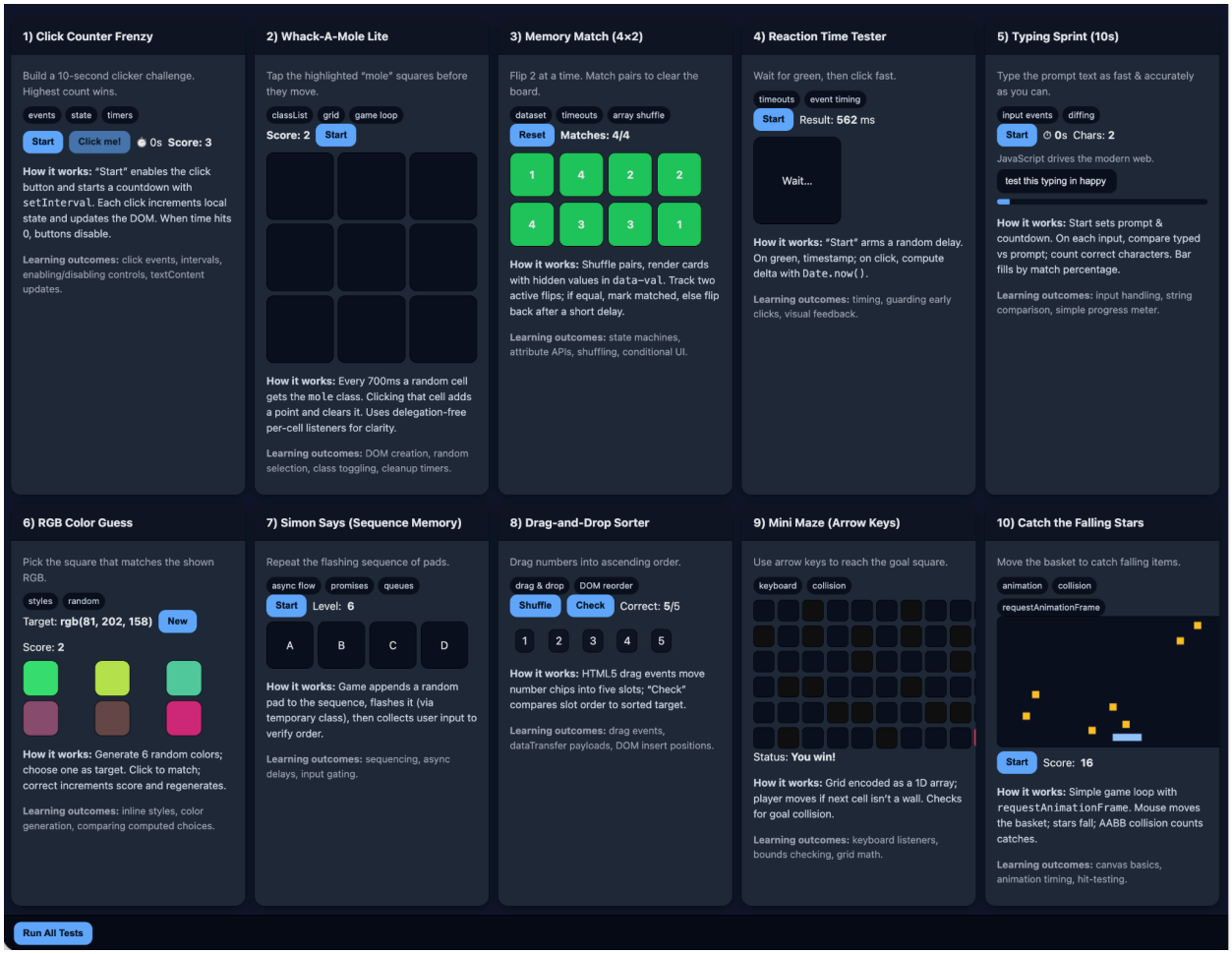
🎮 10 JavaScript DOM Coding Games — Learn by Building Fun Mini Projects



JavaScript and the Document Object Model (DOM) form the foundation of every interactive web page. The best way to truly master them is to *build something* — and that's exactly what these ten small games are designed to help you do.

Each exercise is fully self-contained, runs right in the browser, and comes with a built-in test button that validates your code behavior automatically. No frameworks, no dependencies — just **vanilla JavaScript, HTML, and CSS**.

---

1. Click Counter Frenzy

Objective: Build a 10-second click challenge.

Learning Outcomes:

- Event listeners (`click`)
- Using `setInterval` for timers
- DOM state updates and disabling buttons

How It Works:
The “Start” button enables the clicker and starts a 10-second countdown. Each click increments a score counter displayed in the DOM. When time expires, the buttons disable automatically.

---

2. Whack-A-Mole Lite

Objective: React quickly to click appearing “moles” in a 3×3 grid.

Learning Outcomes:

- Dynamic DOM creation with `document.createElement()`
- Random selection logic
- Class toggling and timing control

How It Works:
Every 700ms, one random cell gets the `.mole` class to highlight it. Clicking the mole increments the score and resets the cell.

---

3. Memory Match (4×2)

Objective: Match pairs of hidden cards.

Learning Outcomes:

- State management and matching logic
- Dataset attributes for storing values
- Conditional UI rendering

How It Works:
The script shuffles pairs of numbers, hides them behind “?” placeholders, and lets players flip two at a time. If they match, the cards stay revealed; if not, they flip back after a short delay.

