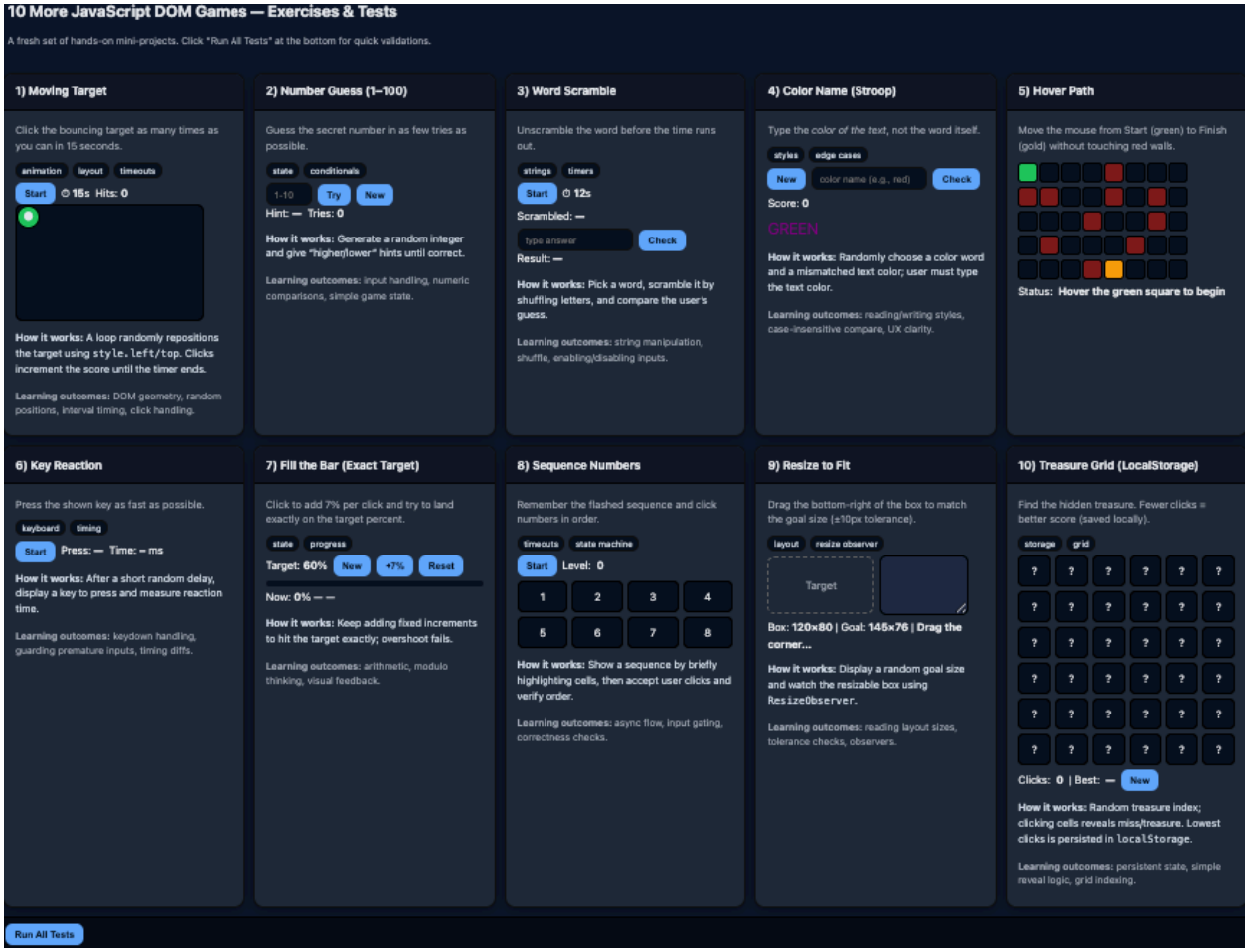

🕹️ Exercise 11 – Moving Target

Goal: click a moving dot as many times as you can in 15 seconds.

How it works:

- When "Start" is clicked, `playing` becomes true, score = 0, and two intervals start:
    - A `setInterval(move, 450)` randomly changes the target's `.style.left` and `.style.top` inside the arena (`getBoundingClientRect()` gives its bounds).
    - Another `setInterval` decrements the timer each second until t ≤ 0, then stops both loops.
- Clicking the target increments score and immediately calls `move()` again.

Concepts practiced: timed animation loops, random positioning, updating DOM text.

Learning outcome: working with layout geometry (`offsetWidth`, `getBoundingClientRect()`), intervals, and event listeners.

🎯 Exercise 12 – Number Guess (1–100)

Goal: guess the random secret number.

