Lab02 – MineSweeper

In this lab you will use javascript “classes”, 2D arrays, and recursion to re-create the ol' Windows Favorite : MineSweeper. We shall create a minimalist version of the game, but use its functionality as the basis of our exercise.

Program Specification

Your lab will consist of 3 files, the HMTL, CSS and javascript. The HTML shall consist of the skeleton required to frame our game grid div. The game grid div shall be empty and populated from javascript. There are many ways to tackle this, but to exercise the specific elements required the following specification shall be used.

A **pseudo class called Cell**, containing the following elements :

row, **col :** represent the row, column this cell holds and maps to the grid buttons

display : represents the “exposed” state of the button/cell – false for all at game start, and true for cells/buttons that have been exposed and are displayed

mine : is this cell a MINE

count : the count of MINEs in the cell adjacent to this cell (all 8)

It shall contain the following methods : (using prototype specification)

Show() : using the row, col values, construct the ID required to access the button mapped to this cell. If the display member is false, leave the cell/button blank. Otherwise, set the background color to white and set the cell/button to either blank(count = 0), the adjacency count, or 'B' if it is a mine. To simplify testing for win, return 1 if the cell is not currently displayed, and 0 otherwise. This value can be used by ShowGrid() to determine if all cells have been exposed.

Bind() : this helper function binds the cell object's mapped button click event to a handler defined within this function. We can leverage variable capture to get the click handler “know” what button/cell has been clicked. You can do this by creating a local variable for both row and col, assigned the cell's row, col member values → this will allow these variables to be used within the anonymous event handler whereby their respective row, col values are “captured” and retain their value within the handler. You will also need to retrieve the button element

from its constructed row, col ID string.

The click handler code must handle 2 conditions. Regular clicking will be assumed to be a "guess-expose" and will need to be checked against a MINE or if not, will invoke our recursive Check() method to expose valid cells. Obviously you lose if you click a MINE. We will use SHIFT-CLICK to allow the user to safely expose a MINE. If the user SHIFT-CLICK'd and it IS a MINE, set display to true (he doesn't lose), if it is NOT a MINE, bad guess, you lose. Upon completion of these operations, be sure to invoke ShowGrid() to update the game.

Your main javascript will contain :

max_row, max_col : sized for game play, defines the 2D array size and boundaries

num_mines : Mines created for each game

an **array (2d)** of these cells

NewGame() : Create the 2D array of cells, designate the desired number of random mines, and then pre-populate the adjacency count values for each cell.

NewGrid() : Create the HTML string to build the table of buttons derived from the cell array, ensuring the ID values are constructed to provide a easy translation between cell coordinate and button ID, upon completion, assign your game grid div this content.

ShowGrid() : Iterate through your cell array invoking Show(), also tracking the number of buttons exposed to determine if a win condition has occurred.

BindGrid() : This special function is used once at game creation to iterate through the cell array and invoke Bind() to allow event binding for each button.

RandomCell() : This utility function will return the cell object randomly chosen from within the 2D array.

CountClose() : This utility function will take a row, col coordinate and count the mines adjacent to it, returning the value. This is used at game start to pre-populate the cell adjacency values.

Check() : This is the workhorse recursive function. Given a row, col coordinate, it shall recursively expose cells going up/down/left/right unless the coordinate is out of bounds OR the cell is already exposed. Only recurse if the adjacency count value is 0. This is very similar to floodfill, but instead of changing the color, you expose (display → true) the cell, ceasing if it is already exposed or blocked (by count > 0).

Minimal Requirement :

A game is started when the page is loaded. The Player is notified of a win condition or lose condition in a status element (but further game play need not be disabled), no extraneous UI is necessary.

This base problem is worth [70] marks.

Enhancements :

Include UI elements to allow :
Game Start / Restart / Play Again, etc
User selection of a level or row/col maximums: ie Easy/Medium/Hard OR Max Row, Col, #Mines
Image for Mine button when exposed
Proper Game Play functionality, ie. On win/lose further input is discarded, until new game.

This enhancement problem is worth [30] marks.

Programming Assumptions

You must ensure that various grid sizes work, you will be signed off on at least 2 sizes.

Programming Requirements / Glossary

1. **NJE** – No javascript errors may be logged during gameplay

Program Signoff

Does it work ?

Hints – That troublesome stuff...

Be sure to tackle this incrementally. You should build your table of buttons, ensuring that the ID's constructed are correct. Build your 2D array with default cell objects, complete ShowGrid() to see that it can display correctly, basically what you already know – build, test, repeat.