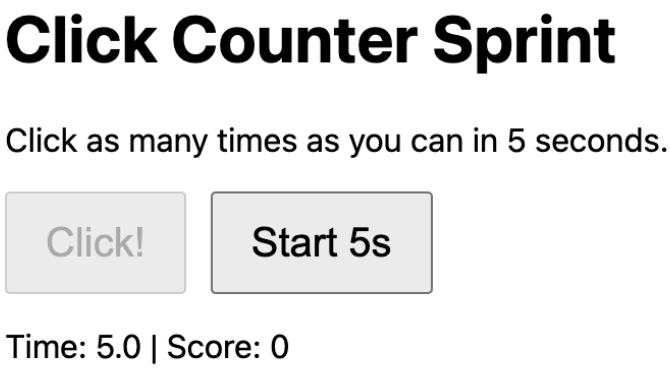
Get Source Code

https://github.com/lsvekis/JavaScript-Games



1) Click Counter Sprint

Goal: Click as many times as possible in 5 seconds.
Practices: timers, state, DOM updates.

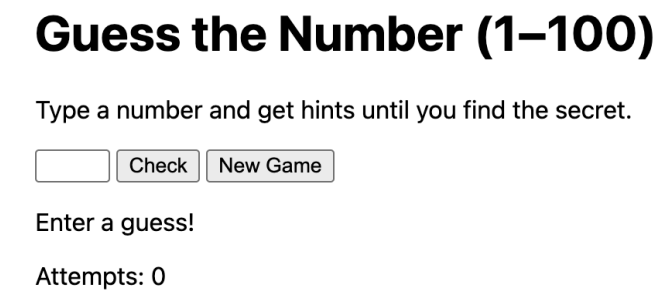
```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Click Sprint</title>
  <style>
    body{font-family:system-ui;padding:20px}
    button{font-size:20px;padding:12px 18px}
    .row{display:flex;gap:12px;align-items:center;margin:10px 0}
  </style>
</head>
<body>
  <h1>Click Counter Sprint</h1>
  <div class="row">
    <button id="btn" disabled>Click!</button>
    <button id="start">Start 5s</button>
  </div>
```

```
  <p>Time: <span id="time">5.0</span> | Score: <span
id="score">0</span></p>
  <p id="result"></p>

<script>
const btn = document.querySelector("#btn");
const startBtn = document.querySelector("#start");
const timeEl = document.querySelector("#time");
const scoreEl = document.querySelector("#score");
const resultEl = document.querySelector("#result");

let score = 0;
let timeLeft = 5.0;
let timerId = null;
let running = false;

btn.addEventListener("click", () => {
  if (!running) return;
  score++;
  scoreEl.textContent = score;
});

startBtn.addEventListener("click", () => {
  if (running) return;
  running = true;
  score = 0;
  timeLeft = 5.0;
  scoreEl.textContent = score;
  resultEl.textContent = "";
  btn.disabled = false;

  const startTime = performance.now();
  timerId = setInterval(() => {
    const elapsed = (performance.now() - startTime) / 1000;
    timeLeft = Math.max(0, 5 - elapsed);
    timeEl.textContent = timeLeft.toFixed(1);

    if (timeLeft <= 0) {
```

```
      clearInterval(timerId);
      running = false;
      btn.disabled = true;
      resultEl.textContent = `Time! Final score: ${score}`;
    }
  }, 50);
});
</script>
</body>
</html>
```

Explanation

- `score`, `timeLeft`, `running` are the *game state*.
- `start` resets state and starts a timer.
- Using `performance.now()` gives smooth timing (not drifting as badly as counting interval ticks).
- Clicking increments score only when `running` is true.



2) Guess the Number

Goal: Guess a random number 1–100 with hints.
Practices: random numbers, inputs, conditionals.

```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Guess</title></head>
<body style="font-family:system-ui;padding:20px">
  <h1>Guess the Number (1–100)</h1>
  <input id="guess" type="number" min="1" max="100" />
```

```
  <button id="check">Check</button>
  <button id="new">New Game</button>
  <p id="msg"></p>
  <p>Attempts: <span id="tries">0</span></p>

<script>
let secret, tries;
const guessEl = document.querySelector("#guess");
const msgEl = document.querySelector("#msg");
const triesEl = document.querySelector("#tries");

function newGame(){
  secret = Math.floor(Math.random()*100)+1;
  tries = 0;
  triesEl.textContent = tries;
  msgEl.textContent = "Enter a guess!";
  guessEl.value = "";
  guessEl.focus();
}
newGame();

document.querySelector("#check").addEventListener("click", () => {
  const g = Number(guessEl.value);
  if (!Number.isInteger(g) || g < 1 || g > 100) {
    msgEl.textContent = "Please enter an integer 1–100.";
    return;
  }
  tries++;
  triesEl.textContent = tries;

  if (g === secret) msgEl.textContent = `Correct! It was ${secret}.`;
  else if (g < secret) msgEl.textContent = "Too low!";
  else msgEl.textContent = "Too high!";
});

document.querySelector("#new").addEventListener("click", newGame);
</script>
</body>
```

```
</html>
```

Explanation

- `secret` is the randomly generated target.
- Every click validates input, increments attempts, then compares g to `secret`.
- `newGame()` resets everything so your UI never “inherits” old state.

Rock Paper Scissors

Pick a move. CPU picks randomly. Track wins/losses/ties.

Rock Paper Scissors

Wins: 0 Losses: 0 Ties: 0

3) Rock Paper Scissors

Goal: Play against the computer; track score.
Practices: arrays, random choice, game rules.

```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>RPS</title></head>
<body style="font-family:system-ui;padding:20px">
  <h1>Rock Paper Scissors</h1>
  <div>
    <button data-m="rock">Rock</button>
    <button data-m="paper">Paper</button>
    <button data-m="scissors">Scissors</button>
  </div>
  <p id="out"></p>
  <p>Wins: <span id="w">0</span> Losses: <span id="l">0</span> Ties:
<span id="t">0</span></p>

<script>
const moves = ["rock","paper","scissors"];
let w=0,l=0,t=0;
const out = document.querySelector("#out");
```

```
function winner(player, cpu){
  if (player === cpu) return "tie";
  if (
    (player==="rock" && cpu==="scissors") ||
    (player==="paper" && cpu==="rock") ||
    (player==="scissors" && cpu==="paper")
  ) return "win";
  return "lose";
}

document.body.addEventListener("click", (e) => {
  const btn = e.target.closest("button[data-m]");
  if (!btn) return;

  const player = btn.dataset.m;
  const cpu = moves[Math.floor(Math.random()*moves.length)];
  const res = winner(player, cpu);

  if (res==="win") w++;
  else if (res==="lose") l++;
  else t++;

  document.querySelector("#w").textContent = w;
  document.querySelector("#l").textContent = l;
  document.querySelector("#t").textContent = t;

  out.textContent = `You: ${player} | CPU: ${cpu} →
${res.toUpperCase()}`;
});
</script>
</body>
</html>
```

Explanation

- `moves[ ]` stores valid choices.

- CPU move uses `Math.floor(Math.random()*3)`.

- `winner()` encodes rules once; UI just uses its result.
- Event delegation (`document.body.addEventListener`) avoids attaching 3 separate listeners.