---

4. Reaction Time Tester

Objective: Measure how fast you can react.

Learning Outcomes:

- Delayed events using `setTimeout`
- Handling early clicks and error states
- Timing calculations with `Date.now()`

How It Works:
When you click “Start,” a random timer delays before turning the square green. Click as soon as you see green — the code measures the milliseconds between display and click.

---

5. Typing Sprint (10 Seconds)

Objective: Type as many correct characters as possible.

Learning Outcomes:

- Input event handling
- String comparison and real-time feedback
- Visual progress updates

How It Works:
A random prompt appears, and your typing is compared live to the prompt. Correct characters are counted, and a progress bar shows your accuracy percentage.

---

6. RGB Color Guess

Objective: Match the displayed RGB color text to the correct color block.

Learning Outcomes:

- DOM styling with inline CSS
- Random color generation
- Event-driven scoring

How It Works:
The game generates six random color squares and displays one of their RGB values as text. Clicking the correct color increases the score and starts a new round.

---

7. Simon Says (Sequence Memory)

Objective: Memorize and repeat color sequences.

Learning Outcomes:

- Sequential animations with Promises
- Async game flow control
- Input order validation

How It Works:
The script adds one new random color to the sequence each round. Each pad flashes briefly. After playback, the player repeats the pattern — if correct, the sequence grows.

---

8. Drag-and-Drop Sorter

Objective: Drag numbers into ascending order.

Learning Outcomes:

- HTML5 Drag-and-Drop API

- Handling `dragstart`, `dragover`, and `drop` events
- DOM reordering and validation

How It Works:
Numbers appear shuffled. You drag them into target slots. The “Check” button compares your order to the correct ascending sequence and displays how many are correct.

---

9. Mini Maze (Arrow Keys)

Objective: Navigate the player square to the goal.

Learning Outcomes:

- Keyboard event handling (`keydown`)
- Grid-based movement and collision detection
- Updating layout dynamically

How It Works:
Each cell is part of a 10×6 grid represented by an array. Arrow key presses attempt to move the player unless blocked by a wall. Reaching the goal square shows a success message.

---

10. Catch the Falling Stars (Canvas)

Objective: Catch falling stars using a moving basket.

Learning Outcomes:

- Canvas rendering and animation loops
- Collision detection (AABB bounding box)
- `requestAnimationFrame` game loop timing

How It Works:
Stars fall from random positions. Moving your mouse controls the basket’s position. The game loop updates star positions, checks for collisions, and increments your score when you catch a

star.

Built-in Test Runner

Each HTML page includes a **Run Tests** button at the bottom.
It runs an automated JavaScript function that simulates interactions and checks expected behaviors — like whether clicking increments a score or the grid populates correctly.

This gives learners instant feedback while reinforcing debugging skills.

Why These Projects Matter

Each of these exercises reinforces *real-world DOM manipulation techniques*:

- Handling user input safely
- Updating UI dynamically
- Timing events precisely
- Managing component state
- Using animations and graphics effectively

They're perfect for classroom labs, coding bootcamps, or self-taught developers ready to push beyond tutorials.

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width,initial-scale=1" />
<title>10 JavaScript DOM Exercises (Games)</title>
<style>
 :root { --bg:#0f172a; --panel:#111827; --card:#1f2937;
```

```
--accent:#60a5fa; --ok:#10b981; --bad:#ef4444; --muted:#9ca3af; }
 body{margin:0;background:var(--bg);color:#e5e7eb;font:15px/1.5
system-ui,Segoe UI,Roboto,Helvetica,Arial,sans-serif}
 header{padding:1.2rem 1.2rem 0.6rem;}
 h1{margin:0;font-size:1.4rem}

.wrap{display:grid;grid-template-columns:repeat(auto-fill,minmax(320px,1f
r));gap:12px;padding:12px}
 .card{background:var(--card);border:1px solid
#111;border-radius:14px;overflow:hidden;box-shadow:0 4px 14px
rgba(0,0,0,.25)}
 .card header{background:var(--panel);padding:.8rem
1rem;border-bottom:1px solid #000}
 .card h2{margin:0;font-size:1rem}
 .card .body{padding:0.8rem 1rem}
 .meta{font-size:.9rem;color:var(--muted);margin:.4rem 0 .6rem}

button,.btn{background:var(--accent);border:0;color:#06121f;padding:.5re
m .8rem;border-radius:10px;cursor:pointer;font-weight:600}
 button:disabled{opacity:.6;cursor:not-allowed}
 .grid{display:grid;grid-template-columns:repeat(3,1fr);gap:6px}
 .cell{aspect-ratio:1/1;background:#0b1220;border:1px solid
#000;border-radius:10px;display:grid;place-items:center;cursor:pointer}
 .hit{background:#0ea5e9}
 .mole{background:#f59e0b}
 .row{display:flex;gap:8px;align-items:center;flex-wrap:wrap}
 input[type="text"]{padding:.5rem .7rem;border-radius:10px;border:1px
solid #111;background:#0b1220;color:#e5e7eb}
 .kbd{background:#111827;border:1px solid #000;padding:.05rem
```