How it works:

- `reset()` picks a random integer 1–100 and clears counters.
- Each click of **Try**:
    - Reads `input.value`, converts with `+input.value`.
    - Increments `tries`, displays 'Higher' / 'Lower' / 'Correct!' in `#ex12-hint`.
- **New** button calls `reset()`.

Concepts practiced: numeric input handling, conditionals, state reset.

Learning outcome: comparing numbers, managing state variables between rounds.

🔤 Exercise 13 – Word Scramble

Goal: unscramble the word before 12 seconds expire.

How it works:

- **Start**:
    - Selects a random word from `words[]`.
    - Scrambles letters using `shuffle()` → random sort → join back.
    - Enables input and **Check**.
    - Starts a `setInterval` timer that decrements `t` every second.
- **Check** compares lowercase trimmed input with target.

- If equal, shows “Correct”, stops timer, disables input.

Concepts practiced: array shuffle, timers, enabling/disabling form elements.

Learning outcome: asynchronous countdown logic, simple word games with strings.

🎨 Exercise 14 – Color Name (Stroop Effect)

Goal: type the **color of the text**, not the word itself.

How it works:

- **New** button chooses random `text` word and independent `color`.
- The displayed `#ex14-word` uses `.textContent = text.toUpperCase()` and `.style.color = color`.
- The target answer is the color used for styling.
- **Check** compares user’s lower-cased input with `targetColor`.
  - If correct → increments score and starts a new round.

Concepts practiced: reading/writing inline CSS, string normalization, game loops.

Learning outcome: manipulating DOM styles dynamically, case-insensitive checks.

🧩 Exercise 15 – Hover Path (Mouse Maze)

Goal: move your mouse from green Start to gold Finish without touching red walls.

How it works:

- A layout array (`'S'`, `'F'`, `1`, `0`) builds a 5×8 grid of `.hcell` elements.
- Each cell gets a `data-type` (`start`, `finish`, `wall`, `empty`).
- `mouseover` on the grid container detects which cell the cursor entered:
  - If entering Start → set `playing=true`.

    - If `playing` and cell = `wall` → lose(); if `finish` → win().
    - Safe cells mark a trail (`.path` class).
- Leaving the grid triggers lose("You left the maze!").

Concepts practiced: event delegation, dataset attributes, interactive feedback.

Learning outcome: bubbling vs non-bubbling mouse events, state machines, and DOM updates.

⌨️ Exercise 16 – Key Reaction Test

Goal: press the shown letter key as quickly as possible.

How it works:

- **Start** sets a random delay (`setTimeout`) before showing a random letter from A–Z.
- When displayed, stores `shownAt = Date.now()` and sets `armed = true`.
- `keydown` event on `window` checks if `e.key.toUpperCase() === target`.
    - If correct, shows reaction time = `Date.now() - shownAt`, disarms.

Concepts practiced: global key events, guarding early inputs, measuring ms diffs.

Learning outcome: event timing and user-response measurement.

📊 Exercise 17 – Fill the Bar (Exact Target)

Goal: reach a randomly chosen target percent using +7 increments.

How it works:

- **New** sets target between 20 and 80 (5 increments).
- **+7 %** increases `now` by 7 until 100, updates `.style.width` on `#ex17-bar`.
- Displays messages: ‘Exact 🎯’, ‘Overshoot!’, or ‘Keep going…’.

- **Reset** clears progress.

Concepts practiced: arithmetic updates, CSS width changes.

Learning outcome: DOM binding between numeric state and visual progress.

🔢 Exercise 18 – Sequence Numbers (Simon Style)

Goal: memorize and click the flashing number sequence.

How it works:

- Creates 8 cells labeled 1–8.
- Each level adds a random index to `seq[]`.
- `play()` loops through the sequence, calling `flash(i)` which temporarily colors a cell.
- Once shown, sets `listening = true`.
- User clicks:
    - If `i === seq[idx]`, increment idx.
    - When entire sequence correct → add next level.
    - If wrong → reset level = 0.

Concepts practiced: async/await timing, sequential logic, DOM feedback.

Learning outcome: implementing a state machine with timed animations and user input validation.

📐 Exercise 19 – Resize to Fit

Goal: manually resize a box to match a target size within ±10 px tolerance.

How it works:

- Randomly chooses a goal width (120–200) and height (70–140) for `#ex19-goal`.

- Displays those dimensions beside the resizable `#ex19-box` (`resize: both` CSS).
- `ResizeObserver` fires on box resize:
    - Reads `clientWidth/Height`, rounds to nearest px.
    - If both within 10 of goal → shows “Close enough! ✅”; else “Drag the corner…”.

Concepts practiced: modern `ResizeObserver`, live layout reading.

Learning outcome: observing DOM geometry changes and tolerance checking.