Coin Flip Streak

Flip a coin and try to get 5 HEADS in a row.

Flip Reset

Current streak (Heads): 0

4) Coin Flip Streak

Goal: Flip a coin; try to hit a streak of 5 heads.
Practices: randomness, streak logic.

```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Coin Streak</title></head>
<body style="font-family:system-ui;padding:20px">
  <h1>Coin Flip Streak</h1>
  <button id="flip">Flip</button>
  <button id="reset">Reset</button>
  <p id="log"></p>
  <p>Current streak (Heads): <span id="streak">0</span></p>

<script>
let streak = 0;
const log = document.querySelector("#log");
const streakEl = document.querySelector("#streak");

document.querySelector("#flip").addEventListener("click", () => {
  const heads = Math.random() < 0.5;
  log.textContent = heads ? "HEADS!" : "TAILS!";
  if (heads) streak++;
  else streak = 0;
  streakEl.textContent = streak;
  if (streak >= 5) log.textContent = "HEADS! 🎉 You hit a 5 streak!";
```

```
});

document.querySelector("#reset").addEventListener("click", () => {
  streak = 0;
  streakEl.textContent = 0;
  log.textContent = "";
});
</script>
</body>
</html>
```

Explanation

- A coin flip is a random boolean.
- Streak increments on heads, resets on tails.
- Win condition: `streak >= 5`.

Reaction Time

Click to start. Wait for green. Click as fast as you can.

5) Reaction Time Tester

Goal: Click when the screen turns green; measure reaction time.
Practices: time measurement, random delays.

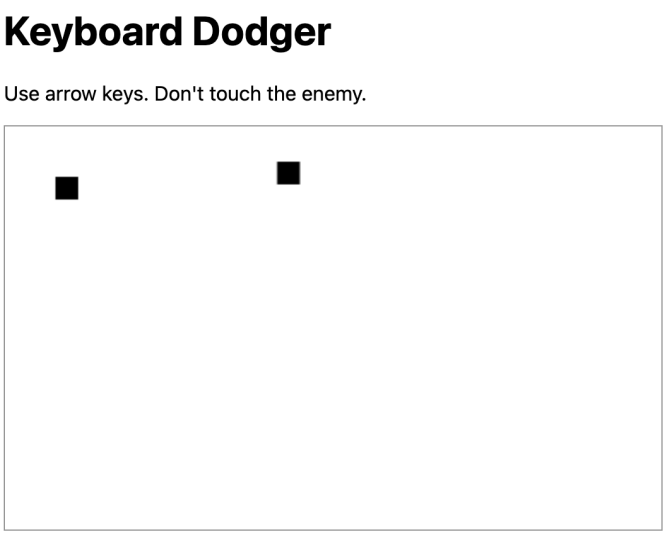
```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Reaction</title>
<style>
```

```
#box{width:320px;height:180px;border-radius:16px;display:grid;place-it
ems:center;

background:#ccc;user-select:none;cursor:pointer;font-family:system-ui;
font-size:20px}
</style></head>
<body style="padding:20px">
  <h1>Reaction Time</h1>
  <div id="box">Click to start</div>
  <p id="msg"></p>

<script>
const box = document.querySelector("#box");
const msg = document.querySelector("#msg");

let state = "idle"; // idle -> waiting -> go
let startMs = 0;
let timeoutId = null;

box.addEventListener("click", () => {
  if (state === "idle") {
    state = "waiting";
    msg.textContent = "Wait for green...";
    box.style.background = "#ccc";
    box.textContent = "Waiting...";

    const delay = 800 + Math.random()*2000;
    timeoutId = setTimeout(() => {
      state = "go";
      box.style.background = "#8f8";
      box.textContent = "CLICK!";
      startMs = performance.now();
    }, delay);
  } else if (state === "waiting") {
    clearTimeout(timeoutId);
    state = "idle";
    msg.textContent = "Too soon! Try again.";
```

```
    box.textContent = "Click to start";
    box.style.background = "#ccc";
  } else if (state === "go") {
    const rt = performance.now() - startMs;
    state = "idle";
    msg.textContent = `Reaction time: ${rt.toFixed(0)} ms`;
    box.textContent = "Click to start";
    box.style.background = "#ccc";
  }
});
</script>
</body>
</html>
```

Explanation

- The game uses a **state machine**:
    - `idle`: ready to start
    - `waiting`: random delay before “go”
    - go: timer started; click ends the round
- Clicking early cancels the timeout and resets.



6) Keyboard Dodger (Move a Square)

Goal: Move your player with arrow keys and avoid a bouncing enemy.
Practices: canvas, keyboard input, game loop.

```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Dodger</title></head>
<body style="font-family:system-ui;padding:20px">
  <h1>Keyboard Dodger</h1>
  <canvas id="c" width="520" height="320" style="border:1px solid
#999"></canvas>
  <p id="status"></p>

<script>
const c = document.querySelector("#c");
const ctx = c.getContext("2d");
const status = document.querySelector("#status");

const player = {x:40,y:40,w:18,h:18,v:3};
const enemy = {x:200,y:120,w:18,h:18,vx:2.6,vy:2.0};
const keys = new Set();
let alive = true;

addEventListener("keydown", e => keys.add(e.key));
addEventListener("keyup", e => keys.delete(e.key));
```

```
function rectHit(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;
}

function tick(){
  ctx.clearRect(0,0,c.width,c.height);

  if (alive){
    // move player
    if (keys.has("ArrowLeft")) player.x -= player.v;
    if (keys.has("ArrowRight")) player.x += player.v;
    if (keys.has("ArrowUp")) player.y -= player.v;
    if (keys.has("ArrowDown")) player.y += player.v;

    // keep inside canvas
    player.x = Math.max(0, Math.min(c.width-player.w, player.x));
    player.y = Math.max(0, Math.min(c.height-player.h, player.y));

    // move enemy and bounce
    enemy.x += enemy.vx; enemy.y += enemy.vy;
    if (enemy.x <= 0 || enemy.x + enemy.w >= c.width) enemy.vx *= -1;
    if (enemy.y <= 0 || enemy.y + enemy.h >= c.height) enemy.vy *= -1;

    if (rectHit(player, enemy)){
      alive = false;
      status.textContent = "Game over! Refresh to retry.";
    }
  }

  // draw
  ctx.fillRect(player.x, player.y, player.w, player.h);
  ctx.fillRect(enemy.x, enemy.y, enemy.w, enemy.h);

  requestAnimationFrame(tick);
}
tick();
</script>
```

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

Explanation

- `requestAnimationFrame()` is your game loop (runs ~60fps).
- Keyboard input is tracked with a `Set` of active keys.
- Collision uses rectangle overlap math.
- Bouncing is flipping velocity when you hit edges.