```
.35rem;border-radius:6px}
 .bar{height:8px;background:#0b1220;border-radius:8px;overflow:hidden}
 .bar>i{display:block;height:100%;background:var(--accent);width:0%}
 .tags{display:flex;gap:6px;flex-wrap:wrap}
 .tag{background:#0b1220;border:1px solid
#111;color:#cbd5e1;padding:.15rem
.45rem;border-radius:999px;font-size:.75rem}
 .goal{width:28px;height:28px;background:#10b981;border-radius:6px}

.player{width:28px;height:28px;background:#f43f5e;border-radius:6px;pos
ition:relative}
 .maze{display:grid;grid-template-columns:repeat(10,28px);gap:6px}

.wall{width:28px;height:28px;background:#111;border-radius:6px;border:
1px solid #000}
 .testbar{position:sticky;bottom:0;background:#030712cc;border-top:1px
solid #000;padding:.6rem 1rem;display:flex;gap:8px;backdrop-filter:
blur(6px)}
 .pill{border-radius:999px;padding:.1rem .5rem;border:1px solid #000}
 .ok{background:#064e3b;color:#d1fae5}
 .bad{background:#7f1d1d;color:#fee2e2}
 .note{font-size:.9rem;color:#cbd5e1}
 .small{font-size:.85rem;color:#9ca3af}
 .board{display:grid;grid-template-columns:repeat(4,60px);gap:8px}

.card2{width:60px;height:60px;border-radius:10px;display:grid;place-items
:center;background:#0b1220;border:1px solid
#000;cursor:pointer;font-weight:700}
 .flipped{background:#22c55e}
```

```
 .hidden{display:none !important}
 .slot{width:40px;height:40px;border:1px dashed
#333;border-radius:8px;display:grid;place-items:center}
 .draggable{padding:.25rem .55rem;background:#111827;border:1px solid
#000;border-radius:8px;cursor:grab}
 .snake{width:240px;height:160px;background:#0b1220;border:1px solid
#000;position:relative;overflow:hidden}
 .seg{position:absolute;width:10px;height:10px;background:#22c55e}
 .food{position:absolute;width:10px;height:10px;background:#f97316}
 canvas{background:#0b1220;border:1px solid #000;border-radius:8px}
</style>
</head>
<body>
<header>
 <h1>10 JavaScript DOM Games — Exercises & Tests</h1>
 <p class="small">Each card contains the UI, controls, and a short
explanation. Use the “Run All Tests” button at the bottom to validate core
interactions automatically.</p>
</header>

<div class="wrap" id="wrap">

 <!-- 1) Click Counter Frenzy -->
 <section class="card" id="ex1">
   <header><h2>1) Click Counter Frenzy</h2></header>
   <div class="body">
     <div class="meta">Build a 10-second clicker challenge. Highest count
wins.</div>
     <div class="tags"><span class="tag">events</span><span
```

```
class="tag">state</span><span class="tag">timers</span></div>
    <div class="row" style="margin:.6rem 0">
      <button id="ex1-start">Start</button>
      <button id="ex1-btn" disabled>Click me!</button>
      <span>⏱️ <span id="ex1-t">10</span>s</span>
      <strong>Score: <span id="ex1-score">0</span></strong>
    </div>
    <p class="note"><strong>How it works:</strong> “Start” enables the click button and starts a countdown with <code>setInterval</code>. Each click increments local state and updates the DOM. When time hits 0, buttons disable.</p>
    <p class="small"><strong>Learning outcomes:</strong> click events, intervals, enabling/disabling controls, textContent updates.</p>
  </div>
 </section>

 <!-- 2) Whack-A-Mole Lite -->
 <section class="card" id="ex2">
  <header><h2>2) Whack-A-Mole Lite</h2></header>
  <div class="body">
    <div class="meta">Tap the highlighted “mole” squares before they move.</div>
    <div class="tags"><span class="tag">classList</span><span class="tag">grid</span><span class="tag">game loop</span></div>
    <div class="row"><strong>Score: <span id="ex2-score">0</span></strong><button id="ex2-start">Start</button></div>
    <div class="grid" id="ex2-grid" style="margin-top:8px"></div>
    <p class="note"><strong>How it works:</strong> Every 700ms a
```

```
random cell gets the <code>mole</code> class. Clicking that cell adds a point and clears it. Uses delegation-free per-cell listeners for clarity.</p>
    <p class="small"><strong>Learning outcomes:</strong> DOM creation, random selection, class toggling, cleanup timers.</p>
  </div>
 </section>

 <!-- 3) Memory Match 4x2 -->
 <section class="card" id="ex3">
  <header><h2>3) Memory Match (4×2)</h2></header>
  <div class="body">
    <div class="meta">Flip 2 at a time. Match pairs to clear the board.</div>
    <div class="tags"><span class="tag">dataset</span><span class="tag">timeouts</span><span class="tag">array shuffle</span></div>
    <div class="row"><button id="ex3-reset">Reset</button><strong>Matches: <span id="ex3-matches">0</span>/4</strong></div>
    <div class="board" id="ex3-board" style="margin-top:8px"></div>
    <p class="note"><strong>How it works:</strong> Shuffle pairs, render cards with hidden values in <code>data-val</code>. Track two active flips; if equal, mark matched, else flip back after a short delay.</p>
    <p class="small"><strong>Learning outcomes:</strong> state machines, attribute APIs, shuffling, conditional UI.</p>
  </div>
 </section>

