" symbols in the grid (these generate moving obstacles).

Files changed:

- `level.js`
- `sketch.js`
- `levels.json`

Next steps you can run locally:

- Open `index.html` in the browser (or run a local server) and play:
 - Use WASD / arrow keys to move.
 - Avoid the red moving obstacles (they're created from "M" in the level data).
 - Touching an obstacle restarts the current level; reaching the orange goal advances to the next level.

Want me to:

- Add a small visual restart flash or sound on collision?
- Tune obstacle speeds or placement in `levels.json`?

ChatGPT 5.2 transcript

My prompt:

I have a vs code for a simple game, I was provided the example code from my profs, i was instructed to Generate a level using arrays or JSON data; use loops to dynamically place tiles, obstacles, or words. I want to add in a level with a moving element that goes up and down and you need to ensure you are not touching it when trying to get past or you have to restart the level. if you pass successfully the screen will shift to the different screen/obstacle:

Prompt Outcome:

Here's a clean way to do it **data-driven** (arrays/JSON), with **loops** placing tiles + a **moving hazard** that goes up/down, and **level transitions** when you reach an exit. I'll give you a full working example you can paste into VS Code and run.



What this includes

- **Levels stored as arrays** (or JSON-compatible)
 - A **loop** that reads the array and spawns:
 - walls #
 - player start s
 - exit E
 - moving hazard M (moves up/down)
 - **Collision**
 - hit a wall → blocked
 - touch hazard → restart level
 - touch exit → load next level (screen shifts to new obstacle layout)
-

Option A: Plain JavaScript + Canvas (one HTML file)

Create `index.html` and paste:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Array Levels + Moving Hazard</title>
  <style>
    body { margin: 0; background: #111; display: grid; place-items: center;
height: 100vh; }
    canvas { border: 2px solid #444; background: #1a1a1a; }
    .hint { color: #bbb; font-family: system-ui; margin-top: 10px; }
```

```

</style>
</head>
<body>
  <div>
    <canvas id="c" width="640" height="384"></canvas>
    <div class="hint">Move: WASD / Arrow keys • Touch hazard = restart •
Reach exit = next level</div>
  </div>

<script>
/** =====
 * 1) LEVEL DATA (arrays/JSON)
 * Legend:
 * # = wall
 * . = empty
 * S = start
 * E = exit
 * M = moving hazard (vertical)
 * ===== */
const LEVELS = [
  {
    name: "Level 1",
    grid: [
      "#####",
      "#S.....#",
      "#.....#...M.....#",
      "#.....#.....#",
      "#.....#####.....#",
      "#.....#.....#",
      "#.....#...E...#",
      "#####",
    ],
    // Optional per-level config:
    hazardSpeed: 1.8
  },
  {
    name: "Level 2",
    grid: [
      "#####",
      "#S.....#",
      "#....#####..M.....#",
      "#.....#####..#",
      "#..#####.....#",
      "#.....#####",
      "#....E.....",
      "#####",
    ],
    hazardSpeed: 2.4
  }
];

// =====
// 2) BASIC ENGINE SETUP
// =====
const canvas = document.getElementById("c");
const ctx = canvas.getContext("2d");

```

```

const TILE = 32;                // tile size (px)
const COLS = 20;                // based on grid width
const ROWS = 8;                 // based on grid height

const keys = new Set();
addEventListener("keydown", (e) => keys.add(e.key.toLowerCase()));
addEventListener("keyup", (e) => keys.delete(e.key.toLowerCase()));

function rectsOverlap(a, b) {
  return (
    a.x < b.x + b.w &&
    a.x + a.w > b.x &&
    a.y < b.y + b.h &&
    a.y + a.h > b.y
  );
}

// =====
// 3) GAME STATE
// =====
let levelIndex = 0;

let walls = [];
let hazards = [];
let exitRect = null;

let player = {
  x: 0, y: 0, w: 22, h: 22,
  speed: 3.0,
  spawnX: 0, spawnY: 0
};

function restartLevel() {
  player.x = player.spawnX;
  player.y = player.spawnY;