Score: 27

7) Simple Pong (One Paddle)

Goal: Keep the ball alive with a paddle.
Practices: canvas physics, collisions.

```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Pong Lite</title></head>
<body style="font-family:system-ui;padding:20px">
<canvas id="c" width="520" height="320" style="border:1px solid
#999"></canvas>
<p>Score: <span id="s">0</span></p>

<script>
const c = document.querySelector("#c");
const ctx = c.getContext("2d");
```

```
const sEl = document.querySelector("#s");

const paddle = {x:20,y:120,w:12,h:80};
const ball = {x:260,y:160,r:8,vx:-3.2,vy:2.2};
let score = 0;

c.addEventListener("mousemove", (e) => {
  const rect = c.getBoundingClientRect();
  const y = e.clientY - rect.top;
  paddle.y = y - paddle.h/2;
  paddle.y = Math.max(0, Math.min(c.height-paddle.h, paddle.y));
});

function tick(){
  ctx.clearRect(0,0,c.width,c.height);

  // move ball
  ball.x += ball.vx;
  ball.y += ball.vy;

  // top/bottom bounce
  if (ball.y-ball.r <= 0 || ball.y+ball.r >= c.height) ball.vy *= -1;

  // paddle collision
  const hitPaddle =
    ball.x - ball.r <= paddle.x + paddle.w &&
    ball.y >= paddle.y && ball.y <= paddle.y + paddle.h &&
    ball.x > paddle.x;

  if (hitPaddle){
    ball.vx *= -1.05; // speed up a bit
    score++;
    sEl.textContent = score;
  }

  // miss
  if (ball.x + ball.r < 0){
    score = 0;
```

```
    sEl.textContent = score;
    ball.x = 260; ball.y = 160;
    ball.vx = -3.2; ball.vy = 2.2;
  }

  // draw
  ctx.fillRect(paddle.x,paddle.y,paddle.w,paddle.h);
  ctx.beginPath();
  ctx.arc(ball.x,ball.y,ball.r,0,Math.PI*2);
  ctx.fill();

  requestAnimationFrame(tick);
}
tick();
</script>
</body>
</html>
```

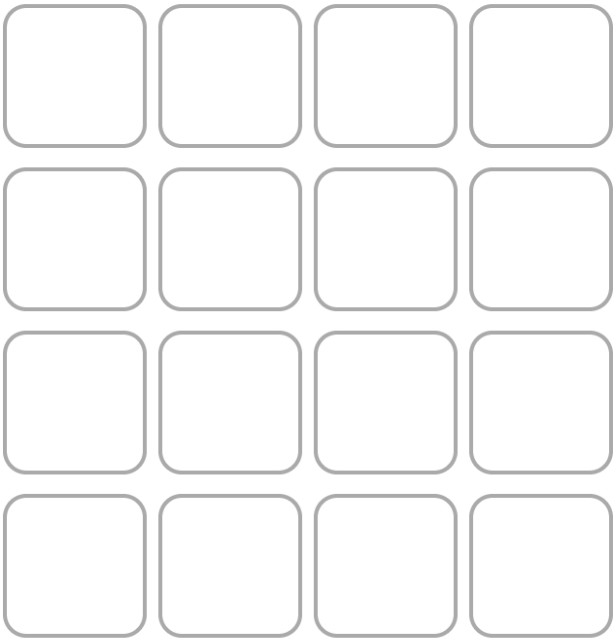
Explanation

- Paddle follows the mouse inside canvas.
- Ball bounces off top/bottom by flipping `vy`.
- Paddle hit detection checks if ball's left edge overlaps paddle region.
- Reset when the ball exits left side.

Whack-a-Mole

Start

Score: 0 | Time: 15



8) Whack-a-Mole (Grid)

Goal: Click the highlighted cell before it moves.
Practices: grid DOM, intervals, event handling.

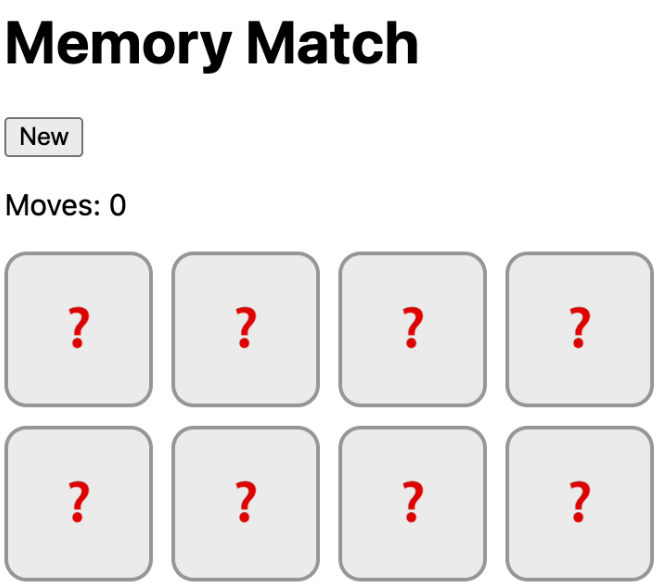
```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Whack</title>
<style>
  #grid{display:grid;grid-template-columns:repeat(4,70px);gap:10px}
  .cell{width:70px;height:70px;border:2px solid
#aaa;border-radius:12px;cursor:pointer}
  .on{background:#ffd56a}
</style></head>
<body style="font-family:system-ui;padding:20px">
  <h1>Whack-a-Mole</h1>
  <button id="start">Start</button>
```

```
  <p>Score: <span id="score">0</span> | Time: <span
id="time">15</span></p>
  <div id="grid"></div>

<script>
const grid = document.querySelector("#grid");
const scoreEl = document.querySelector("#score");
const timeEl = document.querySelector("#time");

const cells = [];
for (let i=0;i<16;i++){
  const d = document.createElement("div");
  d.className = "cell";
  d.dataset.i = i;
  grid.appendChild(d);
  cells.push(d);
}

let score=0, time=15, active=-1, moleId=null, clockId=null;

function pickMole(){
  if (active >= 0) cells[active].classList.remove("on");
  let next;
  do next = Math.floor(Math.random()*cells.length);
  while (next === active);
  active = next;
  cells[active].classList.add("on");
}

grid.addEventListener("click", (e) => {
  const cell = e.target.closest(".cell");
  if (!cell) return;
  if (Number(cell.dataset.i) === active){
    score++;
    scoreEl.textContent = score;
    pickMole();
  }
});
```

```
document.querySelector("#start").addEventListener("click", () => {
  score=0; time=15;
  scoreEl.textContent = score;
  timeEl.textContent = time;

  pickMole();

  clearInterval(moleId);
  clearInterval(clockId);

  moleId = setInterval(pickMole, 700);
  clockId = setInterval(() => {
    time--;
    timeEl.textContent = time;
    if (time <= 0){
      clearInterval(moleId);
      clearInterval(clockId);
      if (active >= 0) cells[active].classList.remove("on");
      active = -1;
      alert(`Time! Final score: ${score}`);
    }
  }, 1000);
});
</script>
</body>
</html>
```

Explanation

- 16 divs form a clickable grid.
- `active` holds the index of the “mole.”
- Two timers run:
  - one moves the mole,
  - one counts down the time.



