Source Code

Structure

```
dist
└── index.html

1 directory, 1 file
```

dist/index.html

```
<!doctype html>
<html lang="ko">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,initial-scale=1" />
    <title>2048 (Vanilla JS)</title>
    <style>
      :root {
        --bg: #0f1220;
        --panel: #171a2b;
        --tile: #232744;
        --text: #e9ecff;
        --muted: #a9b0d8;
        --gap: 10px;
        --radius: 14px;
        --cell: 78px;
      }
      * {
        box-sizing: border-box;
      }
      body {
        margin: 0;
        min-height: 100vh;
        display: flex;
        align-items: center;
        justify-content: center;
        background: linear-gradient(180deg, #0b0d18, #111533);
        color: var(--text);
        font-family:
          system-ui,
          -apple-system,
          Segoe UI,
          Roboto,
          sans-serif;
      }
      .wrap {
        width: min(420px, 92vw);
      }
      header {
        display: flex;
        align-items: flex-end;
        justify-content: space-between;
        margin-bottom: 14px;
        gap: 12px;
      }
      .title {
        line-height: 1;
      }
```

```
.title h1 {
  margin: 0;
  font-size: 44px;
  letter-spacing: -1px;
}
.title p {
  margin: 6px 0 0;
  color: var(--muted);
  font-size: 13px;
}
.stats {
  display: flex;
  gap: 10px;
  align-items: flex-end;
}
.stat {
  background: var(--panel);
  padding: 10px 12px;
  border-radius: 12px;
  min-width: 92px;
  text-align: center;
  box-shadow: 0 10px 24px rgba(0, 0, 0, 0.25);
}
.stat .label {
  font-size: 11px;
  color: var(--muted);
  letter-spacing: 0.5px;
}
.stat .value {
  font-size: 20px;
  font-weight: 800;
  margin-top: 4px;
}
.btns {
  display: flex;
  gap: 10px;
  margin: 12px 0 14px;
}
button {
  border: 0;
  cursor: pointer;
  background: #2b3160;
  color: var(--text);
  padding: 10px 12px;
  border-radius: 12px;
  font-weight: 700;
  box-shadow: 0 10px 24px rgba(0, 0, 0, 0.22);
}
button:active {
  transform: translateY(1px);
}
.board {
  position: relative;
  background: var(--panel);
  padding: var(--gap);
  border-radius: var(--radius);
  box-shadow: 0 16px 40px rgba(0, 0, 0, 0.3);
  user-select: none;
  touch-action: none;
}
.grid {
```

```
  display: grid;
  grid-template-columns: repeat(4, var(--cell));
  grid-template-rows: repeat(4, var(--cell));
  gap: var(--gap);
}
.cell {
  background: rgba(255, 255, 255, 0.06);
  border-radius: 12px;
}
.tiles {
  position: absolute;
  inset: var(--gap);
  display: grid;
  grid-template-columns: repeat(4, var(--cell));
  grid-template-rows: repeat(4, var(--cell));
  gap: var(--gap);
  pointer-events: none;
}
.tile {
  display: flex;
  align-items: center;
  justify-content: center;
  border-radius: 12px;
  font-weight: 900;
  font-size: 26px;
  letter-spacing: -0.5px;
  transform: scale(1);
  transition: transform 120ms ease;
}
.tile.pop {
  transform: scale(1.08);
}
.overlay {
  position: absolute;
  inset: 0;
  display: none;
  align-items: center;
  justify-content: center;
  background: rgba(10, 12, 22, 0.72);
  border-radius: var(--radius);
  text-align: center;
  padding: 24px;
}
.overlay.show {
  display: flex;
}
.overlay .card {
  background: rgba(23, 26, 43, 0.92);
  border: 1px solid rgba(255, 255, 255, 0.1);
  border-radius: 16px;
  padding: 18px 16px;
  width: min(320px, 92%);
  box-shadow: 0 16px 44px rgba(0, 0, 0, 0.45);
}
.overlay h2 {
  margin: 0 0 6px;
  font-size: 22px;
}
.overlay p {
  margin: 0 0 14px;
  color: var(--muted);
```

```
      font-size: 13px;
    }
    .actions {
      display: flex;
      gap: 10px;
      justify-content: center;
      margin-bottom: 10px;
    }
    .hint {
      margin-top: 10px;
      color: var(--muted);
      font-size: 12px;
    }
    @media (max-width: 380px) {
      :root {
        --cell: 66px;
      }
      .title h1 {
        font-size: 40px;
      }
    }
  </style>
</head>
<body>
  <div class="wrap">
    <header>
      <div class="title">
        <h1>2048</h1>
        <p>Arrow keys / WASD. Mobile: swipe. Step by Step, ok?</p>