💎 Exercise 20 – Treasure Grid (with LocalStorage)

Goal: find the hidden treasure in as few clicks as possible; lowest score is saved.

How it works:

- Builds 6×6 grid (36 cells) each labeled ‘?’.
- Picks random treasure index (`idx=rand(36)`).
- On cell click:
    - Increments clicks; if already clicked, ignores.
    - If `i===idx` → display 💎, store best score in `localStorage`.
    - Else mark ‘·’.
- **New** rebuilds grid and loads best score from storage.

Concepts practiced: grid generation, per-cell handlers, persistent storage.

Learning outcome: saving progress locally with `localStorage`, comparing and updating records.

⚙️ Test Bar (Functionality Checker)

At the bottom, a **“Run All Tests”** button executes quick sanity checks for each exercise:

- Each async test triggers a key action (e.g., click Start, simulate input) and pushes pass/fail messages into `logs[]`.
- Counts total passed/failed and updates sticky summary pills.

Purpose: give instant confirmation that each widget’s core logic is wired correctly.

🧠 Overall Learning Outcomes (11 – 20)

Concept	Where Practiced
Event listeners (click, keydown, mouseover)	All games
Timers & Intervals	11, 13, 16, 18
State management	12, 15, 17, 18
Randomization	11, 12, 13, 14, 18, 20
DOM geometry & layout	11, 19
CSS updates via JS	14, 17, 18
Storage APIs	20
Asynchronous flows / Promises	18
UX feedback & game loops	All

Code

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width,initial-scale=1" />
<title>10 More JavaScript DOM 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,1fr));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{background:var(--accent);border:0;color:#06121f;padding:.5rem
.8rem;border-radius:10px;cursor:pointer;font-weight:600}
 .row{display:flex;gap:8px;align-items:center;flex-wrap:wrap}
 .small{font-size:.85rem;color:#9ca3af}
 .note{font-size:.9rem;color:#cbd5e1}
 .tag{background:#0b1220;border:1px solid #111;color:#cbd5e1;padding:.15rem
.45rem;border-radius:999px;font-size:.75rem;margin-right:6px}
 .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}
 .hidden{display:none}
 .arena{position:relative;width:260px;height:160px;border:1px solid
#000;border-radius:10px;background:#0b1220;overflow:hidden}

.target{position:absolute;width:28px;height:28px;border-radius:50%;background:#22c55e;
border:1px solid #000;display:grid;place-items:center;font-weight:700}
 .grid{display:grid;grid-template-columns:repeat(5,1fr);gap:6px}
 .cell{height:40px;border-radius:8px;border:1px solid
#000;background:#0b1220;display:grid;place-items:center;cursor:pointer}
 input[type=text],input[type=number]{padding:.45rem
.6rem;border-radius:10px;border:1px solid #111;background:#0b1220;color:#e5e7eb}
 .bar{height:8px;background:#0b1220;border-radius:8px;overflow:hidden}
 .bar>i{display:block;height:100%;background:var(--accent);width:0%}
```

```
 .pad{width:64px;height:64px;border-radius:10px;border:1px solid
#000;background:#0b1220;display:grid;place-items:center;font-weight:700;cursor:pointer
}
 .drag-box{width:120px;height:80px;background:#1f2a44;border:1px solid
#000;border-radius:10px;resize:both;overflow:auto}
 .goal-box{border:1px dashed
#666;border-radius:10px;width:140px;height:90px;display:grid;place-items:center;color:
#9ca3af}
 .hover-grid{display:grid;grid-template-columns:repeat(8,24px);gap:6px}
 .hcell{width:24px;height:24px;border-radius:6px;border:1px solid
#000;background:#0b1220}
 .wall{background:#111}
 .start{background:#22c55e}
 .finish{background:#f59e0b}
   /* Add or keep these styles (tweak to taste) */
 .hover-grid{display:grid}
 .hcell{width:24px;height:24px;border-radius:6px;border:1px solid
#000;background:#0b1220}
 .hcell.wall{background:#7f1d1d}
 .hcell.start{background:#22c55e}
 .hcell.finish{background:#f59e0b}
 .hcell.path{background:#1e293b}       /* trail */
 .hp-flash{animation:hpflash .3s linear 1}
 @keyframes hpflash{from{filter:brightness(1.6)}to{filter:brightness(1)}}
</style>
</head>
<body>
<header>
 <h1>10 More JavaScript DOM Games — Exercises & Tests</h1>
 <p class="small">A fresh set of hands-on mini-projects. Click “Run All Tests” at the
bottom for quick validations.</p>
</header>

<div class="wrap">

 <!-- 1) Moving Target -->
 <section class="card" id="ex11">
   <header><h2>1) Moving Target</h2></header>
   <div class="body">
```

