Source Code

Structure

```
dist
├── game.js
├── index.html
└── style.css

1 directory, 3 files
```

dist/game.js

```
(() => {
  const $ = (id) => document.getElementById(id);
  const N = 4;
  const scoreEl = $('score');
  const bestEl = $('best');
  const bestTimeEl = $('bestTime');
  const tilesEl = $('tiles');
  const overlay = $('overlay');
  const overTitle = $('overTitle');
  const overMsg = $('overMsg');
  const btnDismiss = $('dismiss');
  const btnNew = $('new');
  const btnUndo = $('undo');
  const btnAgain = $('again');
  const boardEl = $('board');
  document.querySelector('.grid').innerHTML = '<div class="cell"></div>'.repeat(N * N);

  const LS = {
    best: 'vanilla2048_best',
    state: 'vanilla2048_state',
    bestTime: 'vanilla2048_best_time',
  };

  const colors = {
    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' },
    4096: { bg: '#c44d7c', fg: '#fff' },
    8192: { bg: '#b154a8', fg: '#f6f0ff' },
    16384: { bg: '#8a63d3', fg: '#f5f2ff' },
    32768: { bg: '#6c7af2', fg: '#0a1230' },
    65536: { bg: '#47a5f5', fg: '#041a2b' },
    131072: { bg: '#29c7df', fg: '#04181c' },
  };

  let grid, score, best, won, startAt, bestTime, winDismissed, prevState;
```

```
const emptyGrid = () => Array.from({ length: N }, () => Array(N).fill(0));
const loadBest = () => {
  const v = +localStorage.getItem(LS.best);
  return Number.isFinite(v) ? v : 0;
};
const loadBestTime = () => {
  const v = +localStorage.getItem(LS.bestTime);
  return Number.isFinite(v) && v > 0 ? v : null;
};
const formatMs = (ms) => {
  if (ms == null) return '–';
  const sec = Math.round(ms / 1e3);
  return `${Math.floor(sec / 60)}:${String(sec % 60).padStart(2, '0')}`;
};
const 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';
};
const persist = () => {
  localStorage.setItem(
    LS.state,
    JSON.stringify({ grid, score, best, won, startAt, winDismissed, bestTime })
  );
  localStorage.setItem(LS.best, best);
  if (bestTime != null) localStorage.setItem(LS.bestTime, bestTime);
};
const restore = () => {
  try {
    const raw = localStorage.getItem(LS.state);
    if (!raw) return false;
    const s = JSON.parse(raw);
    if (!s?.grid) return false;
    grid = s.grid;
    score = +s.score || 0;
    best = Math.max(loadBest(), +s.best || 0);
    won = !!s.won;
    startAt = +s.startAt || Date.now();
    winDismissed = !!s.winDismissed;
    bestTime = loadBestTime();
    return true;
  } catch {
    return false;
  }
};
const emptyCell = () => {
  const free = [];
  for (let r = 0; r < N; r++) for (let c = 0; c < N; c++) if (!grid[r][c]) free.push([r, c]);
  return free.length ? free[(Math.random() * free.length) | 0] : null;
};
const spawn = () => {
  const pos = emptyCell();
  if (!pos) return null;
  const [r, c] = pos;
  grid[r][c] = Math.random() < 0.9 ? 2 : 4;
  return pos;
};
const canMove = () => {
  for (let r = 0; r < N; r++) for (let c = 0; c < N; c++) if (!grid[r][c]) return true;
  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) || (c + 1 < N && grid[r][c + 1] === v)) return true;
    }
  return false;
};
const render = (pop) => {
  tilesEl.innerHTML = '';
  for (let r = 0; r < N; r++)
    for (let c = 0; c < N; c++) {
      const v = grid[r][c];
      if (!v) continue;
      const t = document.createElement('div');
      const sty = colors[v] || { bg: '#7b5bd6', fg: '#fff' };
      t.className = 'tile';
      t.textContent = v;
      t.style.background = sty.bg;
      t.style.color = sty.fg;
      t.style.fontSize = v >= 8192 ? '18px' : v >= 1024 ? '22px' : '26px';
      t.style.gridRowStart = r + 1;
      t.style.gridColumnStart = c + 1;
      if (pop && pop[0] === r && pop[1] === c) {
        t.classList.add('pop');
        setTimeout(() => t.classList.remove('pop'), 130);
      }
      tilesEl.appendChild(t);
    }
  scoreEl.textContent = score;
  bestEl.textContent = best;
  bestTimeEl.textContent = formatMs(bestTime);
};
const slideLine = (line) => {
  const arr = line.filter(Boolean);
  let gained = 0;
  for (let i = 0; i < arr.length - 1; i++)
    if (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(Boolean);