 <!-- 4) Reaction Time -->
```

```
 <section class="card" id="ex4">
   <header><h2>4) Reaction Time Tester</h2></header>
   <div class="body">
     <div class="meta">Wait for green, then click fast.</div>
     <div class="tags"><span class="tag">timeouts</span><span
class="tag">event timing</span></div>
     <div class="row">
       <button id="ex4-start">Start</button>
       <span>Result: <strong id="ex4-ms">–</strong> ms</span>
     </div>
     <div id="ex4-pad" class="cell"
style="margin-top:8px;height:120px;border-radius:12px">Wait…</div>
     <p class="note"><strong>How it works:</strong> “Start” arms a
random delay. On green, timestamp; on click, compute delta with
<code>Date.now()</code>.</p>
     <p class="small"><strong>Learning outcomes:</strong> timing,
guarding early clicks, visual feedback.</p>
   </div>
 </section>

 <!-- 5) Typing Sprint (10s WPM-ish) -->
 <section class="card" id="ex5">
   <header><h2>5) Typing Sprint (10s)</h2></header>
   <div class="body">
     <div class="meta">Type the prompt text as fast & accurately as you
can.</div>
     <div class="tags"><span class="tag">input events</span><span
class="tag">diffing</span></div>
     <div class="row"><button id="ex5-start">Start</button><span>⏱ <b
```

```
id="ex5-t">10</b>s</span><span>Chars: <b
id="ex5-c">0</b></span></div>
    <div id="ex5-prompt" class="small" style="margin:.4rem 0
.3rem"></div>
    <input id="ex5-in" type="text" placeholder="Start typing…" disabled />
    <div class="bar" style="margin-top:8px"><i id="ex5-bar"></i></div>
    <p class="note"><strong>How it works:</strong> Start sets prompt &
countdown. On each input, compare typed vs prompt; count correct
characters. Bar fills by match percentage.</p>
    <p class="small"><strong>Learning outcomes:</strong> input
handling, string comparison, simple progress meter.</p>
  </div>
 </section>

 <!-- 6) RGB Color Guess -->
 <section class="card" id="ex6">
  <header><h2>6) RGB Color Guess</h2></header>
  <div class="body">
    <div class="meta">Pick the square that matches the shown RGB.</div>
    <div class="tags"><span class="tag">styles</span><span
class="tag">random</span></div>
    <div class="row">
      <span>Target: <b id="ex6-target"></b></span>
      <button id="ex6-new">New</button>
      <span>Score: <b id="ex6-score">0</b></span>
    </div>
    <div class="grid" id="ex6-grid" style="margin-top:8px"></div>
    <p class="note"><strong>How it works:</strong> Generate 6 random
colors; choose one as target. Click to match; correct increments score and
```

```
regenerates.</p>
    <p class="small"><strong>Learning outcomes:</strong> inline styles, color generation, comparing computed choices.</p>
  </div>
 </section>

 <!-- 7) Simon Says (Sequence Memory) -->
 <section class="card" id="ex7">
  <header><h2>7) Simon Says (Sequence Memory)</h2></header>
  <div class="body">
    <div class="meta">Repeat the flashing sequence of pads.</div>
    <div class="tags"><span class="tag">async flow</span><span class="tag">promises</span><span class="tag">queues</span></div>
    <div class="row"><button id="ex7-start">Start</button> Level: <b id="ex7-level">0</b></div>
    <div class="row" style="gap:6px;margin-top:6px">
      <div class="cell" id="ex7-a" style="width:64px;height:64px">A</div>
      <div class="cell" id="ex7-b" style="width:64px;height:64px">B</div>
      <div class="cell" id="ex7-c" style="width:64px;height:64px">C</div>
      <div class="cell" id="ex7-d" style="width:64px;height:64px">D</div>
    </div>
    <p class="note"><strong>How it works:</strong> Game appends a random pad to the sequence, flashes it (via temporary class), then collects user input to verify order.</p>
    <p class="small"><strong>Learning outcomes:</strong> sequencing, async delays, input gating.</p>
  </div>
 </section>
```

```
 <!-- 8) Drag-and-Drop Sorter -->
 <section class="card" id="ex8">
   <header><h2>8) Drag-and-Drop Sorter</h2></header>
   <div class="body">
     <div class="meta">Drag numbers into ascending order.</div>
     <div class="tags"><span class="tag">drag & drop</span><span
class="tag">DOM reorder</span></div>
     <div class="row"><button id="ex8-new">Shuffle</button><button
id="ex8-check">Check</button><span>Correct: <b
id="ex8-ok">0</b>/5</span></div>
     <div class="row" id="ex8-pool"
style="margin-top:6px;gap:6px"></div>
     <div class="row" id="ex8-slots"
style="margin-top:6px;gap:6px"></div>
     <p class="note"><strong>How it works:</strong> HTML5 drag events
move number chips into five slots; “Check” compares slot order to sorted
target.</p>
     <p class="small"><strong>Learning outcomes:</strong> drag events,
dataTransfer payloads, DOM insert positions.</p>
   </div>
 </section>

 <!-- 9) Mini Maze (Arrow Keys) -->
 <section class="card" id="ex9">
   <header><h2>9) Mini Maze (Arrow Keys)</h2></header>
   <div class="body">
     <div class="meta">Use arrow keys to reach the goal square.</div>
     <div class="tags"><span class="tag">keyboard</span><span
class="tag">collision</span></div>
```