9) Memory Match (4 pairs)

Goal: Flip cards to find matching pairs.
Practices: arrays, shuffle, state, matching logic.

```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Memory</title>
<style>
  #board{display:grid;grid-template-columns:repeat(4,80px);gap:10px}
  .card{height:80px;border:2px solid
#999;border-radius:12px;display:grid;place-items:center;
        font-size:28px;cursor:pointer;user-select:none}
  .down{background:#eee}
  .up{background:#c9f}
  .done{background:#8f8;cursor:default}
</style></head>
<body style="font-family:system-ui;padding:20px">
  <h1>Memory Match</h1>
  <button id="new">New</button>
  <p>Moves: <span id="moves">0</span></p>
  <div id="board"></div>

<script>
const symbols = ["🍎","🍎","⭐","⭐","🐱","🐱","⚽","⚽"];
```

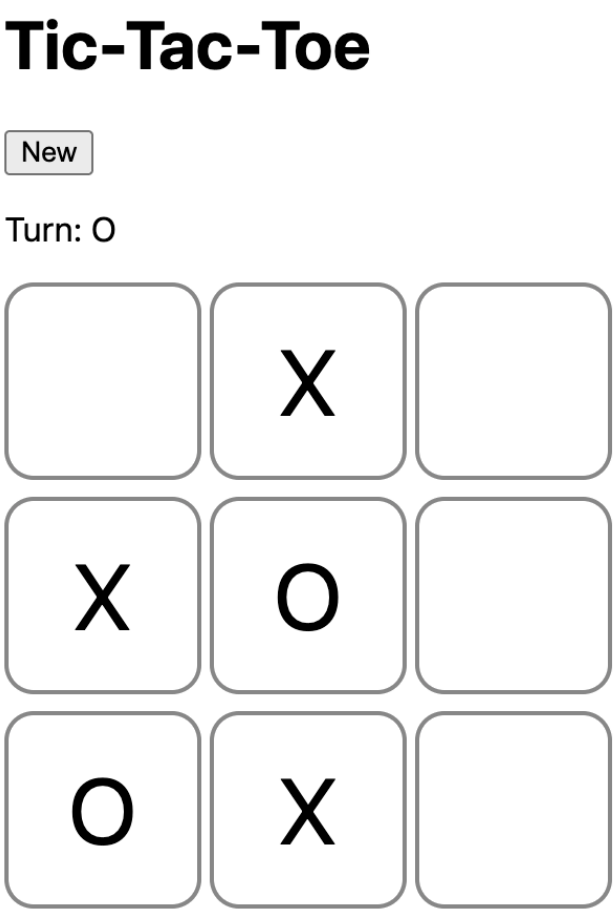
```
const board = document.querySelector("#board");
const movesEl = document.querySelector("#moves");
let deck = [];
let first = null;
let lock = false;
let moves = 0;

function shuffle(arr){
  for (let i=arr.length-1;i>0;i--){
    const j = Math.floor(Math.random()*(i+1));
    [arr[i],arr[j]] = [arr[j],arr[i]];
  }
  return arr;
}

function render(){
  board.innerHTML = "";
  deck.forEach((val, idx) => {
    const d = document.createElement("div");
    d.className = "card down";
    d.dataset.idx = idx;
    d.textContent = "❓";
    board.appendChild(d);
  });
}

function newGame(){
  deck = shuffle([...symbols]);
  first = null;
  lock = false;
  moves = 0;
  movesEl.textContent = moves;
  render();
}
newGame();

board.addEventListener("click", (e) => {
  const card = e.target.closest(".card");
```

```
  if (!card || lock) return;

  const i = Number(card.dataset.idx);
  if (card.classList.contains("done") ||
card.classList.contains("up")) return;

  // flip up
  card.classList.remove("down");
  card.classList.add("up");
  card.textContent = deck[i];

  if (!first){
    first = card;
    return;
  }

  // second card
  moves++;
  movesEl.textContent = moves;

  const firstIdx = Number(first.dataset.idx);
  const match = deck[firstIdx] === deck[i];

  if (match){
    first.classList.add("done");
    card.classList.add("done");
    first = null;
  } else {
    lock = true;
    setTimeout(() => {
      first.classList.remove("up");
      first.classList.add("down");
      first.textContent = "❓";

      card.classList.remove("up");
      card.classList.add("down");
      card.textContent = "❓";
```

```
      first = null;
      lock = false;
    }, 650);
  }
});

document.querySelector("#new").addEventListener("click", newGame);
</script>
</body>
</html>
```

Explanation

- `deck` is a shuffled array of symbols.
- `first` stores the first flipped card element.
- `lock` prevents extra clicks while cards flip back.
- A match marks both cards `.done` so they can't be clicked again.



10) Tic-Tac-Toe

Goal: Two-player tic tac toe with win detection.
Practices: arrays, win patterns.

```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Tic Tac Toe</title>
<style>
  #b{display:grid;grid-template-columns:repeat(3,90px);gap:8px}
  .c{width:90px;height:90px;border:2px solid #888;border-radius:14px;

display:grid;place-items:center;font-size:44px;cursor:pointer;user-sel
ect:none}
</style></head>
<body style="font-family:system-ui;padding:20px">
  <h1>Tic-Tac-Toe</h1>
  <button id="new">New</button>
```

```
  <p id="turn">Turn: X</p>
  <div id="b"></div>

<script>
const b = document.querySelector("#b");
const turnEl = document.querySelector("#turn");
const winLines = [
  [0,1,2],[3,4,5],[6,7,8],
  [0,3,6],[1,4,7],[2,5,8],
  [0,4,8],[2,4,6]
];

let board, player, over;

function newGame(){
  board = Array(9).fill("");
  player = "X";
  over = false;
  turnEl.textContent = "Turn: X";
  b.innerHTML = "";
  for (let i=0;i<9;i++){
    const d = document.createElement("div");
    d.className = "c";
    d.dataset.i = i;
    b.appendChild(d);
  }
}
newGame();

function checkWin(p){
  return winLines.some(line => line.every(i => board[i] === p));
}

b.addEventListener("click", (e) => {
  const cell = e.target.closest(".c");
  if (!cell || over) return;
  const i = Number(cell.dataset.i);
  if (board[i]) return;
```

```
  board[i] = player;
  cell.textContent = player;

  if (checkWin(player)){
    over = true;
    turnEl.textContent = `${player} wins!`;
    return;
  }
  if (board.every(v => v)){
    over = true;
    turnEl.textContent = "Draw!";
    return;
  }

  player = player === "X" ? "O" : "X";
  turnEl.textContent = `Turn: ${player}`;
});

document.querySelector("#new").addEventListener("click", newGame);
</script>
</body>
</html>
```

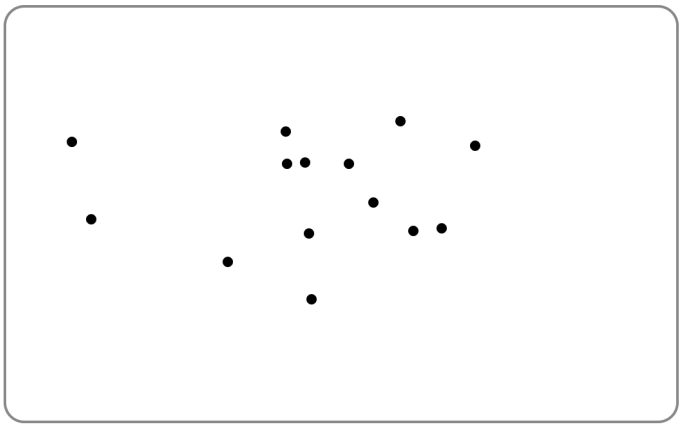
Explanation

- `board[ ]` stores each square (“”, “X”, “O”).
- `winLines` lists all winning index combos.
- After each move:
    - check win,
    - check draw,
    - swap player.