      </div>
      <div class="stats">
        <div class="stat">
          <div class="label">SCORE</div>
          <div class="value" id="score">0</div>
        </div>
        <div class="stat">
          <div class="label">BEST</div>
          <div class="value" id="best">0</div>
        </div>
        <div class="stat">
          <div class="label">BEST TIME</div>
          <div class="value" id="bestTime">–</div>
        </div>
      </div>
    </header>

    <div class="btns">
      <button id="new">New Game</button>
      <button id="undo">Undo</button>
    </div>

    <div class="board" id="board" aria-label="2048 board">
      <div class="grid" aria-hidden="true">
        <div class="cell"></div>
        <div class="cell"></div>
        <div class="cell"></div>
        <div class="cell"></div>
        <div class="cell"></div>
        <div class="cell"></div>
        <div class="cell"></div>
        <div class="cell"></div>
```

```
      <div class="cell"></div>
      <div class="cell"></div>
      <div class="cell"></div>
      <div class="cell"></div>
      <div class="cell"></div>
      <div class="cell"></div>
      <div class="cell"></div>
      <div class="cell"></div>
    </div>
    <div class="tiles" id="tiles"></div>

    <div class="overlay" id="overlay">
      <div class="card">
        <h2 id="overTitle">Game Over</h2>
        <p id="overMsg">No more moves. Try again, my son.</p>
        <div class="actions">
          <button id="again">Play Again</button>
          <button id="dismiss" style="display: none">Continue</button>
        </div>
        <div class="hint">Tip: Don’t spam moves. Keep big tiles in a corner.</div>
      </div>
    </div>
  </div>
</div>

<script>
  (() => {
    const N = 4;
    const scoreEl = document.getElementById('score');
    const bestEl = document.getElementById('best');
    const bestTimeEl = document.getElementById('bestTime');
    const tilesEl = document.getElementById('tiles');
    const overlay = document.getElementById('overlay');
    const overTitle = document.getElementById('overTitle');
    const overMsg = document.getElementById('overMsg');
    const btnDismiss = document.getElementById('dismiss');

    const btnNew = document.getElementById('new');
    const btnUndo = document.getElementById('undo');
    const btnAgain = document.getElementById('again');
    const boardEl = document.getElementById('board');

    const LS_BEST = 'vanilla2048_best';
    const LS_STATE = 'vanilla2048_state';
    const LS_BEST_TIME = 'vanilla2048_best_time';

    let grid, score, best, won, startAt, bestTime, winDismissed;
    let prevState = null; // for undo

    const colors = new Map([
      [0, { bg: 'rgba(255,255,255,.00)', fg: '#e9ecff' }],
      [2, { bg: '#2a2f55', fg: '#e9ecff' }],
      [4, { bg: '#323a6a', fg: '#e9ecff' }],
      [8, { bg: '#3c4aa0', fg: '#ffffff' }],
      [16, { bg: '#3f73b8', fg: '#ffffff' }],
      [32, { bg: '#3aa6b3', fg: '#06232a' }],
      [64, { bg: '#3bd08f', fg: '#062115' }],
      [128, { bg: '#8ad33b', fg: '#1b2406' }],
      [256, { bg: '#d6c93a', fg: '#2a2406' }],
      [512, { bg: '#d99a2f', fg: '#2a1506' }],
      [1024, { bg: '#d06a2b', fg: '#2a0f06' }],
```

```
  [2048, { bg: '#c83b3b', fg: '#fff' }],
]);

function emptyGrid() {
  return Array.from({ length: N }, () => Array(N).fill(0));
}

function cloneState(state) {
  return {
    grid: state.grid.map((r) => r.slice()),
    score: state.score,
    best: state.best,
    won: state.won,
    startAt: state.startAt,
    winDismissed: state.winDismissed,
  };
}

function savePrev() {
  prevState = cloneState({ grid, score, best, won });
}

function setOverlay(show, title = 'Game Over', msg = 'No more moves.', allowClose = false) {
  overlay.classList.toggle('show', show);
  overTitle.textContent = title;
  overMsg.textContent = msg;
  btnDismiss.style.display = allowClose ? 'inline-block' : 'none';
}

function loadBest() {
  const v = Number(localStorage.getItem(LS_BEST) || 0);
  return Number.isFinite(v) ? v : 0;
}

function loadBestTime() {