```
      <div class="meta">Click the bouncing target as many times as you can in 15 seconds.</div>
      <span class="tag">animation</span><span class="tag">layout</span><span class="tag">timeouts</span>
      <div class="row" style="margin-top:6px">
        <button id="ex11-start">Start</button>
        <span>⏱ <b id="ex11-time">15</b>s</span>
        <span>Hits: <b id="ex11-score">0</b></span>
      </div>
      <div class="arena" id="ex11-arena"><div class="target" id="ex11-target">●</div></div>
      <p class="note"><strong>How it works:</strong> A loop randomly repositions the target using <code>style.left/top</code>. Clicks increment the score until the timer ends.</p>
      <p class="small"><strong>Learning outcomes:</strong> DOM geometry, random positions, interval timing, click handling.</p>
    </div>
  </section>

  <!-- 2) Number Guess (1–100) -->
  <section class="card" id="ex12">
    <header><h2>2) Number Guess (1–100)</h2></header>
    <div class="body">
      <div class="meta">Guess the secret number in as few tries as possible.</div>
      <span class="tag">state</span><span class="tag">conditionals</span>
      <div class="row" style="margin-top:6px">
        <input id="ex12-in" type="number" min="1" max="100" placeholder="1–100" />
        <button id="ex12-try">Try</button>
        <button id="ex12-new">New</button>
      </div>
      <div class="row"><span>Hint: <b id="ex12-hint">—</b></span><span>Tries: <b id="ex12-tries">0</b></span></div>
      <p class="note"><strong>How it works:</strong> Generate a random integer and give “higher/lower” hints until correct.</p>
      <p class="small"><strong>Learning outcomes:</strong> input handling, numeric comparisons, simple game state.</p>
    </div>
  </section>
```

```
 <!-- 3) Word Scramble -->
 <section class="card" id="ex13">
   <header><h2>3) Word Scramble</h2></header>
   <div class="body">
     <div class="meta">Unscramble the word before the time runs out.</div>
     <span class="tag">strings</span><span class="tag">timers</span>
     <div class="row" style="margin-top:6px">
       <button id="ex13-start">Start</button>
       <span>⏱ <b id="ex13-t">12</b>s</span>
     </div>
     <div style="margin:6px 0">Scrambled: <b id="ex13-scr">—</b></div>
     <div class="row"><input id="ex13-in" type="text" placeholder="type answer"
disabled /><button id="ex13-check" disabled>Check</button></div>
     <div>Result: <b id="ex13-res">—</b></div>
     <p class="note"><strong>How it works:</strong> Pick a word, scramble it by
shuffling letters, and compare the user's guess.</p>
     <p class="small"><strong>Learning outcomes:</strong> string manipulation,
shuffle, enabling/disabling inputs.</p>
   </div>
 </section>

 <!-- 4) Color Name (Stroop) -->
 <section class="card" id="ex14">
   <header><h2>4) Color Name (Stroop)</h2></header>
   <div class="body">
     <div class="meta">Type the <i>color of the text</i>, not the word itself.</div>
     <span class="tag">styles</span><span class="tag">edge cases</span>
     <div class="row" style="margin-top:6px">
       <button id="ex14-new">New</button>
       <input id="ex14-in" type="text" placeholder="color name (e.g., red)" />
       <button id="ex14-check">Check</button>
       <span>Score: <b id="ex14-score">0</b></span>
     </div>
     <div id="ex14-word" style="font-size:1.4rem;margin-top:6px">—</div>
     <p class="note"><strong>How it works:</strong> Randomly choose a color word and a
mismatched text color; user must type the text color.</p>
     <p class="small"><strong>Learning outcomes:</strong> reading/writing styles,
case-insensitive compare, UX clarity.</p>
   </div>
```

```
 </section>

 <!-- 5) Hover Path -->
<section class="card" id="ex5">
 <header><h2>5) Hover Path</h2></header>
 <div class="body">
   <div class="meta">Move the mouse from Start (green) to Finish (gold) without
touching red walls.</div>
   <div id="hp-grid" class="hover-grid"
style="grid-template-columns:repeat(8,24px);gap:6px"></div>
   <div class="row" style="margin-top:6px">Status: <b id="hp-status">Hover the green
square to begin</b></div>
 </div>
</section>

 <!-- 6) Key Reaction -->
 <section class="card" id="ex16">
   <header><h2>6) Key Reaction</h2></header>
   <div class="body">
     <div class="meta">Press the shown key as fast as possible.</div>
     <span class="tag">keyboard</span><span class="tag">timing</span>
     <div class="row" style="margin-top:6px">
       <button id="ex16-start">Start</button>
       <span>Press: <b id="ex16-key">—</b></span>
       <span>Time: <b id="ex16-ms">-</b> ms</span>
     </div>
     <p class="note"><strong>How it works:</strong> After a short random delay,
display a key to press and measure reaction time.</p>
     <p class="small"><strong>Learning outcomes:</strong> keydown handling, guarding
premature inputs, timing diffs.</p>
   </div>
 </section>