Hot / Cold Hunt

New

Hotter 🔥



11) Hot/Cold Click Hunt

Goal: Find a hidden point; clicks show “hotter/colder”.
Practices: distance math.

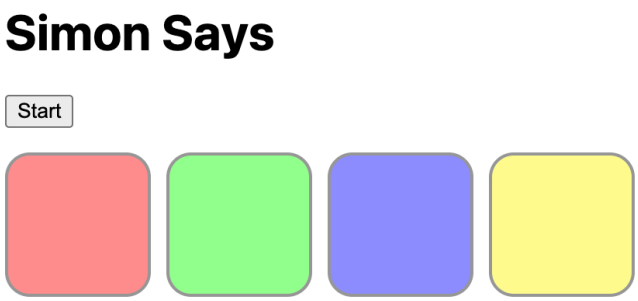
```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Hot Cold</title>
<style>
  #area{width:520px;height:320px;border:2px solid
#888;border-radius:16px;position:relative}

#dot{position:absolute;width:8px;height:8px;border-radius:50%;backgrou
nd:black;transform:translate(-50%,-50%)}
</style></head>
<body style="font-family:system-ui;padding:20px">
  <h1>Hot / Cold Hunt</h1>
  <button id="new">New</button>
  <p id="msg">Click inside the box.</p>
  <div id="area"></div>

<script>
const area = document.querySelector("#area");
const msg = document.querySelector("#msg");
let target = {x:0,y:0};
let lastDist = null;
```

```
function newGame(){
  target.x = Math.random()*area.clientWidth;
  target.y = Math.random()*area.clientHeight;
  lastDist = null;
  msg.textContent = "New target! Click inside the box.";
  area.innerHTML = "";
}
newGame();

function dist(a,b){
  const dx = a.x-b.x, dy=a.y-b.y;
  return Math.hypot(dx, dy);
}

area.addEventListener("click", (e) => {
  const rect = area.getBoundingClientRect();
  const p = {x: e.clientX - rect.left, y: e.clientY - rect.top};

  const d = dist(p, target);

  const dot = document.createElement("div");
  dot.id = "dot";
  dot.style.left = p.x + "px";
  dot.style.top = p.y + "px";
  area.appendChild(dot);

  if (d < 15){
    msg.textContent = "Found it! 🎯 Press New.";
    return;
  }

  if (lastDist === null) msg.textContent = "Keep going...";
  else msg.textContent = d < lastDist ? "Hotter 🔥" : "Colder ❄️";

  lastDist = d;
});
```

```
document.querySelector("#new").addEventListener("click", newGame);
</script>
</body>
</html>
```

Explanation

- The target is a random `(x,y)` inside the play area.
- Each click computes distance with `Math.hypot(dx,dy)`.
- Compare distance to the previous click to say hotter/colder.



12) Mini “Simon Says” (4 buttons)

Goal: Watch a sequence, then repeat it.
Practices: async timing, arrays.

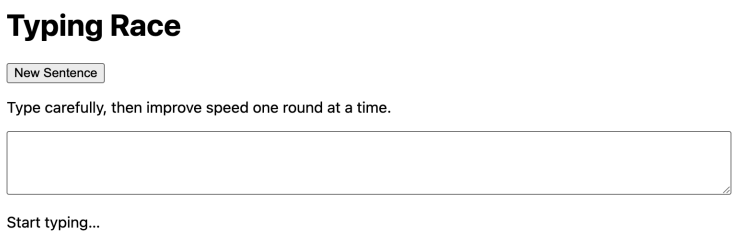
```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Simon</title>
<style>
  .pad{width:90px;height:90px;border-radius:16px;border:2px solid
#777;cursor:pointer;opacity:.75}
  #pads{display:flex;gap:10px}
  .on{opacity:1;outline:4px solid #0002}
</style></head>
<body style="font-family:system-ui;padding:20px">
  <h1>Simon Says</h1>
  <button id="start">Start</button>
  <p id="msg"></p>
  <div id="pads">
    <div class="pad" data-i="0" style="background:#f66"></div>
```

```
    <div class="pad" data-i="1" style="background:#6f6"></div>
    <div class="pad" data-i="2" style="background:#66f"></div>
    <div class="pad" data-i="3" style="background:#ff6"></div>
  </div>

<script>
const pads = [...document.querySelectorAll(".pad")];
const msg = document.querySelector("#msg");
let seq = [];
let input = [];
let accepting = false;

const wait = (ms) => new Promise(r => setTimeout(r, ms));

async function flash(i){
  pads[i].classList.add("on");
  await wait(250);
  pads[i].classList.remove("on");
  await wait(120);
}

async function playSeq(){
  accepting = false;
  msg.textContent = "Watch…";
  for (const i of seq) await flash(i);
  input = [];
  accepting = true;
  msg.textContent = "Your turn!";
}

document.querySelector("#start").addEventListener("click", async () =>
{
  seq = [];
  seq.push(Math.floor(Math.random()*4));
  await playSeq();
});

pads.forEach(p => p.addEventListener("click", async () => {
```

```
  if (!accepting) return;
  const i = Number(p.dataset.i);
  input.push(i);
  await flash(i);

  // validate as you go
  const k = input.length - 1;
  if (input[k] !== seq[k]){
    msg.textContent = "Wrong! Press Start to try again.";
    accepting = false;
    return;
  }

  // round complete
  if (input.length === seq.length){
    msg.textContent = "Nice! Next round…";
    await wait(450);
    seq.push(Math.floor(Math.random()*4));
    await playSeq();
  }
}));
</script>
</body>
</html>
```

Explanation

- `seq` holds the correct pattern.
- `input` stores what the player has clicked.
- `async/await` + `wait()` makes “play sequence with delays” readable.
- Validation happens after every click (good UX).



13) Typing Race (Words per minute)

Goal: Type a sentence; measure speed and accuracy.
Practices: string comparison, timing.