  // also reset hazards to their spawn positions
  for (const hz of hazards) {
    hz.y = hz.spawnY;
    hz.dir = 1;
  }
}

function loadLevel(i) {
  levelIndex = i;
  walls = [];
  hazards = [];
  exitRect = null;

  const level = LEVELS[levelIndex];
  const grid = level.grid;

  // Loop through grid and dynamically place tiles/objects
  for (let r = 0; r < grid.length; r++) {
    for (let c = 0; c < grid[r].length; c++) {
      const ch = grid[r][c];
      const x = c * TILE;

```

```

const y = r * TILE;

if (ch === "#") {
  walls.push({ x, y, w: TILE, h: TILE });
} else if (ch === "S") {
  player.x = x + (TILE - player.w) / 2;
  player.y = y + (TILE - player.h) / 2;
  player.spawnX = player.x;
  player.spawnY = player.y;
} else if (ch === "E") {
  exitRect = { x: x + 4, y: y + 4, w: TILE - 8, h: TILE - 8 };
} else if (ch === "M") {
  // moving hazard is a rectangle that bounces between vertical bounds
  hazards.push({
    x: x + 6,
    y: y + 6,
    w: TILE - 12,
    h: TILE - 12,
    dir: 1,
    speed: level.hazardSpeed ?? 2.0,
    // movement bounds (in pixels): keep it in its column, between two
tiles
    minY: Math.max(0, y - TILE * 1), // 1 tile above marker
    maxY: Math.min(canvas.height - TILE, y + TILE * 2), // 2 tiles
below marker
    spawnY: y + 6
  });
}
}
}
}

loadLevel(0);

// =====
// 4) COLLISION HELPERS
// =====
function moveWithWallCollision(dx, dy) {
  // Move X then resolve
  player.x += dx;
  for (const w of walls) {
    if (rectsOverlap(player, w)) {
      if (dx > 0) player.x = w.x - player.w;
      if (dx < 0) player.x = w.x + w.w;
    }
  }

  // Move Y then resolve
  player.y += dy;
  for (const w of walls) {
    if (rectsOverlap(player, w)) {
      if (dy > 0) player.y = w.y - player.h;
      if (dy < 0) player.y = w.y + w.h;
    }
  }
}
}

```

```

// =====
// 5) UPDATE + DRAW LOOP
// =====
function update() {
  // Player input
  let dx = 0, dy = 0;
  if (keys.has("arrowleft") || keys.has("a")) dx -= player.speed;
  if (keys.has("arrowright") || keys.has("d")) dx += player.speed;
  if (keys.has("arrowup") || keys.has("w")) dy -= player.speed;
  if (keys.has("arrowdown") || keys.has("s")) dy += player.speed;

  if (dx || dy) moveWithWallCollision(dx, dy);

  // Move hazards (up/down bounce)
  for (const hz of hazards) {
    hz.y += hz.dir * hz.speed;
    if (hz.y <= hz.minY) { hz.y = hz.minY; hz.dir = 1; }
    if (hz.y >= hz.maxY) { hz.y = hz.maxY; hz.dir = -1; }

    // If player touches hazard => restart
    if (rectsOverlap(player, hz)) {
      restartLevel();
      return;
    }
  }

  // If player reaches exit => next level
  if (exitRect && rectsOverlap(player, exitRect)) {
    const next = levelIndex + 1;
    if (next < LEVELS.length) loadLevel(next);
    else loadLevel(0); // loop back (or show "win" screen)
  }
}

function draw() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  // Title
  ctx.fillStyle = "#ddd";
  ctx.font = "14px system-ui";
  ctx.fillText(LEVELS[levelIndex].name, 10, 18);

  // Walls
  ctx.fillStyle = "#444";
  for (const w of walls) ctx.fillRect(w.x, w.y, w.w, w.h);