 <!-- 7) Fill the Bar (Exact Target) -->
 <section class="card" id="ex17">
   <header><h2>7) Fill the Bar (Exact Target)</h2></header>
   <div class="body">
     <div class="meta">Click to add 7% per click and try to land exactly on the target
percent.</div>
```

```
      <span class="tag">state</span><span class="tag">progress</span>
      <div class="row" style="margin-top:6px">
        <span>Target: <b id="ex17-target">—</b>%</span>
        <button id="ex17-new">New</button>
        <button id="ex17-add">+7%</button>
        <button id="ex17-reset">Reset</button>
      </div>
      <div class="bar" style="margin-top:6px"><i id="ex17-bar"></i></div>
      <div style="margin-top:4px">Now: <b id="ex17-now">0</b>% — <span
id="ex17-msg">—</span></div>
      <p class="note"><strong>How it works:</strong> Keep adding fixed increments to
hit the target exactly; overshoot fails.</p>
      <p class="small"><strong>Learning outcomes:</strong> arithmetic, modulo thinking,
visual feedback.</p>
    </div>
  </section>

  <!-- 8) Sequence Numbers -->
  <section class="card" id="ex18">
    <header><h2>8) Sequence Numbers</h2></header>
    <div class="body">
      <div class="meta">Remember the flashed sequence and click numbers in order.</div>
      <span class="tag">timeouts</span><span class="tag">state machine</span>
      <div class="row" style="margin-top:6px"><button id="ex18-start">Start</button>
Level: <b id="ex18-level">0</b></div>
      <div class="grid" id="ex18-grid"
style="grid-template-columns:repeat(4,1fr);margin-top:6px"></div>
      <p class="note"><strong>How it works:</strong> Show a sequence by briefly
highlighting cells, then accept user clicks and verify order.</p>
      <p class="small"><strong>Learning outcomes:</strong> async flow, input gating,
correctness checks.</p>
    </div>
  </section>

  <!-- 9) Resize to Fit -->
  <section class="card" id="ex19">
    <header><h2>9) Resize to Fit</h2></header>
    <div class="body">
      <div class="meta">Drag the bottom-right of the box to match the goal size (±10px
```

```
tolerance).</div>
      <span class="tag">layout</span><span class="tag">resize observer</span>
      <div class="row" style="margin-top:6px">
        <div class="goal-box" id="ex19-goal">Target</div>
        <div class="drag-box" id="ex19-box"></div>
      </div>
      <div style="margin-top:6px">Box: <b id="ex19-size">—</b> | Goal: <b
id="ex19-gsize">—</b> | <b id="ex19-msg">Try it</b></div>
      <p class="note"><strong>How it works:</strong> Display a random goal size and
watch the resizable box using <code>ResizeObserver</code>.</p>
      <p class="small"><strong>Learning outcomes:</strong> reading layout sizes,
tolerance checks, observers.</p>
    </div>
  </section>

  <!-- 10) Treasure Grid (LocalStorage best) -->
  <section class="card" id="ex20">
    <header><h2>10) Treasure Grid (LocalStorage)</h2></header>
    <div class="body">
      <div class="meta">Find the hidden treasure. Fewer clicks = better score (saved
locally).</div>
      <span class="tag">storage</span><span class="tag">grid</span>
      <div class="grid" id="ex20-grid"
style="margin-top:6px;grid-template-columns:repeat(6,1fr)"></div>
      <div class="row" style="margin-top:6px">Clicks: <b id="ex20-clicks">0</b> | Best:
<b id="ex20-best">—</b> <button id="ex20-new">New</button></div>
      <p class="note"><strong>How it works:</strong> Random treasure index; clicking
cells reveals miss/treasure. Lowest clicks is persisted in
<code>localStorage</code>.</p>
      <p class="small"><strong>Learning outcomes:</strong> persistent state, simple
reveal logic, grid indexing.</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>
/* Utilities */
const $ = s => document.querySelector(s);
const $$ = s => Array.from(document.querySelectorAll(s));
const rand = n => Math.floor(Math.random()*n);
const wait = ms => new Promise(r=>setTimeout(r,ms));
function shuffle(a){ return
a.map(v=>[Math.random(),v]).sort((x,y)=>x[0]-y[0]).map(x=>x[1]); }