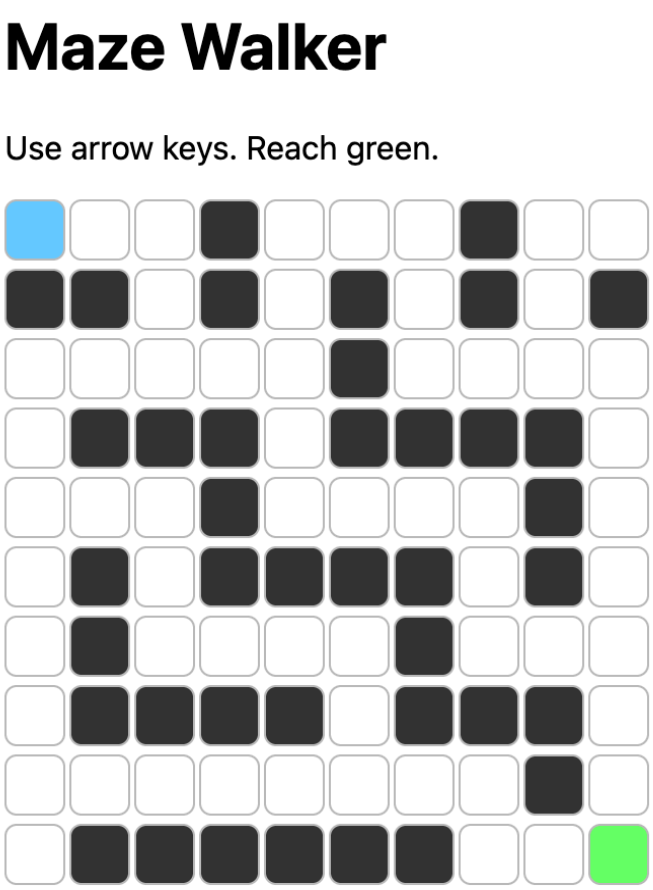
```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Typing Race</title></head>
<body style="font-family:system-ui;padding:20px;max-width:750px">
  <h1>Typing Race</h1>
  <button id="new">New Sentence</button>
  <p id="target"></p>
  <textarea id="in" rows="4" style="width:100%"></textarea>
  <p id="stats"></p>

<script>
const samples = [
  "JavaScript makes the web interactive and fun to build.",
  "Small games are the best way to practice logic and timing.",
  "Type carefully, then improve speed one round at a time."
];

const targetEl = document.querySelector("#target");
const inputEl = document.querySelector("#in");
const statsEl = document.querySelector("#stats");

let target = "";
let start = 0;

function newRound(){
  target = samples[Math.floor(Math.random()*samples.length)];
  targetEl.textContent = target;
  inputEl.value = "";
  statsEl.textContent = "Start typing…";
```

```
  inputEl.focus();
  start = 0;
}
newRound();

inputEl.addEventListener("input", () => {
  if (!start) start = performance.now();

  const typed = inputEl.value;
  const elapsedMin = (performance.now() - start) / 60000;

  const words = typed.trim() ? typed.trim().split(/\s+/).length : 0;
  const wpm = elapsedMin > 0 ? Math.round(words / elapsedMin) : 0;

  // accuracy: compare chars up to typed length
  let correct = 0;
  for (let i=0;i<typed.length;i++){
    if (typed[i] === target[i]) correct++;
  }
  const acc = typed.length ? Math.round((correct/typed.length)*100) :
100;

  statsEl.textContent = `WPM: ${wpm} | Accuracy: ${acc}%`;

  if (typed === target){
    statsEl.textContent += " ✅ Finished!";
  }
});

document.querySelector("#new").addEventListener("click", newRound);
</script>
</body>
</html>
```

Explanation

- Start time begins on first input so “idle time” isn’t counted.
- WPM uses `words / minutes`.

- Accuracy compares typed characters vs target characters in the same positions.



14) Maze Walker (Grid + Walls)

Goal: Move through a tiny maze using arrow keys.
Practices: grid arrays, collision rules.

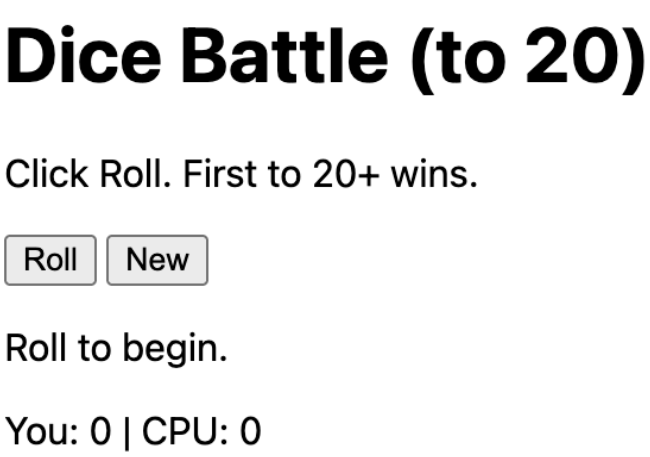
```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Maze</title>
<style>
  #g{display:grid;grid-template-columns:repeat(10,28px);gap:4px}
  .t{width:28px;height:28px;border-radius:6px;border:1px solid #bbb}
  .w{background:#333}
  .p{background:#6cf}
  .e{background:#6f6}
</style></head>
<body style="font-family:system-ui;padding:20px">
```

```
  <h1>Maze Walker</h1>
  <p>Use arrow keys. Reach green.</p>
  <div id="g"></div>
  <p id="msg"></p>

<script>
const gridEl = document.querySelector("#g");
const msg = document.querySelector("#msg");

// 0 empty, 1 wall
const maze = [
  [0,0,0,1,0,0,0,1,0,0],
  [1,1,0,1,0,1,0,1,0,1],
  [0,0,0,0,0,1,0,0,0,0],
  [0,1,1,1,0,1,1,1,1,0],
  [0,0,0,1,0,0,0,0,1,0],
  [0,1,0,1,1,1,1,0,1,0],
  [0,1,0,0,0,0,1,0,0,0],
  [0,1,1,1,1,0,1,1,1,0],
  [0,0,0,0,0,0,0,0,1,0],
  [0,1,1,1,1,1,1,0,0,0],
];

let p = {r:0,c:0};
const end = {r:9,c:9};

function draw(){
  gridEl.innerHTML = "";
  for (let r=0;r<10;r++){
    for (let c=0;c<10;c++){
      const d = document.createElement("div");
      d.className = "t";
      if (maze[r][c] === 1) d.classList.add("w");
      if (r === end.r && c === end.c) d.classList.add("e");
      if (r === p.r && c === p.c) d.classList.add("p");
      gridEl.appendChild(d);
    }
  }
```

```
}
draw();

addEventListener("keydown", (e) => {
  const dir =
{ArrowUp:[-1,0],ArrowDown:[1,0],ArrowLeft:[0,-1],ArrowRight:[0,1]}[e.k
ey];
  if (!dir) return;

  const nr = p.r + dir[0];
  const nc = p.c + dir[1];
  if (nr < 0 || nr > 9 || nc < 0 || nc > 9) return;
  if (maze[nr][nc] === 1) return; // wall blocks movement

  p = {r:nr,c:nc};
  draw();

  if (p.r === end.r && p.c === end.c) msg.textContent = "You escaped!
🎉";
});
</script>
</body>
</html>
```

Explanation

- Maze is a 2D array of 0/1.
- Movement is “attempted,” then rejected if out of bounds or on a wall.
- Rendering is simple: rebuild the grid each move (fine for small boards).



15) Dice Roller Battle (Player vs CPU)

Goal: First to 20 points wins. Each turn roll 1–6.
Practices: turn logic, simple UI state.

```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Dice Battle</title></head>
<body style="font-family:system-ui;padding:20px">
  <h1>Dice Battle (to 20)</h1>
  <button id="roll">Roll</button>
  <button id="new">New</button>
  <p id="log"></p>
  <p>You: <span id="p">0</span> | CPU: <span id="c">0</span></p>

<script>
let p=0, c=0, over=false;
const log = document.querySelector("#log");
const pEl = document.querySelector("#p");
const cEl = document.querySelector("#c");

function die(){ return Math.floor(Math.random()*6)+1; }
function render(){ pEl.textContent=p; cEl.textContent=c; }

function newGame(){
  p=0; c=0; over=false;
  log.textContent="Roll to begin.";
```

```
  render();
}
newGame();

document.querySelector("#roll").addEventListener("click", () => {
  if (over) return;

  const pr = die();
  p += pr;

  const cr = die();
  c += cr;

  log.textContent = `You rolled ${pr}. CPU rolled ${cr}.`;
  render();

  if (p >= 20 || c >= 20){
    over = true;
    if (p === c) log.textContent += " It's a tie!";
    else log.textContent += (p > c) ? " You win!" : " CPU wins!";
  }
});

document.querySelector("#new").addEventListener("click", newGame);
</script>
</body>
</html>
```

Explanation

- Both players roll each turn (simultaneous).
- `over` prevents further gameplay after someone reaches 20.
- Keeping `render()` separate prevents missed UI updates.