```
    <div id="ex9-maze" class="maze" style="margin-top:8px"></div>
    <div class="row"><span>Status: <b
id="ex9-status">Ready</b></span></div>
    <p class="note"><strong>How it works:</strong> Grid encoded as a
1D array; player moves if next cell isn’t a wall. Checks for goal
collision.</p>
    <p class="small"><strong>Learning outcomes:</strong> keyboard
listeners, bounds checking, grid math.</p>
  </div>
 </section>

 <!-- 10) Catch the Falling Stars -->
 <section class="card" id="ex10">
  <header><h2>10) Catch the Falling Stars</h2></header>
  <div class="body">
    <div class="meta">Move the basket to catch falling items.</div>
    <div class="tags"><span class="tag">animation</span><span
class="tag">collision</span><span
class="tag">requestAnimationFrame</span></div>
    <canvas id="ex10-c" width="320" height="180"></canvas>
    <div class="row"><button id="ex10-start">Start</button> Score: <b
id="ex10-score">0</b></div>
    <p class="note"><strong>How it works:</strong> Simple game loop
with <code>requestAnimationFrame</code>. Mouse moves the basket;
stars fall; AABB collision counts catches.</p>
    <p class="small"><strong>Learning outcomes:</strong> canvas basics,
animation timing, hit-testing.</p>
  </div>
 </section>
```

```
</div>

<!-- Test bar -->
<div class="testbar">
 <button id="run-tests">Run All Tests</button>
 <span id="test-summary" class="pill ok hidden"></span>
 <span id="test-errors" class="pill bad hidden"></span>
</div>

<script>
/* ----------------------------
 Utility helpers (shared)
----------------------------*/
const $ = sel => document.querySelector(sel);
const $$ = sel => Array.from(document.querySelectorAll(sel));
const rand = (n) => Math.floor(Math.random()*n);
const shuffle = (arr) =>
arr.map(v=>[Math.random(),v]).sort((a,b)=>a[0]-b[0]).map(x=>x[1]);
const wait = (ms) => new Promise(res=>setTimeout(res, ms));
function simulateClick(el){ el.dispatchEvent(new
MouseEvent('click',{bubbles:true})); }
function simulateKey(el,key){ el.dispatchEvent(new
KeyboardEvent('keydown',{key,bubbles:true})); }

/*
==========================================================
==========
  1) Click Counter Frenzy
```

```
========================================================
========== */
(function(){
 const start = $('#ex1-start'), btn = $('#ex1-btn'), t = $('#ex1-t'), scoreEl
= $('#ex1-score');
 let time=10, score=0, timer=null, running=false;

 start.addEventListener('click', () => {
   if(running) return;
   running = true; time=10; score=0;
   t.textContent=time; scoreEl.textContent=score;
   btn.disabled=false; start.disabled=true;
   timer = setInterval(()=>{
     time--; t.textContent=time;
     if(time<=0){ clearInterval(timer); btn.disabled=true;
start.disabled=false; running=false; }
   },1000);
 });

 btn.addEventListener('click', () => {
   if(btn.disabled) return;
   score++; scoreEl.textContent=score;
 });

 // expose for tests
 window._ex1 = { start, btn, t, scoreEl, get running(){return running;} };
})();

/*
```

```
==========================================================
==========
  2) Whack-A-Mole Lite
==========================================================
========== */
(function(){
 const grid = $('#ex2-grid'), start = $('#ex2-start'), scoreEl =
$('#ex2-score');
 let cells=[], score=0, loop=null, active=-1;

 function build(){
   grid.innerHTML=''; cells=[];
   for(let i=0;i<9;i++){
     const c=document.createElement('div'); c.className='cell';
c.tabIndex=0;
     c.addEventListener('click', ()=>{
       if(i===active){ score++; scoreEl.textContent=score;
c.classList.remove('mole'); active=-1; }
     });
     grid.appendChild(c); cells.push(c);
   }
 }
 build();

 start.addEventListener('click', ()=>{
   score=0; scoreEl.textContent=score; clearInterval(loop);
   loop = setInterval(()=>{
     if(active>=0) cells[active].classList.remove('mole');
     active = rand(9);
```

```
    cells[active].classList.add('mole');
  },700);
  setTimeout(()=>{ clearInterval(loop);
cells.forEach(c=>c.classList.remove('mole')); active=-1; }, 10000);
 });

 window._ex2 = { grid, start, scoreEl, get active(){return active;} };
})();

/*
==========================================================
==========
  3) Memory Match (4×2)
==========================================================
========== */
(function(){
 const board = $('#ex3-board'), reset = $('#ex3-reset'), matchesEl =
$('#ex3-matches');
 let first=null, lock=false, matches=0;

 function setup(){
  first=null; lock=false; matches=0; matchesEl.textContent=matches;
  board.innerHTML='';
  const vals = shuffle([1,1,2,2,3,3,4,4]);
  vals.forEach(v=>{
    const d=document.createElement('div'); d.className='card2';
d.dataset.val=v;
    d.textContent='?';
    d.addEventListener('click', ()=>flip(d));
```

```
    board.appendChild(d);
  });
}

async function flip(card){
  if(lock || card.classList.contains('flipped')) return;
  card.textContent=card.dataset.val; card.classList.add('flipped');
  if(!first){ first=card; return; }
  lock=true;
  if(first.dataset.val===card.dataset.val){
    matches++; matchesEl.textContent=matches;
    first.classList.add('matched'); card.classList.add('matched');
    lock=false; first=null;
  } else {
    await wait(600);
    first.textContent='?'; first.classList.remove('flipped');
    card.textContent='?'; card.classList.remove('flipped');
    lock=false; first=null;
  }
}

reset.addEventListener('click', setup);
setup();

window._ex3 = { board, reset, matchesEl, get matches(){return matches;}
};
})();