  // Exit
  if (exitRect) {
    ctx.fillStyle = "#2ecc71";
    ctx.fillRect(exitRect.x, exitRect.y, exitRect.w, exitRect.h);
  }

  // Hazards
  ctx.fillStyle = "#e74c3c";
  for (const hz of hazards) ctx.fillRect(hz.x, hz.y, hz.w, hz.h);

  // Player

```

```

    ctx.fillStyle = "#4aa3ff";
    ctx.fillRect(player.x, player.y, player.w, player.h);
}

function loop() {
    update();
    draw();
    requestAnimationFrame(loop);
}
loop();
</script>
</body>
</html>

```

How to run it:

- In VS Code, open the folder
- Right-click `index.html` → “Open with Live Server” (or just open in browser)

How to adapt this to your professor’s starter code quickly

If your prof already has:

- a `player`
- a `draw()` loop
- an `update()` loop
- tile rendering

Then you only need these “drop-in” parts:

1. **Level data** (`LEVELS`)
2. **Parser loop** inside `loadLevel()` that reads characters and builds arrays
3. **Hazard update** (up/down bounce)
4. **Hazard collision** → restart
5. **Exit collision** → load next level

If you paste **your current code (or even just the file that handles levels + collision)**, I’ll rewrite it *in your exact style* (same variables/functions) so you can submit it without it looking like a totally different project.

My prompt #2:

this is for vs code can you give me a prompt to tell the built in copilot in vs code, it is not understanding what I want

Prompt Outcome #2:

Got you — this is *exactly* a **prompting problem**, not a coding one. VS Code Copilot is super literal, so you have to spell out **structure + constraints** very clearly.

Below is a **copy-paste prompt** you can drop straight into Copilot Chat in VS Code. It's written to match **prof instructions**, not vague "make a game" language.



Best Copilot Prompt (Copy–Paste This)

I am working on a simple JavaScript game in VS Code.

Constraints (must follow exactly):

- Use **arrays or JSON data** to define levels
- Use **loops** to dynamically generate the level from that data
- Do **not hardcode** obstacles or hazards in the draw function

What I need implemented:

1. Add a new level definition stored in an array or JSON object
2. Include a **moving obstacle** that moves **up and down continuously**
3. The moving obstacle must be generated from the level data (e.g., a symbol like "M")
4. If the player **touches the moving obstacle**, the level **restarts**
5. If the player **reaches the exit**, the game **loads the next level**

Technical requirements:

- Use a loop to read the level data and create walls, player start, exit, and moving obstacle
- The moving obstacle should reverse direction when reaching min/max Y bounds
- Collision detection should be handled using rectangles
- Add comments explaining how the level data is parsed and how the obstacle movement works

Output format:

- Modify my existing code (do not rewrite the entire project)

- Clearly label:
 - level data
 - level loading loop
 - moving obstacle update logic
 - collision logic

Generate code that fits cleanly into a beginner-level course project.

Why this works (important)

Copilot fails when:

- you say *“add a moving obstacle”* ❌
- you don’t say *where it comes from* ❌
- you don’t say *what NOT to do* ❌

This prompt:

- **forces arrays/JSON**
 - **forces loops**
 - **forces restart logic**
 - **forces level switching**
 - prevents Copilot from hardcoding stuff (which pros hate)
-

If Copilot still messes up

Use this **follow-up correction prompt**:

You hardcoded the obstacle instead of generating it from the level array.
Refactor the code so the moving obstacle is created only when parsing the level data inside a loop.

Or:

Do not add new drawing logic outside the existing update/render loop.
Only extend the level loading and update logic.

Want me to tailor this exactly?

If you paste:

- your **actual file name** (`sketch.js`, `game.js`, etc.)
- whether it's **canvas** / **p5.js** / **plain JS**

I'll rewrite the prompt so Copilot edits the **exact sections your prof will look at**, line by line.