Score: 1 Misses: 3

16) Falling Objects Catcher

Goal: Move a basket left/right to catch falling dots.
Practices: canvas loop, spawning, collision.

```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Catcher</title></head>
<body style="font-family:system-ui;padding:20px">
<canvas id="c" width="520" height="320" style="border:1px solid
#999"></canvas>
<p>Score: <span id="s">0</span> Misses: <span id="m">0</span></p>

<script>
const c = document.querySelector("#c");
const ctx = c.getContext("2d");
const sEl = document.querySelector("#s");
const mEl = document.querySelector("#m");

const basket = {x:240,y:290,w:70,h:14,v:5};
const keys = new Set();
let drops = [];
let score=0, miss=0, spawn=0;

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

function spawnDrop(){
```

```
  drops.push({x: Math.random()*c.width, y:-10, r:7, vy:2.2 +
Math.random()*1.5});
}

function tick(){
  ctx.clearRect(0,0,c.width,c.height);

  // move basket
  if (keys.has("ArrowLeft")) basket.x -= basket.v;
  if (keys.has("ArrowRight")) basket.x += basket.v;
  basket.x = Math.max(0, Math.min(c.width-basket.w, basket.x));

  // spawn periodically
  spawn++;
  if (spawn % 45 === 0) spawnDrop();

  // update drops
  drops = drops.filter(d => {
    d.y += d.vy;

    const caught =
      d.x >= basket.x && d.x <= basket.x + basket.w &&
      d.y + d.r >= basket.y && d.y - d.r <= basket.y + basket.h;

    if (caught){
      score++; sEl.textContent = score;
      return false;
    }
    if (d.y - d.r > c.height){
      miss++; mEl.textContent = miss;
      return false;
    }
    return true;
  });

  // draw basket
  ctx.fillRect(basket.x,basket.y,basket.w,basket.h);
```

```
  // draw drops
  for (const d of drops){
    ctx.beginPath();
    ctx.arc(d.x,d.y,d.r,0,Math.PI*2);
    ctx.fill();
  }

  requestAnimationFrame(tick);
}
tick();
</script>
</body>
</html>
```

Explanation

- Droplets are objects in an array.
- Each frame: move basket, maybe spawn a new drop, move drops, check collisions.
- `filter()` removes drops that were caught or missed (keeps array clean).

17) Aim Trainer (Click Targets)

Goal: Click 10 targets as fast as possible.
Practices: positioning, timing, counting.

```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Aim</title>
<style>
  #arena{width:520px;height:320px;border:2px solid
#888;border-radius:16px;position:relative}

.t{position:absolute;width:30px;height:30px;border-radius:50%;backgrou
nd:#f66;cursor:pointer}
</style></head>
<body style="font-family:system-ui;padding:20px">
  <h1>Aim Trainer</h1>
  <button id="start">Start</button>
  <p id="msg"></p>
  <div id="arena"></div>

<script>
const arena = document.querySelector("#arena");
const msg = document.querySelector("#msg");

let hits=0;
let startTime=0;
let targetEl=null;

function placeTarget(){
  if (targetEl) targetEl.remove();
  targetEl = document.createElement("div");
  targetEl.className = "t";
  const x = Math.random()*(arena.clientWidth-30);
  const y = Math.random()*(arena.clientHeight-30);
  targetEl.style.left = x+"px";
  targetEl.style.top = y+"px";
  arena.appendChild(targetEl);

  targetEl.addEventListener("click", () => {
    hits++;
    if (hits >= 10){
      const ms = performance.now() - startTime;
```

```
      msg.textContent = `Done! ${ms.toFixed(0)} ms for 10 hits.`;
      placeTarget(); // optional: leave one
      return;
    }
    placeTarget();
  }, {once:true});
}

document.querySelector("#start").addEventListener("click", () => {
  hits=0;
  startTime = performance.now();
  msg.textContent = "Hit 10 targets!";
  placeTarget();
});
</script>
</body>
</html>
```

Explanation

- A target is a positioned `div` inside a relative container.
- Each click increments hits, and you either place a new target or finish.
- `{once:true}` ensures that target's click handler runs only once.

Game over! Press Space to restart.

18) Endless Runner (Jump Over Blocks)

Goal: Press Space to jump over obstacles.
Practices: gravity, jumping, obstacles, loop.

```
<!doctype html>
<html>
```

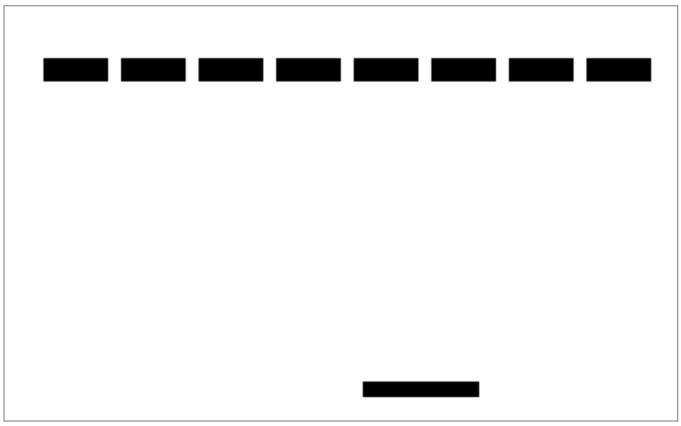
```
<head><meta charset="utf-8"><title>Runner</title></head>
<body style="font-family:system-ui;padding:20px">
<canvas id="c" width="520" height="220" style="border:1px solid
#999"></canvas>
<p id="msg"></p>

<script>
const c = document.querySelector("#c");
const ctx = c.getContext("2d");
const msg = document.querySelector("#msg");

const ground = 190;
const player = {x:60,y:ground,w:18,h:24,vy:0,onGround:true};
let obs = [];
let t = 0;
let alive = true;

addEventListener("keydown", e => {
  if (e.code === "Space" && player.onGround && alive){
    player.vy = -9.5;
    player.onGround = false;
  }
  if (e.code === "Space" && !alive){
    // quick reset
    obs = []; t=0; alive=true; msg.textContent="";
    player.y=ground; player.vy=0; player.onGround=true;
  }
});

function hit(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;
}

function tick(){
  ctx.clearRect(0,0,c.width,c.height);

  // ground line
```

```
ctx.fillRect(0, ground+player.h, c.width, 4);

if (alive){
  // gravity
  player.vy += 0.45;
  player.y += player.vy;

  if (player.y >= ground){
    player.y = ground;
    player.vy = 0;
    player.onGround = true;
  }

  // spawn obstacles
  t++;
  if (t % 90 === 0){
    obs.push({x:c.width+20,y:ground+8,w:16,h:16,vx: -4.2});
  }

  // move obstacles
  obs.forEach(o => o.x += o.vx);
  obs = obs.filter(o => o.x + o.w > -10);

  // collision
  for (const o of obs){
    if (hit(player, o)){
      alive = false;
      msg.textContent = "Game over! Press Space to restart.";
    }
  }
}

// draw player
ctx.fillRect(player.x, player.y, player.w, player.h);

// draw obstacles
obs.forEach(o => ctx.fillRect(o.x,o.y,o.w,o.h));
```

```
  requestAnimationFrame(tick);
}
tick();
</script>
</body>
</html>
```

Explanation

- Jump is setting `vy` negative; gravity increases `vy` each frame.
- Ground collision clamps player to `ground`.
- Obstacles spawn on a timer and move left.
- Restart is a fast reset of arrays and state.