/*
```

```
/* ================================================================
   4) Reaction Time Tester
   ================================================================ */
(function(){
 const start=$('#ex4-start'), pad=$('#ex4-pad'), msEl=$('#ex4-ms');
 let armed=false, greenAt=0, to=null;

 start.addEventListener('click', ()=>{
   armed=true; msEl.textContent='–'; pad.style.background='';
pad.textContent='Wait…';
   clearTimeout(to);
   to=setTimeout(()=>{ pad.style.background='#16a34a';
pad.textContent='Click!'; greenAt=Date.now(); }, 400 + rand(1200));
 });

 pad.addEventListener('click', ()=>{
   if(!armed) return;
   if(!greenAt){ msEl.textContent='Too soon'; armed=false; return; }
   const delta = Date.now()-greenAt;
   msEl.textContent=String(delta);
   armed=false; greenAt=0; pad.style.background='';
pad.textContent='Wait…';
 });

 window._ex4 = { start, pad, msEl };
})();
```

```
/*
================================================
==========
  5) Typing Sprint
================================================
========== */
(function(){
 const prompts = [
   "JavaScript drives the modern web.",
   "DOM events update interactive UIs.",
   "Async code keeps apps responsive."
 ];
 const start=$('#ex5-start'), tEl=$('#ex5-t'), cEl=$('#ex5-c'),
promptEl=$('#ex5-prompt'), input=$('#ex5-in'), bar=$('#ex5-bar');
 let target='', timer=null, time=10, correct=0;

 function setPrompt(){
   target = prompts[rand(prompts.length)];
   promptEl.textContent=target;
 }

 start.addEventListener('click', ()=>{
   setPrompt(); input.value=''; input.disabled=false; input.focus();
   time=10; correct=0; tEl.textContent=time; cEl.textContent=correct;
bar.style.width='0%';
   clearInterval(timer);
   timer=setInterval(()=>{ time--; tEl.textContent=time; if(time<=0){
clearInterval(timer); input.disabled=true; } },1000);
 });
```

```
 input.addEventListener('input', ()=>{
   const typed = input.value;
   let ok=0;
   for(let i=0;i<typed.length && i<target.length;i++){
     if(typed[i]===target[i]) ok++;
   }
   correct = ok; cEl.textContent=correct;
   const pct = Math.round((correct/target.length)*100);
   bar.style.width = pct + '%';
 });

 window._ex5 = { start, input, promptEl, cEl, tEl };
})();

/*
================================================
==========
  6) RGB Color Guess
================================================
========== */
(function(){
 const targetEl=$('#ex6-target'), grid=$('#ex6-grid'), btn=$('#ex6-new'),
scoreEl=$('#ex6-score');
 let colors=[], answer=0, score=0;

 function rc(){ return rand(256); }
 function rgb(r,g,b){ return `rgb(${r}, ${g}, ${b})`; }
```

```
 function newRound(){
   grid.innerHTML=''; colors=[];
   for(let i=0;i<6;i++){
     const col = rgb(rc(),rc(),rc()); colors.push(col);
   }
   answer = rand(6);
   targetEl.textContent = colors[answer];
   colors.forEach((c,i)=>{
     const d=document.createElement('div'); d.className='cell';
d.style.height='48px';
     d.style.background=c;
     d.addEventListener('click', ()=>{
       if(i===answer){ score++; scoreEl.textContent=score; newRound(); }
     });
     grid.appendChild(d);
   });
 }

 btn.addEventListener('click', newRound);
 newRound();

 window._ex6 = { targetEl, grid, btn, scoreEl, get answer(){return
answer;}, get colors(){return colors;} };
})();

/*
==========================================================
==========
  7) Simon Says
```

```
========================================================
========== */
(function(){
 const pads = [$('#ex7-a'), $('#ex7-b'), $('#ex7-c'), $('#ex7-d')];
 const start=$('#ex7-start'), levelEl=$('#ex7-level');
 let seq=[], idx=0, listening=false;

 function flash(p){ return new Promise(res=>{ p.classList.add('hit');
setTimeout(()=>{ p.classList.remove('hit'); setTimeout(res,120); }, 300);
}); }

 async function play(){
   listening=false;
   for(const i of seq){ await flash(pads[i]); }
   idx=0; listening=true;
 }

 function next(){
   seq.push(rand(4));
   levelEl.textContent=seq.length;
   play();
 }

 pads.forEach((p,i)=>{
   p.addEventListener('click', ()=>{
     if(!listening) return;
     if(i===seq[idx]){ idx++; if(idx===seq.length){ listening=false;
setTimeout(next,400);} }
     else { listening=false; seq=[]; levelEl.textContent='0'; }
```

```
  });
 });

 start.addEventListener('click', ()=>{ seq=[]; levelEl.textContent='0';
next(); });

 window._ex7 = { start, levelEl, pads, get seq(){return seq.slice();} };
})();