/* 11) Moving Target */
(function(){
 const arena = $('#ex11-arena'), target=$('#ex11-target'), start=$('#ex11-start');
 const scoreEl=$('#ex11-score'), timeEl=$('#ex11-time');
 let score=0, t=15, loop=null, timer=null, playing=false;

 function move(){
   const rect = arena.getBoundingClientRect();
   const maxX = rect.width - target.offsetWidth;
   const maxY = rect.height - target.offsetHeight;
   target.style.left = Math.max(0, rand(maxX)) + 'px';
   target.style.top  = Math.max(0, rand(maxY)) + 'px';
 }

 start.addEventListener('click', ()=>{
   if(playing) return;
   playing=true; score=0; t=15; scoreEl.textContent=score; timeEl.textContent=t;
   loop = setInterval(move, 450);
   timer = setInterval(()=>{ t--; timeEl.textContent=t; if(t<=0){ clearInterval(loop);
clearInterval(timer); playing=false; } },1000);
 });

 target.addEventListener('click', ()=>{ if(playing){ score++;
scoreEl.textContent=score; move(); } });

 window._ex11 = {start,target,arena,scoreEl,timeEl};
```

```
})();

/* 12) Number Guess */
(function(){
 const input=$('#ex12-in'), btn=$('#ex12-try'), hint=$('#ex12-hint'),
triesEl=$('#ex12-tries'), newBtn=$('#ex12-new');
 let secret=1+rand(100), tries=0;
 function reset(){ secret=1+rand(100); tries=0; triesEl.textContent=tries;
hint.textContent='—'; input.value=''; }
 btn.addEventListener('click', ()=>{
   const n = +input.value;
   if(!n) return;
   tries++; triesEl.textContent=tries;
   hint.textContent = n===secret ? 'Correct!' : (n<secret ? 'Higher' : 'Lower');
 });
 newBtn.addEventListener('click', reset);
 reset();
 window._ex12 = {input,btn,hint,triesEl,newBtn};
})();

/* 13) Word Scramble */
(function(){
 const words =
['javascript','browser','library','promise','closure','iterator','element','event','co
ntext','frontend'];
 const start=$('#ex13-start'), tEl=$('#ex13-t'), scr=$('#ex13-scr'),
inp=$('#ex13-in'), check=$('#ex13-check'), res=$('#ex13-res');
 let target='', t=12, timer=null, running=false;

 function scramble(w){ return shuffle(w.split('')).join(''); }

 start.addEventListener('click', ()=>{
   if(running) return;
   running=true; t=12; target = words[rand(words.length)];
   scr.textContent = scramble(target);
   tEl.textContent=t; res.textContent='—';
   inp.disabled=false; inp.value=''; check.disabled=false; inp.focus();
   clearInterval(timer);
   timer=setInterval(()=>{ t--; tEl.textContent=t; if(t<=0){ clearInterval(timer);
```

```
running=false; inp.disabled=true; check.disabled=true; res.textContent='Time!
('+target+')'; } },1000);
 });

 check.addEventListener('click', ()=>{
   if(!running) return;
   if(inp.value.trim().toLowerCase()===target){ res.textContent='Correct';
running=false; clearInterval(timer); inp.disabled=true; check.disabled=true; }
   else { res.textContent='Nope'; }
 });

 window._ex13 = {start,check,inp,scr,tEl,res};
})();

/* 14) Stroop Color Name */
(function(){
 const colors = ['red','green','blue','orange','purple','yellow','pink','cyan'];
 const word=$('#ex14-word'), input=$('#ex14-in'), check=$('#ex14-check'),
scoreEl=$('#ex14-score'), btn=$('#ex14-new');
 let targetColor='';

 function newRound(){
   const text = colors[rand(colors.length)];
   const col = colors[rand(colors.length)];
   targetColor = col;
   word.textContent = text.toUpperCase();
   word.style.color = col;
   input.value='';
 }

 btn.addEventListener('click', newRound);
 check.addEventListener('click', ()=>{
   const guess = input.value.trim().toLowerCase();
   if(!guess) return;
   if(guess===targetColor){ scoreEl.textContent = (+scoreEl.textContent||0)+1;
newRound(); }
 });

 newRound();
```

```
 window._ex14 = {btn,check,input,word,scoreEl};
})();

/* 15) Hover Path */

(function(){
 const gridEl = document.querySelector('#hp-grid');
 const statusEl = document.querySelector('#hp-status');

