Source Code

Structure

```
dist
├── bundle.js
├── index.html
└── styles.css

1 directory, 3 files
```

dist/bundle.js

```
(() => {
  var __getOwnPropNames = Object.getOwnPropertyNames;
  var __esm = (fn, res) => function __init() {
    return fn && (res = (0, fn[__getOwnPropNames(fn)[0]])(fn = 0)), res;
  };
  var __commonJS = (cb, mod) => function __require() {
    return mod || (0, cb[__getOwnPropNames(cb)[0]])((mod = { exports: {} }).exports, mod), mod.exports;
  };

  // src/engine/js/Cell.js
  var Cell;
  var init_Cell = __esm({
    "src/engine/js/Cell.js"() {
      Cell = class {
        constructor() {
          this.isMine = false;
          this.isOpen = false;
          this.isFlagged = false;
          this.adjacentMines = 0;
        }
      };
    }
  });

  // src/engine/js/Board.js
  var Board;
  var init_Board = __esm({
    "src/engine/js/Board.js"() {
      init_Cell();
      Board = class _Board {
        constructor(width, height, mineCount) {
          this.width = width;
          this.height = height;
          this.mineCount = mineCount;
          this.cells = _Board.#createEmptyBoard(width, height);
          _Board.#placeMines(this.cells, width, height, mineCount);
          _Board.#calculateAdjacents(this.cells, width, height);
          this.openedCellsCount = 0;
          this.state = "IN_PROGRESS";
        }
        static #createEmptyBoard(width, height) {
          return Array.from({ length: height }, () => Array.from({ length: width }, () => new Cell()));
        }
        static #placeMines(cells, width, height, mineCount) {
          let placedMines = 0;
          while (placedMines < mineCount) {
```

```
    const x = Math.floor(Math.random() * width);
    const y = Math.floor(Math.random() * height);
    if (!cells[y][x].isMine) {
      cells[y][x].isMine = true;
      placedMines++;
    }
  }
}
static #calculateAdjacents(cells, width, height) {
  const directions = _Board.#DIRECTIONS;
  for (let y = 0; y < height; y++) {
    for (let x = 0; x < width; x++) {
      if (cells[y][x].isMine) continue;
      let adjMines = 0;
      for (const [dx, dy] of directions) {
        const nx = x + dx;
        const ny = y + dy;
        if (nx >= 0 && nx < width && ny >= 0 && ny < height) {
          if (cells[ny][nx].isMine) {
            adjMines++;
          }
        }
      }
      cells[y][x].adjacentMines = adjMines;
    }
  }
}
static #DIRECTIONS = [
  [-1, -1],
  [-1, 0],
  [-1, 1],
  [0, -1],
  [0, 1],
  [1, -1],
  [1, 0],
  [1, 1]
];
openCell(x, y) {
  if (this.outOfBounds(x, y) || this.state !== "IN_PROGRESS") return;
  const cell = this.cells[y][x];
  if (cell.isOpen || cell.isFlagged) return;
  if (cell.isMine) {
    this.state = "LOST";
    this.openAllMines();
    return;
  }
  this.BFS_reveal_from(x, y);
  if (this.openedCellsCount === this.width * this.height - this.mineCount) {
    this.state = "WON";
    return;
  }
}
outOfBounds(x, y) {
  return x < 0 || x >= this.width || y < 0 || y >= this.height;
}
BFS_reveal_from(x, y) {
  const queue = [];
  let head = 0;
  const visited = /* @__PURE__ */ new Set();
  queue.push([x, y]);
  while (head < queue.length) {
```

```
          const [cx, cy] = queue[head++];
          if (this.outOfBounds(cx, cy)) continue;
          if (visited.has(cy * this.width + cx)) continue;
          visited.add(cy * this.width + cx);
          const cell = this.cells[cy][cx];
          if (cell.isOpen || cell.isFlagged) continue;
          if (cell.isMine) continue;
          cell.isOpen = true;
          this.openedCellsCount++;
          if (cell.adjacentMines > 0) continue;
          for (const [dx, dy] of _Board.#DIRECTIONS) {
            const nx = cx + dx;
            const ny = cy + dy;
            if (this.outOfBounds(nx, ny)) continue;
            const neighbor = this.cells[ny][nx];
            if (neighbor.isOpen || neighbor.isFlagged) continue;
            queue.push([nx, ny]);
          }
        }
      }
      openAllMines() {
        for (let y = 0; y < this.height; y++) {
          for (let x = 0; x < this.width; x++) {
            const cell = this.cells[y][x];
            cell.isOpen = true;
          }
        }
      }
      toggleFlag(x, y) {
        if (this.outOfBounds(x, y) || this.state !== "IN_PROGRESS") return;
        const cell = this.cells[y][x];
        if (cell.isOpen) return;
        cell.isFlagged = !cell.isFlagged;
      }
      getCell(x, y) {
        if (this.outOfBounds(x, y)) return null;
        return this.cells[y][x];
      }
      getState() {
        return this.state;
      }
    };
  }
});

// src/ui/script.js
var require_script = __commonJS({
  "src/ui/script.js"() {
    init_Board();
    var board;
    var boardElement = document.getElementById("board");
    var statusElement = document.getElementById("status");
    var resetButton = document.getElementById("reset");
    var widthRange = document.getElementById("width-range");
    var heightRange = document.getElementById("height-range");
    var minesRange = document.getElementById("mines-range");
    var widthValue = document.getElementById("width-value");
    var heightValue = document.getElementById("height-value");
    var minesValue = document.getElementById("mines-value");
    init();
    function init() {
```

```
  const width = Number(widthRange.value);
  const height = Number(heightRange.value);
  const mines = Math.min(Number(minesRange.value), width * height);
  board = new Board(width, height, mines);
  boardElement.style.gridTemplateColumns = `repeat(${width}, 32px)`;
  boardElement.innerHTML = "";
  for (let y = 0; y < height; y++) {
    for (let x = 0; x < width; x++) {
      const cellElement = document.createElement("div");
      cellElement.classList.add("cell");
      cellElement.dataset.x = x;
      cellElement.dataset.y = y;
      boardElement.appendChild(cellElement);
    }
  }
  render();
}
function syncMinesMax() {
  const width = Number(widthRange.value);
  const height = Number(heightRange.value);
  const maxMines = width * height;
  minesRange.max = String(maxMines);
  if (Number(minesRange.value) > maxMines) {
    minesRange.value = String(maxMines);
  }
  minesValue.textContent = minesRange.value;
}
function updateSettingValue() {
  widthValue.textContent = widthRange.value;
  heightValue.textContent = heightRange.value;
  minesValue.textContent = minesRange.value;
}
boardElement.addEventListener("click", (event) => {
  const cellElement = event.target.closest(".cell");
  if (!cellElement) return;
  const x = Number(cellElement.dataset.x);
  const y = Number(cellElement.dataset.y);
  board.openCell(x, y);
  render();
});
boardElement.addEventListener("contextmenu", (event) => {
  event.preventDefault();
  const cellElement = event.target.closest(".cell");
  if (!cellElement) return;
  const x = Number(cellElement.dataset.x);
  const y = Number(cellElement.dataset.y);
  board.toggleFlag(x, y);
  render();
});
resetButton.addEventListener("click", () => {
  init();
});
widthRange.addEventListener("input", () => {
  updateSettingValue();
  syncMinesMax();
});
heightRange.addEventListener("input", () => {
  updateSettingValue();
  syncMinesMax();
});
minesRange.addEventListener("input", () => {
```

```
        updateSettingValue();
      });
      function render() {
        const cells = boardElement.children;
        for (const cellElement of cells) {
          const x = Number(cellElement.dataset.x);
          const y = Number(cellElement.dataset.y);
          const cell = board.getCell(x, y);
          cellElement.className = "cell";
          cellElement.textContent = "";
          if (cell.isOpen) {
            cellElement.classList.add("open");
            if (cell.isMine) {
              cellElement.classList.add("mine");
              cellElement.textContent = "\u{1F4A3}";
            } else if (cell.adjacentMines > 0) {
              cellElement.textContent = cell.adjacentMines;
            }
          } else if (cell.isFlagged) {
            cellElement.classList.add("flagged");
            cellElement.textContent = "\u{1F6A9}";
          }
        }
        statusElement.textContent = board.getState();
        if (board.getState() !== "IN_PROGRESS") {
          alert(`You ${board.getState() === "WON" ? "won" : "lost"}!`);
        }
      }
      updateSettingValue();
      syncMinesMax();
    }
  });
  require_script();
})();
```