Miss! Refresh to retry.

19) Simple Breakout (One Row)

Goal: Bounce ball to break blocks.
Practices: collision detection, arrays of bricks.

```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Breakout</title></head>
<body style="font-family:system-ui;padding:20px">
<canvas id="c" width="520" height="320" style="border:1px solid
#999"></canvas>
<p id="msg"></p>
```

```
<script>
const c = document.querySelector("#c");
const ctx = c.getContext("2d");
const msg = document.querySelector("#msg");

const paddle = {x:220,y:290,w:90,h:12};
const ball = {x:260,y:160,r:7,vx:3,vy:3};

let bricks = [];
for (let i=0;i<8;i++){
  bricks.push({x:30+i*60,y:40,w:50,h:18,alive:true});
}

c.addEventListener("mousemove", (e) => {
  const r = c.getBoundingClientRect();
  paddle.x = (e.clientX - r.left) - paddle.w/2;
  paddle.x = Math.max(0, Math.min(c.width-paddle.w, paddle.x));
});

function circleRectHit(ball, rect){
  // clamp circle center to rectangle bounds
  const cx = Math.max(rect.x, Math.min(ball.x, rect.x+rect.w));
  const cy = Math.max(rect.y, Math.min(ball.y, rect.y+rect.h));
  const dx = ball.x - cx, dy = ball.y - cy;
  return (dx*dx + dy*dy) <= ball.r*ball.r;
}

function tick(){
  ctx.clearRect(0,0,c.width,c.height);

  ball.x += ball.vx;
  ball.y += ball.vy;

  if (ball.x-ball.r<=0 || ball.x+ball.r>=c.width) ball.vx *= -1;
  if (ball.y-ball.r<=0) ball.vy *= -1;

  // paddle bounce
  if (circleRectHit(ball, paddle) && ball.vy > 0){
```

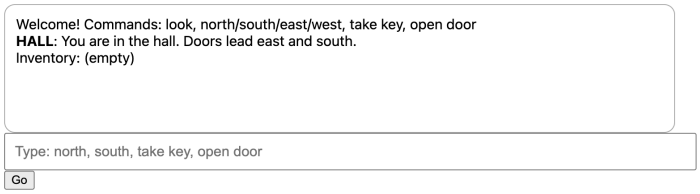
```
    ball.vy *= -1;
  }

  // bricks
  for (const b of bricks){
    if (!b.alive) continue;
    if (circleRectHit(ball, b)){
      b.alive = false;
      ball.vy *= -1;
      break;
    }
  }

  // lose
  if (ball.y - ball.r > c.height){
    msg.textContent = "Miss! Refresh to retry.";
  }

  // win
  if (bricks.every(b => !b.alive)){
    msg.textContent = "You cleared all bricks! 🎉";
  }

  // draw
  ctx.fillRect(paddle.x,paddle.y,paddle.w,paddle.h);

  ctx.beginPath();
  ctx.arc(ball.x,ball.y,ball.r,0,Math.PI*2);
  ctx.fill();

  for (const b of bricks){
    if (!b.alive) continue;
    ctx.fillRect(b.x,b.y,b.w,b.h);
  }

  requestAnimationFrame(tick);
}
tick();
```

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

Explanation

- Bricks are objects with `alive` flags.
- `circleRectHit()` does robust circle-rectangle collision by clamping the circle center to the rectangle and checking radius distance.
- On collision, the brick is removed and the ball bounces.

Find the Key (Text Adventure)



20) “Find the Key” Adventure (Text Game)

Goal: Navigate rooms, find a key, unlock exit.
Practices: objects/maps, simple command parsing.

```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Text Adventure</title></head>
<body style="font-family:system-ui;padding:20px;max-width:750px">
  <h1>Find the Key (Text Adventure)</h1>
  <div id="out" style="border:1px solid
#aaa;border-radius:12px;padding:12px;min-height:120px"></div>
  <input id="cmd" placeholder="Type: north, south, take key, open
door" style="width:100%;padding:10px;font-size:16px" />
  <button id="go">Go</button>

<script>
const out = document.querySelector("#out");
const cmd = document.querySelector("#cmd");

const rooms = {
```

```
  hall:  {desc:"You are in the hall. Doors lead east and south.",
exits:{east:"study", south:"kitchen"}},
  study: {desc:"A quiet study. There is a shiny key here. West returns
to hall.", exits:{west:"hall"}, item:"key"},
  kitchen:{desc:"A kitchen with an exit door to the south. North
returns to hall.", exits:{north:"hall", south:"exit"}}
};

let loc = "hall";
let inv = new Set();
let keyTaken = false;

function print(text){
  out.innerHTML += `<div>${text}</div>`;
  out.scrollTop = out.scrollHeight;
}

function look(){
  const r = rooms[loc];
  print(`<b>${loc.toUpperCase()}</b>: ${r.desc}`);
  if (r.item && !keyTaken) print("You see: key");
  print("Inventory: " + (inv.size ? [...inv].join(", ") : "(empty)"));
}

function handle(raw){
  const s = raw.trim().toLowerCase();
  if (!s) return;

  if (s === "look") { look(); return; }

  if (s.startsWith("take")){
    if (loc === "study" && !keyTaken){
      inv.add("key");
      keyTaken = true;
      print("You take the key.");
    } else {
      print("There is nothing to take.");
    }
```

```
    return;
  }

  if (s === "open door" || s === "open"){
    if (loc !== "kitchen") { print("There is no door here."); return;
}
    if (!inv.has("key")) { print("The door is locked. You need a
key."); return; }
    print("You unlock the door and escape! 🎉");
    return;
  }

  // movement: north/south/east/west
  const r = rooms[loc];
  if (["north","south","east","west"].includes(s)){
    const next = r.exits[s];
    if (!next) { print("You can't go that way."); return; }
    if (next === "exit"){
      print("The exit door is locked. Maybe try: open door");
      return;
    }
    loc = next;
    look();
    return;
  }

  print("Unknown command. Try: look, north/south/east/west, take key,
open door");
}

document.querySelector("#go").addEventListener("click", () => {
  handle(cmd.value);
  cmd.value = "";
  cmd.focus();
});
cmd.addEventListener("keydown", (e) => {
  if (e.key === "Enter") document.querySelector("#go").click();
});
```

```
print("Welcome! Commands: look, north/south/east/west, take key, open
door");
look();
</script>
</body>
</html>
```

Explanation

- `rooms` is a map (object) describing each location and its exits.
- `loc` is current room; `inv` stores inventory.
- The “parser” is just a few string checks—simple but powerful.
- This is a great stepping stone toward bigger RPG mechanics.