  const v = Number(localStorage.getItem(LS_BEST_TIME) || NaN);
  return Number.isFinite(v) && v > 0 ? v : null;
}

function formatMs(ms) {
  if (ms == null) return '–';
  const totalSec = Math.round(ms / 1000);
  const m = Math.floor(totalSec / 60);
  const s = totalSec % 60;
  return `${m}:${String(s).padStart(2, '0')}`;
}

function setBest(v) {
  best = Math.max(best, v);
  localStorage.setItem(LS_BEST, String(best));
}

function setBestTime(ms) {
  if (ms == null) return;
  bestTime = bestTime == null ? ms : Math.min(bestTime, ms);
  localStorage.setItem(LS_BEST_TIME, String(bestTime));
}

function serialize() {
  return JSON.stringify({ grid, score, best, won, startAt });
}
```

```
function persist() {
  localStorage.setItem(LS_STATE, serialize());
  localStorage.setItem(LS_BEST, String(best));
}

function restore() {
  try {
    const raw = localStorage.getItem(LS_STATE);
    if (!raw) return false;
    const s = JSON.parse(raw);
    if (!s || !Array.isArray(s.grid)) return false;
    grid = s.grid;
    score = Number(s.score) || 0;
    best = Math.max(loadBest(), Number(s.best) || 0);
    won = !!s.won;
    startAt = Number(s.startAt) || Date.now();
    return true;
  } catch {
    return false;
  }
}

function randEmptyCell() {
  const empties = [];
  for (let r = 0; r < N; r++) {
    for (let c = 0; c < N; c++) {
      if (grid[r][c] === 0) empties.push([r, c]);
    }
  }
  if (empties.length === 0) return null;
  return empties[(Math.random() * empties.length) | 0];
}

function spawn() {
  const cell = randEmptyCell();
  if (!cell) return false;
  const [r, c] = cell;
  grid[r][c] = Math.random() < 0.9 ? 2 : 4;
  return true;
}

function canMove() {
  // any empty
  for (let r = 0; r < N; r++) {
    for (let c = 0; c < N; c++) {
      if (grid[r][c] === 0) return true;
    }
  }
  // adjacent equal
  for (let r = 0; r < N; r++) {
    for (let c = 0; c < N; c++) {
      const v = grid[r][c];
      if (r + 1 < N && grid[r + 1][c] === v) return true;
      if (c + 1 < N && grid[r][c + 1] === v) return true;
    }
  }
  return false;
}

function render(popPos = null) {
```

```
  tilesEl.innerHTML = '';
  for (let r = 0; r < N; r++) {
    for (let c = 0; c < N; c++) {
      const v = grid[r][c];
      if (v === 0) continue;
      const t = document.createElement('div');
      t.className = 'tile';
      t.textContent = v;

      const style = colors.get(v) || { bg: '#7b5bd6', fg: '#fff' };
      t.style.background = style.bg;
      t.style.color = style.fg;
      // font size adjust
      if (v >= 1024) t.style.fontSize = '22px';
  if (v >= 8192) t.style.fontSize = '18px';

      t.style.gridRowStart = r + 1;
      t.style.gridColumnStart = c + 1;

      if (popPos && popPos[0] === r && popPos[1] === c) {
        t.classList.add('pop');
        setTimeout(() => t.classList.remove('pop'), 130);
      }

      tilesEl.appendChild(t);
    }
  }
  scoreEl.textContent = String(score);
  bestEl.textContent = String(best);
  bestTimeEl.textContent = formatMs(bestTime);
}

function slideLine(line) {
  // line: [a,b,c,d]
  const arr = line.filter((v) => v !== 0);
  let gained = 0;
  for (let i = 0; i < arr.length - 1; i++) {
    if (arr[i] !== 0 && arr[i] === arr[i + 1]) {
      arr[i] *= 2;
      gained += arr[i];
      arr[i + 1] = 0;
      if (arr[i] === 2048) won = true;
    }
  }
  const merged = arr.filter((v) => v !== 0);
  while (merged.length < N) merged.push(0);
  return { line: merged, gained };
}

function move(dir) {
  // dir: 'L','R','U','D'
  setOverlay(false);

  const before = grid.map((r) => r.slice());
  const beforeScore = score;
  const beforeWon = won;

  let moved = false;
  let pop = null;

  if (dir === 'L' || dir === 'R') {
```

```
  for (let r = 0; r < N; r++) {
    const row = grid[r].slice();
    const work = dir === 'R' ? row.reverse() : row;
    const { line, gained } = slideLine(work);
    const out = dir === 'R' ? line.reverse() : line;
    if (out.some((v, i) => v !== grid[r][i])) moved = true;
    grid[r] = out;
    score += gained;
  }
} else {