/*
==========================================================
==========
  8) Drag-and-Drop Sorter
==========================================================
========== */
(function(){
 const pool=$('#ex8-pool'), slots=$('#ex8-slots'), btnNew=$('#ex8-new'),
btnCheck=$('#ex8-check'), okEl=$('#ex8-ok');
 let nums=[1,2,3,4,5];

 function render(){
   pool.innerHTML=''; slots.innerHTML='';
   shuffle(nums.slice()).forEach(n=>{
     const chip=document.createElement('div'); chip.className='draggable';
chip.textContent=n; chip.draggable=true;
     chip.addEventListener('dragstart', e=>{
e.dataTransfer.setData('text/plain', n); e.dataTransfer.dropEffect='move';
chip.classList.add('dragging'); });
     chip.addEventListener('dragend', ()=>chip.classList.remove('dragging'));
```

```
    pool.appendChild(chip);
  });
  for(let i=0;i<5;i++){
    const s=document.createElement('div'); s.className='slot';
s.dataset.index=i;
    s.addEventListener('dragover', e=>e.preventDefault());
    s.addEventListener('drop', e=>{
      e.preventDefault();
      const val=e.dataTransfer.getData('text/plain');
      const existing = s.querySelector('.draggable');
      if(existing) pool.appendChild(existing);
      const chip =
Array.from(pool.querySelectorAll('.draggable')).find(c=>c.textContent===va
l) ||
Array.from(slots.querySelectorAll('.draggable')).find(c=>c.textContent===v
al);
      if(chip) s.appendChild(chip);
    });
    slots.appendChild(s);
  }
 }
 btnNew.addEventListener('click', render);
 btnCheck.addEventListener('click', ()=>{
  const got = Array.from(slots.children).map(s =>
s.textContent.trim()).filter(Boolean).map(Number);
  let correct=0; for(let i=0;i<5;i++){ if(got[i]===i+1) correct++; }
  okEl.textContent=correct;
 });
 render();
```

```
 window._ex8 = { pool, slots, btnCheck, okEl };
})();

/*
================================================
==========
 9) Mini Maze
================================================
========== */
(function(){
 const mazeEl=$('#ex9-maze'), status=$('#ex9-status');
 // 0 empty, 1 wall, 2 player, 3 goal
 const layout = [
   2,0,1,0,0,0,1,0,0,0,
   1,0,1,0,1,0,1,0,1,0,
   0,0,0,0,1,0,0,0,1,0,
   0,1,1,0,0,0,1,0,0,0,
   0,0,0,1,1,0,0,1,1,0,
   0,1,0,0,0,1,0,0,0,3
 ];
 const w=10, h=6;
 let p=0, goal=0;

 function render(){
   mazeEl.innerHTML='';
   layout.forEach((v,i)=>{
     const d=document.createElement('div');
     if(v===1){ d.className='wall'; }
```

```
    else if(v===2){ d.className='player'; p=i; }
    else if(v===3){ d.className='goal'; goal=i; }
    else { d.className='wall'; d.style.visibility='hidden';
d.classList.remove('wall'); d.style.visibility='visible'; d.className='';
d.classList.add(); d.style.width='28px'; d.style.height='28px';
d.style.border='1px solid #000'; d.style.borderRadius='6px';
d.style.background='#0b1220'; }
    mazeEl.appendChild(d);
  });
 }
 render();

 function move(dx,dy){
  const x=p%w, y=Math.floor(p/w);
  const nx=x+dx, ny=y+dy;
  if(nx<0||nx>=w||ny<0||ny>=h) return;
  const ni = ny*w+nx;
  if(layout[ni]===1) return;
  // move
  layout[p]=0; layout[ni]=2; p=ni; render();
  if(p===goal){ status.textContent='You win!'; }
 }

 window.addEventListener('keydown', (e)=>{
  if(['ArrowUp','ArrowDown','ArrowLeft','ArrowRight'].includes(e.key))
e.preventDefault();
  if(e.key==='ArrowUp') move(0,-1);
  if(e.key==='ArrowDown') move(0,1);
  if(e.key==='ArrowLeft') move(-1,0);
```

```
  if(e.key==='ArrowRight') move(1,0);
 });

 window._ex9 = { move, status, get p(){return p;}, get goal(){return goal;}
};
})();

/*
==========================================================
==========
  10) Catch the Falling Stars (Canvas)
==========================================================
========== */
(function(){
 const c = $('#ex10-c'), ctx=c.getContext('2d'), start=$('#ex10-start'),
scoreEl=$('#ex10-score');
 let basketX=160, running=false, stars=[], last=0, score=0;

 c.addEventListener('mousemove', e=>{
  const rect=c.getBoundingClientRect();
  basketX = e.clientX - rect.left;
 });

 function spawn(){
  stars.push({ x: rand(c.width-10), y:-10, v: 40+rand(60) });
 }

 function loop(ts){
  if(!running) return;
```

```
  const dt = last? (ts-last)/1000 : 0; last=ts;
  // update
  if(Math.random()<0.05) spawn();
  stars.forEach(s=> s.y += s.v*dt);
  // collide (AABB)
  const bx=basketX-20, by=c.height-18, bw=40, bh=10;
  stars = stars.filter(s=>{
    const hit = s.x < bx+bw && s.x+10 > bx && s.y < by+bh && s.y+10 >
by;
    if(hit){ score++; scoreEl.textContent=score; return false; }
    return s.y < c.height+10;
  });
  // draw
  ctx.clearRect(0,0,c.width,c.height);
  // basket
  ctx.fillStyle='#93c5fd'; ctx.fillRect(bx,by,bw,bh);
  // stars
  ctx.fillStyle='#fbbf24';
  stars.forEach(s=> ctx.fillRect(s.x,s.y,10,10));
  requestAnimationFrame(loop);
}

start.addEventListener('click', ()=>{
  running=true; score=0; scoreEl.textContent=score; stars=[]; last=0;
  requestAnimationFrame(loop);
  setTimeout(()=>{ running=false; }, 10000);
});

window._ex10 = { c, start, scoreEl, get running(){return running;} };
```

```
})();