  while (merged.length < N) merged.push(0);
  return { line: merged, gained };
};
const move = (dir) => {
  setOverlay(false);
  const before = grid.map((r) => r.slice());
  const prevScore = score;
  const prevWon = 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) {
    grid = before;
    score = prevScore;
    won = prevWon;
    return;
  }
  prevState = {
    grid: before,
    score: prevScore,
    best,
    won: prevWon,
    startAt,
    winDismissed,
    bestTime,
  };
  if (score > best) {
    best = score;
    localStorage.setItem(LS.best, best);
  }
  pop = spawn();
  const justWon = !prevWon && won;
  const clearMs = justWon ? Date.now() - startAt : null;
  if (justWon) {
    bestTime = bestTime == null ? clearMs : Math.min(bestTime, clearMs);
    localStorage.setItem(LS.bestTime, bestTime);
    winDismissed = false;
  }
  persist();
  render(pop);
  if (won && !winDismissed) {
    const msg = justWon ? `Time: ${formatMs(clearMs)} · Best: ${formatMs(bestTime)}` : `Best: ${formatMs(bestTime)}`;
    setOverlay(true, 'You Win', `2048 reached. ${msg} Keep going if you want.`, true);
  } else if (!canMove()) {
    setOverlay(true, 'Game Over', 'No more moves. Reset and do it better, my son.');
  }
};
const newGame = () => {
  grid = emptyGrid();
  score = 0;
  won = false;
  best = loadBest();
  bestTime = loadBestTime();
  startAt = Date.now();
  winDismissed = false;
  prevState = null;
  spawn();
```

```
  spawn();
  persist();
  setOverlay(false);
  render();
};
const 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();
};
const onKey = (e) => {
  const k = e.key.toLowerCase();
  if (['arrowleft', 'a'].includes(k)) return e.preventDefault(), move('L');
  if (['arrowright', 'd'].includes(k)) return e.preventDefault(), move('R');
  if (['arrowup', 'w'].includes(k)) return e.preventDefault(), move('U');
  if (['arrowdown', 's'].includes(k)) return e.preventDefault(), move('D');
};
let touchX = 0,
  touchY = 0,
  touching = false;
const onTouchStart = (ev) => {
  const t = ev.changedTouches[0];
  touchX = t.clientX;
  touchY = t.clientY;
  touching = true;
};
const onTouchEnd = (ev) => {
  if (!touching) return;
  const t = ev.changedTouches[0];
  const dx = t.clientX - touchX;
  const dy = t.clientY - touchY;
  touching = false;
  if (Math.max(Math.abs(dx), Math.abs(dy)) < 18) return;
  move(Math.abs(dx) > Math.abs(dy) ? (dx > 0 ? 'R' : 'L') : dy > 0 ? 'D' : 'U');
};

btnNew.onclick = newGame;
btnUndo.onclick = undo;
btnAgain.onclick = newGame;
btnDismiss.onclick = () => ((winDismissed = true), setOverlay(false));
window.addEventListener('keydown', onKey, { passive: false });
boardEl.addEventListener('touchstart', onTouchStart, { passive: true });
boardEl.addEventListener('touchend', onTouchEnd, { passive: true });

best = loadBest();
bestTime = loadBestTime();
winDismissed = false;
if (!restore()) newGame();
else {
  render();
  setOverlay(false);
}
```

```
})();
```

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</title>
    <link rel="stylesheet" href="style.css" />
  </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>
        <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 defer src="game.js"></script>
  </body>
</html>
```

dist/style.css

```css
: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;
}
```

```css
.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 rgb(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 rgb(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 rgb(0 0 0 / 30%);
  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: rgb(255 255 255 / 6%);
  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: rgb(10 12 22 / 72%);
  border-radius: var(--radius);
  text-align: center;
  padding: 24px;
}

.overlay.show {
  display: flex;
}

.overlay .card {
  background: rgb(23 26 43 / 92%);
  border: 1px solid rgb(255 255 255 / 10%);
  border-radius: 16px;
  padding: 18px 16px;
  width: min(320px, 92%);
  box-shadow: 0 16px 44px rgb(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 (width <= 380px) {
  :root {
    --cell: 66px;
  }

  .title h1 {
    font-size: 40px;
  }
}
```