 // Layout (8 x 5 = 40 cells). 'S' = start, 'F' = finish, 1 = wall, 0 = empty
 const layout = [
   'S',0, 0, 0, 1, 0, 0, 0,
    1 ,1, 0, 0, 1, 0, 1, 0,
    0 ,0, 0, 1, 0, 0, 1, 0,
    0 ,1, 0, 0, 0, 1, 0, 0,
    0 ,0, 0, 1,'F',0, 0, 0
 ];

 const cells = [];
 let playing = false;

 function build(){
   gridEl.innerHTML = '';
   cells.length = 0;
   layout.forEach((v,i)=>{
     const d = document.createElement('div');
     d.className = 'hcell';
     if (v === 1) d.classList.add('wall');
     if (v === 'S') d.classList.add('start');
     if (v === 'F') d.classList.add('finish');
     d.dataset.type = (v===1?'wall': v==='S'?'start': v==='F'?'finish':'empty');
     gridEl.appendChild(d);
     cells.push(d);
   });
   statusEl.textContent = 'Hover the green square to begin';
   cells.forEach(c=>c.classList.remove('path'));
   playing = false;
 }
```

```
function lose(reason='You hit a wall!'){
  playing = false;
  statusEl.textContent = `💥 ${reason} (move over Start to try again)`;
  gridEl.classList.add('hp-flash');
  setTimeout(()=>gridEl.classList.remove('hp-flash'), 300);
}

function win(){
  playing = false;
  statusEl.textContent = '🎉 You made it! Great job!';
}

// Use mouseover (bubbles) so the parent can listen for child entry
gridEl.addEventListener('mouseover', (e)=>{
  const cell = e.target.closest('.hcell');
  if (!cell || !gridEl.contains(cell)) return;

  const t = cell.dataset.type;

  // Start: arm the run and clear old trail
  if (!playing && t === 'start') {
    playing = true;
    statusEl.textContent = 'Go! Move carefully to the Finish…';
    cells.forEach(c=>c.classList.remove('path'));
    return;
  }

  if (!playing) return; // not armed yet

  // Collision checks while playing
  if (t === 'wall') {
    lose();
    return;
  }
  if (t === 'finish') {
    win();
    return;
  }
```

```
    // Paint a trail on safe cells
    if (t === 'empty') {
      cell.classList.add('path');
    }
  });

  // Leaving the maze counts as a fail if mid-run
  gridEl.addEventListener('mouseleave', ()=>{
    if (playing) lose('You left the maze!');
  });

  build();
})();



/* 16) Key Reaction */
(function(){
  const start=$('#ex16-start'), keyEl=$('#ex16-key'), msEl=$('#ex16-ms');
  const keys = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.split('');
  let target='', armed=false, shownAt=0, to=null;

  start.addEventListener('click', ()=>{
    armed=false; keyEl.textContent='—'; msEl.textContent='-'; clearTimeout(to);
    to=setTimeout(()=>{
      target = keys[rand(keys.length)];
      keyEl.textContent=target;
      shownAt = Date.now(); armed=true;
    }, 300 + rand(1200));
  });

  window.addEventListener('keydown', (e)=>{
    if(!armed) return;
    if(e.key.toUpperCase()===target){ msEl.textContent = String(Date.now()-shownAt);
armed=false; }
  });

  window._ex16 = {start,keyEl,msEl};
})();
```

```
/* 17) Fill the Bar */
(function(){
 const tEl=$('#ex17-target'), add=$('#ex17-add'), reset=$('#ex17-reset'),
bar=$('#ex17-bar'), nowEl=$('#ex17-now'), msg=$('#ex17-msg'), newBtn=$('#ex17-new');
 let target=0, now=0;

 function setTarget(){ target = 20 + 5*rand(13); // 20..80 step 5
   tEl.textContent=target; now=0; nowEl.textContent=now; bar.style.width='0%';
msg.textContent='—';
 }
 add.addEventListener('click', ()=>{
   now = Math.min(100, now+7);
   nowEl.textContent=now; bar.style.width=now+'%';
   if(now===target){ msg.textContent='Exact! 🎯'; }
   else if(now>target){ msg.textContent='Overshoot!'; }
   else { msg.textContent='Keep going…'; }
 });
 reset.addEventListener('click', ()=>{ now=0; nowEl.textContent=now;
bar.style.width='0%'; msg.textContent='—'; });
 newBtn.addEventListener('click', setTarget);
 setTarget();

 window._ex17 = {add,reset,newBtn,bar,nowEl,tEl,msg};
})();

/* 18) Sequence Numbers */
(function(){
 const grid=$('#ex18-grid'), start=$('#ex18-start'), levelEl=$('#ex18-level');
 let seq=[], idx=0, listening=false, cells=[];
 for(let i=0;i<8;i++){ const d=document.createElement('div'); d.className='cell';
d.textContent=String(i+1); grid.appendChild(d); cells.push(d); }

 function flash(i){ return new Promise(res=>{ const c=cells[i]; const
prev=c.style.background; c.style.background='#22c55e'; setTimeout(()=>{
c.style.background=''; setTimeout(res,120); },300); }); }

 async function play(){
   listening=false;
```

```
    for(const i of seq){ await flash(i); }
    idx=0; listening=true;
  }

  function next(){
    seq.push(rand(8));
    levelEl.textContent=seq.length;
    play();
  }

  cells.forEach((c,i)=>c.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._ex18 = {start,levelEl,grid};
})();