dist/index.html

```
<!doctype html>
<html lang="ko">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Minesweeper</title>
    <link rel="stylesheet" href="./styles.css" />
  </head>
  <body>
    <div id="hud">
      <span id="status">IN_PROGRESS</span>
      <button id="reset">Reset</button>
    </div>

    <div id="settings" aria-label="Board settings">
      <label class="setting">
        <span class="setting-label">Width</span>
        <input id="width-range" type="range" min="4" max="50" value="16" />
        <span id="width-value" class="setting-value">16</span>
      </label>
      <label class="setting">
        <span class="setting-label">Height</span>
        <input id="height-range" type="range" min="4" max="50" value="16" />
```

```
        <span id="height-value" class="setting-value">16</span>
      </label>
      <label class="setting">
        <span class="setting-label">Mines</span>
        <input id="mines-range" type="range" min="1" max="256" value="20" />
        <span id="mines-value" class="setting-value">20</span>
      </label>
    </div>

    <div id="board" aria-label="Minesweeper board"></div>

    <script src="./bundle.js"></script>
  </body>
</html>
```

dist/styles.css

```
body {
  font-family: sans-serif;
  background: #111;
  color: #eee;
  display: flex;
  flex-direction: column;
  align-items: center;
}

#hud {
  margin: 12px;
  display: flex;
  gap: 12px;
}

#settings {
  display: flex;
  flex-direction: column;
  gap: 8px;
  padding: 12px;
  border: 1px solid #333;
  border-radius: 8px;
  margin-bottom: 12px;
  min-width: 280px;
}

.setting {
  display: grid;
  grid-template-columns: 72px 1fr 48px;
  align-items: center;
  gap: 12px;
}

.setting-label {
  font-size: 0.9rem;
  color: #bbb;
}

.setting-value {
  text-align: right;
  font-variant-numeric: tabular-nums;
}
```

```css
#board {
  display: grid;
  gap: 2px;
  user-select: none;
}

.cell {
  width: 32px;
  height: 32px;
  background: #444;
  display: flex;
  align-items: center;
  justify-content: center;
  cursor: pointer;
  font-weight: bold;
}

.cell.open {
  background: #222;
  cursor: default;
}

.cell.flagged {
  background: orange;
}

.cell.mine {
  background: darkred;
}
```