  for (let c = 0; c < N; c++) {
    const col = [];
    for (let r = 0; r < N; r++) col.push(grid[r][c]);
    const work = dir === 'D' ? col.reverse() : col;
    const { line, gained } = slideLine(work);
    const out = dir === 'D' ? line.reverse() : line;
    for (let r = 0; r < N; r++) {
      if (grid[r][c] !== out[r]) moved = true;
      grid[r][c] = out[r];
    }
    score += gained;
  }
}

if (!moved) {
  // no-op
  grid = before;
  score = beforeScore;
  won = beforeWon;
  return;
}

savePrev(); // after confirming move? keep before state for undo
// Actually we want undo to return to "before"
prevState = {
  grid: before,
  score: beforeScore,
  best,
  won: beforeWon,
  startAt,
  winDismissed,
};

if (score > best) setBest(score);

// spawn a new tile and pop it
const spawned = randEmptyCell();
if (spawned) {
  const [r, c] = spawned;
  grid[r][c] = Math.random() < 0.9 ? 2 : 4;
  pop = [r, c];
}

const newlyWon = !beforeWon && won;
let achievedTime = null;
if (newlyWon) {
  achievedTime = Date.now() - startAt;
  setBestTime(achievedTime);
  winDismissed = false;
}
```

```
  persist();
  render(pop);

  if (won) {
    const clearTimeMsg = newlyWon
      ? `Time: ${formatMs(achievedTime)} · Best: ${formatMs(bestTime)}`
      : `Best: ${formatMs(bestTime)}`;
    if (!winDismissed) {
      setOverlay(true, 'You Win', `2048 reached. ${clearTimeMsg} Keep going if you want.`, true);
    }
  } else if (!canMove()) {
    setOverlay(true, 'Game Over', 'No more moves. Reset and do it better, my son.');
  }
}

function newGame() {
  grid = emptyGrid();
  score = 0;
  won = false;
  best = loadBest();
  bestTime = loadBestTime();
  startAt = Date.now();
  winDismissed = false;
  prevState = null;
  spawn();
  spawn();
  persist();
  setOverlay(false);
  render();
}

function undo() {
  if (!prevState) return;
  grid = prevState.grid.map((r) => r.slice());
  score = prevState.score;
  won = prevState.won;
  best = loadBest();
  bestTime = loadBestTime();
  startAt = prevState.startAt || Date.now();
  winDismissed = prevState.winDismissed || false;
  prevState = null;
  persist();
  setOverlay(false);
  render();
}

function onKey(e) {
  const k = e.key.toLowerCase();
  if (['arrowleft', 'a'].includes(k)) {
    e.preventDefault();
    move('L');
  } else if (['arrowright', 'd'].includes(k)) {
    e.preventDefault();
    move('R');
  } else if (['arrowup', 'w'].includes(k)) {
    e.preventDefault();
    move('U');
  } else if (['arrowdown', 's'].includes(k)) {
    e.preventDefault();
    move('D');
  }
```

```
        }

        // Touch swipe
        let sx = 0,
          sy = 0,
          active = false;
        function onTouchStart(ev) {
          const t = ev.changedTouches[0];
          sx = t.clientX;
          sy = t.clientY;
          active = true;
        }
        function onTouchEnd(ev) {
          if (!active) return;
          const t = ev.changedTouches[0];
          const dx = t.clientX - sx;
          const dy = t.clientY - sy;
          active = false;

          const ax = Math.abs(dx),
            ay = Math.abs(dy);
          if (Math.max(ax, ay) < 18) return; // tiny swipe ignore
          if (ax > ay) {
            move(dx > 0 ? 'R' : 'L');
          } else {
            move(dy > 0 ? 'D' : 'U');
          }
        }

        btnNew.addEventListener('click', newGame);
        btnUndo.addEventListener('click', undo);
        btnAgain.addEventListener('click', newGame);
        btnDismiss.addEventListener('click', () => {
          winDismissed = true;
          setOverlay(false);
        });

        window.addEventListener('keydown', onKey, { passive: false });
        boardEl.addEventListener('touchstart', onTouchStart, { passive: true });
        boardEl.addEventListener('touchend', onTouchEnd, { passive: true });

        // init
        best = loadBest();
        bestTime = loadBestTime();
        winDismissed = false;
        if (!restore()) newGame();
        else {
          render();
          setOverlay(false);
        }
      })();
    </script>
  </body>
</html>
```