/*
================================================
==========
  Shared Automated Tests
  - Click “Run All Tests” to validate core interactions.
================================================
========== */
(function(){
 const btn=$('#run-tests'), sum=$('#test-summary'), err=$('#test-errors');

 function ok(msg){ logs.push({ok:true,msg}); }
 function bad(msg){ logs.push({ok:false,msg}); }

 async function test1(){
   const {start,btn:clicker,scoreEl} = window._ex1;
   simulateClick(start);
   if(clicker.disabled) bad('ex1: click button should be enabled after start');
else ok('ex1: enabled after start');
   const s0 = +scoreEl.textContent; simulateClick(clicker); const
s1=+scoreEl.textContent;
   if(s1===s0+1) ok('ex1: score increments'); else bad('ex1: score did not
increment');
 }

 async function test2(){
   const {start, grid} = window._ex2;
   simulateClick(start);
```

```
  await wait(750);
  const mole = grid.querySelector('.mole');
  if(mole){ ok('ex2: mole appears'); simulateClick(mole); ok('ex2: mole can be clicked'); }
  else bad('ex2: no mole appeared');
 }

 async function test3(){
  const {board, reset} = window._ex3;
  simulateClick(reset);
  const cards = board.querySelectorAll('.card2');
  if(cards.length===8) ok('ex3: board has 8 cards'); else bad('ex3: wrong card count');
  simulateClick(cards[0]); simulateClick(cards[1]);
  ok('ex3: cards flip on click');
 }

 async function test4(){
  const {start, pad, msEl} = window._ex4;
  simulateClick(start);
  // too soon click
  simulateClick(pad);
  if(msEl.textContent==='Too soon') ok('ex4: guards early clicks');
  else bad('ex4: did not guard early clicks');
 }

 async function test5(){
  const {start, input, promptEl, cEl} = window._ex5;
  simulateClick(start);
```

```
  await wait(50);
  const t = promptEl.textContent;
  input.value = t.slice(0,3);
  input.dispatchEvent(new Event('input',{bubbles:true}));
  if(+cEl.textContent===3) ok('ex5: counts correct chars'); else bad('ex5: incorrect char count');
 }

 async function test6(){
  const {grid, targetEl} = window._ex6;
  const target = targetEl.textContent;
  const cell = Array.from(grid.children).find(d => d.style.background === target);
  if(cell){ ok('ex6: target color rendered'); }
  else bad('ex6: target color not found among choices');
 }

 async function test7(){
  const {start, pads, levelEl} = window._ex7;
  simulateClick(start);
  await wait(900);
  if(+levelEl.textContent>=1) ok('ex7: level increments'); else bad('ex7: level did not start');
  simulateClick(pads[0]); ok('ex7: allows input clicks');
 }

 async function test8(){
  const {pool, slots, btnCheck, okEl} = window._ex8;
  // Move "1" into slot[0] if available
```

```
  const one = Array.from(pool.children).find(c=>c.textContent==='1') ||
Array.from(slots.children).map(s=>s.firstElementChild).find(c=>c &&
c.textContent==='1');
  if(one){
    const s0 = slots.children[0];
    s0.dispatchEvent(new DragEvent('dragover',{bubbles:true}));
    const dt = new DataTransfer(); dt.setData('text/plain','1');
    s0.dispatchEvent(new
DragEvent('drop',{dataTransfer:dt,bubbles:true}));
    simulateClick(btnCheck);
    if(+okEl.textContent>=1) ok('ex8: drop registered & check counts'); else
bad('ex8: check did not count');
  } else bad('ex8: could not locate chip "1"');
 }

 async function test9(){
  const {move, status} = window._ex9;
  move(1,0); // attempt move right
  ok('ex9: movement callable');
  status.textContent='Ready'; // reset
 }

 async function test10(){
  const {start, scoreEl} = window._ex10;
  simulateClick(start);
  await wait(300);
  ok('ex10: game loop started');
  // We can't simulate mouse here reliably; just ensure scoreEl exists
  if(scoreEl) ok('ex10: score element present');
```

```
 }

 let logs=[];
 async function runAll(){
   logs=[];
   const tests=[test1,test2,test3,test4,test5,test6,test7,test8,test9,test10];
   for(const t of tests){ try{ await t(); } catch(e){ bad(t.name+': threw
'+e.message); } }
   const pass = logs.filter(l=>l.ok).length, fail=logs.length-pass;
   sum.textContent = `${pass}/${logs.length} passed`;
   err.textContent = `${fail} failed`;
   sum.classList.remove('hidden'); err.classList.remove('hidden');
   sum.className = 'pill ' + (fail? 'bad':'ok');
   err.className = 'pill ' + (fail? 'bad':'ok');
   if(!fail){ err.classList.add('hidden'); }
 }

 btn.addEventListener('click', runAll);
 window.runAllTests = runAll; // also expose globally
})();
</script>
</body>
</html>
```