/* 19) Resize to Fit */
(function(){
  const box=$('#ex19-box'), goal=$('#ex19-goal'), sizeEl=$('#ex19-size'),
gsizeEl=$('#ex19-gsize'), msg=$('#ex19-msg');
  const gw = 120 + rand(80), gh = 70 + rand(70); // goal size
  goal.style.width = gw+'px'; goal.style.height = gh+'px';
  gsizeEl.textContent = gw+'×'+gh;
  const ro = new ResizeObserver(()=>{
    const w = Math.round(box.clientWidth), h=Math.round(box.clientHeight);
    sizeEl.textContent = w+'×'+h;
    msg.textContent = (Math.abs(w-gw)<=10 && Math.abs(h-gh)<=10) ? 'Close enough! ✅' :
'Drag the corner…';
  });
  ro.observe(box);

  window._ex19 = {box,goal,sizeEl,gsizeEl,msg};
})();
```

```
/* 20) Treasure Grid (localStorage) */
(function(){
 const grid=$('#ex20-grid'), clicksEl=$('#ex20-clicks'), bestEl=$('#ex20-best'),
newBtn=$('#ex20-new');
 let idx=-1, clicks=0;
 function build(){
   grid.innerHTML=''; clicks=0; clicksEl.textContent=clicks; idx=rand(36);
   for(let i=0;i<36;i++){
     const d=document.createElement('div'); d.className='cell'; d.textContent='?';
     d.addEventListener('click', ()=>{
       if(d.dataset.done) return;
       clicks++; clicksEl.textContent=clicks; d.dataset.done='1';
       if(i===idx){ d.textContent='💎'; d.style.background='#22c55e'; const best =
+(localStorage.getItem('treasureBest')||0); if(!best||clicks<best){
localStorage.setItem('treasureBest', String(clicks)); }
         bestEl.textContent = localStorage.getItem('treasureBest') || '—';
       } else { d.textContent='·'; d.style.color='#64748b'; }
     });
     grid.appendChild(d);
   }
   bestEl.textContent = localStorage.getItem('treasureBest') || '—';
 }
 newBtn.addEventListener('click', build);
 build();

 window._ex20 = {grid,clicksEl,bestEl,newBtn};
})();

/* ------------------------
  Minimal Test Harness
-------------------------*/
(function(){
 const btn=$('#run-tests'), sum=$('#test-summary'), err=$('#test-errors');
 let logs=[];
 const ok = m=>logs.push({ok:true,msg:m});
 const bad = m=>logs.push({ok:false,msg:m});

 async function test11(){ const {start,arena,target}=window._ex11; start.click();
```

```
await wait(500); ok('ex11: starts and moves'); target.click(); ok('ex11: target
clickable'); }
 async function test12(){ const {input,btn,hint}=window._ex12; input.value='50';
btn.click(); ok('ex12: accepts input, gives hint'); if(hint.textContent!=='—')
ok('ex12: hint updated'); }
 async function test13(){ const {start,check,inp,scr}=window._ex13; start.click();
await wait(50); inp.value=scr.textContent; check.click(); ok('ex13: check works'); }
 async function test14(){ const {btn,check,input}=window._ex14; btn.click();
input.value='red'; check.click(); ok('ex14: round/playable'); }
 async function test15(){ const {grid}=window._ex15; const start =
grid.querySelector('.start'); start && start.dispatchEvent(new
Event('mouseenter',{bubbles:true})); ok('ex15: start hoverable'); }
 async function test16(){ const {start}=window._ex16; start.click(); ok('ex16:
starts'); }
 async function test17(){ const {add}=window._ex17; add.click(); ok('ex17: add
updates'); }
 async function test18(){ const {start}=window._ex18; start.click(); await wait(900);
ok('ex18: level started'); }
 async function test19(){ const {box}=window._ex19; box.style.width='150px';
box.style.height='90px'; ok('ex19: resizable box exists'); }
 async function test20(){ const {grid}=window._ex20; const
c=grid.querySelector('.cell'); c && c.click(); ok('ex20: grid clickable'); }

 btn.addEventListener('click', async ()=>{
   logs=[];
   const
tests=[test11,test12,test13,test14,test15,test16,test17,test18,test19,test20];
   for(const t of tests){
     try{ await t(); }catch(e){ bad(t.name+': '+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');
 });
})();
```

```
</script>
</body>
</html>
```