JavaScript Basics for Minesweeper Game

This document provides an overview of JavaScript concepts and basics used in the Minesweeper game implementation. We will cover JavaScript syntax, variables, control flow, functions, DOM manipulation, and event handling.

# Detailed Breakdown of Key JavaScript Functions

## 1. inGame()

This function initializes the game whenever it starts or resets, clearing previous game data and setting up the grid.

Key Steps:

- Clear Grid: Removes any existing cells from the HTML document to avoid overlap.

- Reset Data Structures: Prepares for fresh cell and mine data by resetting arrays.

- First Click Protection: Ensures the first clicked cell is safe.

- Calls setupGrid, placeMines, and renderGrid to configure, place mines, and display the grid.

## 2. setupGrid()

Configures the grid dimensions and initializes each cell with default properties based on the selected difficulty level.

Key Steps:

- Adjust Grid Size and Mine Count: Sets grid size and mine count according to the difficulty setting.

- Style CSS Grid: Dynamically sets columns to match gridSize for a uniform layout.

- Initialize Cell Properties: Each cell is initialized as an object with mine, revealed, and adjacentMines properties.

## 3. placeMines(excludeRow, excludeCol)

Places mines in random cells, ensuring the first-clicked cell is safe and increments adjacent mine counts for neighboring cells.

Key Steps:

- Loop Until Mines Are Placed: Randomly places mines until mineCount is reached.

- Avoid First-Click Cell: Ensures the clicked cell and adjacent cells are mine-free.

- Increment Adjacent Mine Counts: Updates surrounding cells' adjacentMines counts to indicate mine proximity.

## 4. renderGrid()

Creates and displays each cell in the grid as HTML elements, setting up event listeners for player interactions.

Key Steps:

- Loop Over Grid Size: Iterates to create cell elements, adding row and column attributes.

- Add Event Listeners: Adds click (to reveal cells) and right-click (to toggle flags) functionality.

- Append Cells to DOM: Adds cells to the grid container, making them visible to players.

## 5. handleCellClick(row, col)

Manages the first-click protection logic and reveals the clicked cell.

Key Steps:

- First Click Protection: Ensures the first-clicked cell is safe by removing any mine and replacing it elsewhere.

- Calls revealCell(): Once safe, it reveals the clicked cell to start gameplay.

## 6. revealCell(row, col)

Reveals the clicked cell, displaying contents or triggering game over if it contains a mine.

Key Steps:

- Early Exit: Exits if the cell is already revealed or flagged.

- Game Over: If the cell contains a mine, shows the mine, displays 'Game Over', and reveals all mines.

- Display Numbers or Recursion: If no mine, shows adjacent mine count or calls revealAdjacentCells for empty cells.

## 7. revealAdjacentCells(row, col)

Recursively reveals all safe neighboring cells around the clicked cell if no adjacent mines are nearby.

Key Steps:

- Loops Over Neighbors: Checks all adjacent cells for bounds and whether they're safe to reveal.

- Calls revealCell Recursively: Reveals safe neighbors, clearing connected empty spaces.

# Updated Function Explanations

## Restarting the Game (`restartGame()`)

The restart function is triggered when the player clicks the restart button or changes the difficulty level.

### Why This Call Is Made:

• `inGame()`: This function resets the entire game state, including the grid, mine placement, and player flags, ensuring a fresh start.

### Key Actions:

• Clears the previous grid from the DOM to avoid overlap or conflicts.  
• Resets all relevant variables, such as `cells`, `minePositions`, and `firstClick`.  
• Dynamically updates the grid and mine configuration based on the current difficulty level.

## Changing Grid Size

The grid size is determined by the selected difficulty level (`easy`, `medium`, or `hard`). This dynamically adjusts the game's complexity.

### Why This Change Is Necessary:

• Modifying the grid size ensures that players experience varied levels of challenge, catering to different skill levels.  
• Each difficulty level corresponds to specific settings for grid size and the number of mines (`8x8` for easy, `12x12` for medium, `16x16` for hard).

### Key Actions:

• Updates global variables (`gridSize` and `mineCount`) to reflect the chosen difficulty.  
• Adjusts the CSS grid layout (`gridTemplateColumns`) dynamically to match the new grid size.  
• Reinitializes cell properties and prepares the game grid for the updated settings.

# Existing Functions

## inGame()

This function initializes or resets the game by setting up the grid, placing mines, and preparing the DOM for gameplay.

### Why This Call Is Made:

• `setupGrid()`: Configures the grid's structure and initializes all cells with default properties.  
• `placeMines()`: Strategically places mines while adhering to the first-click safety rules.  
• `renderGrid()`: Prepares the DOM by displaying the grid and setting up event listeners for user interactions.

### Key Actions:

• Resets global variables like `cells`, `minePositions`, and `firstClick` to their initial states.  
• Clears any previously rendered grid from the DOM to avoid overlap or artifacts.

## setupGrid()

Sets up the game grid dimensions and initializes properties for each cell based on the selected difficulty.

### Why Loops Are Used:

• A nested loop iterates through each row and column to create and initialize a 2D array of cell objects.  
• This ensures every cell in the grid has default properties like `mine`, `revealed`, and `adjacentMines` set up correctly.

### Key Actions:

• Updates CSS grid styles dynamically to match the grid size.  
• Resets the `cells` data structure, ensuring each cell is ready for a new game.

## placeMines(excludeRow, excludeCol)

Randomly places mines on the grid while avoiding the first-click cell and its neighbors.

### Why Loops Are Used:

• A `while` loop ensures that the required number of mines is placed by repeating until `mineCount` is reached.  
• Nested loops increment the `adjacentMines` count for all neighboring cells around each mine.

### Key Actions:

• Ensures the clicked cell and its immediate neighbors are mine-free, maintaining game fairness.  
• Updates the `adjacentMines` property for each neighboring cell to reflect mine proximity.

## renderGrid()

Generates HTML elements for each cell in the grid and appends them to the DOM for user interaction.

### Why Loops Are Used:

• A nested loop iterates through all cells to create corresponding HTML elements.  
• Event listeners for `click` and `contextmenu` (right-click) are added to each cell for user interactions.

### Key Actions:

• Dynamically creates cell elements and sets data attributes for row and column indices.  
• Adds the generated cells to the grid container in the DOM.