Sprint 1 Challenge

Mine Sweeper

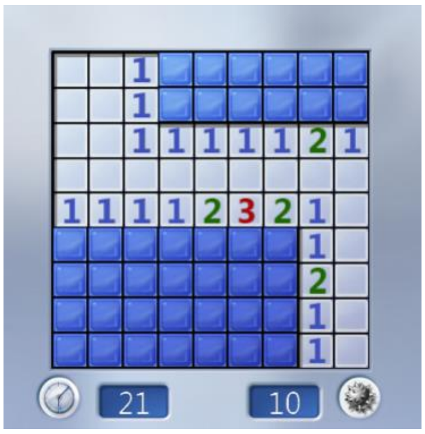
Blow your Mind

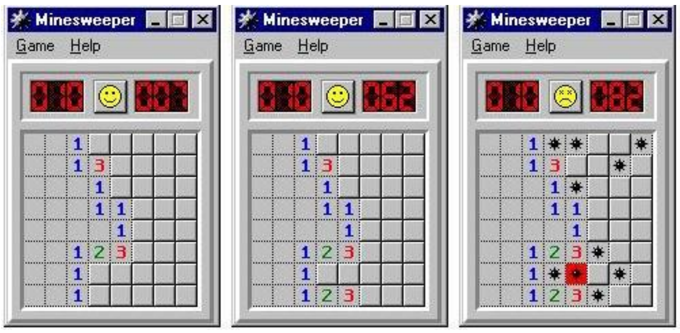
Preview

Your challenge is to create the Minesweeper game, and it's not an easy one. Let's practice some breaths.

Good.

Play the game a little bit and relax





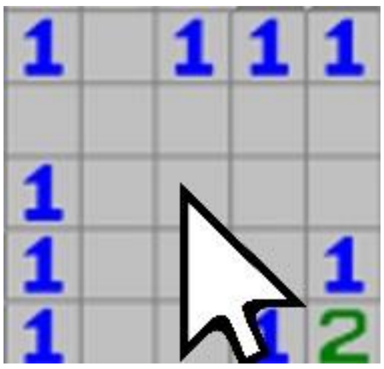
It’s a good thing we studied about Matrixes. Isn’t it?

Features:

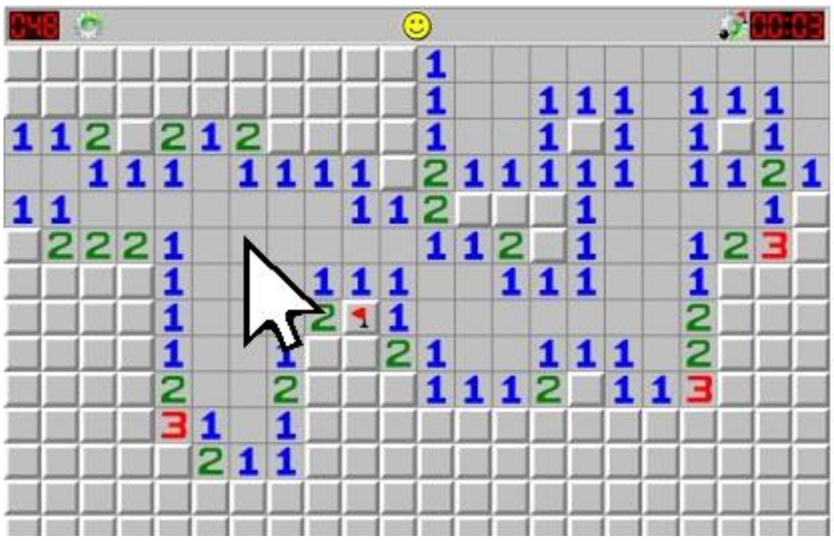
- Minesweeper functionality based on the reference game
- Show a timer that starts on first click (right / left) and stops when game is over.
- Right click to flag/unflag a suspected cell (you cannot reveal a flagged cell)
- When clicking a mine, all mines should be revealed
- game ends when:
    - user clicked a mine
    - all the mines are flagged and all the other cells are shown
- Support 3 levels of the game
    - Beginner (4*4 with 2 MINES)
    - Medium (6 * 6 with 5 MINES)
    - Expert (8 * 8 with 15 MINES)
- If you have the time, take freedom with the design and try giving it a nice shape.

About Expanding

Expanding a cell to 2 levels:



Full expand like in the real game (BONUS):



Development - Tips and Guidelines

As you know, there is usually more than one way to approach a challenge.

But as a guideline, we suggest having the following functions (it is ok to have more functions as needed).

`initGame()`	This is called when page loads
`buildBoard()`	Builds the board by setting mines at random locations, and then calling the setMinesNegsCount() Then return the created board
`setMinesNegsCount(board)`	Sets mines-count to neighbours
`renderBoard``(board)`	Print the board as a <table> to the page
`cellClicked(elCell, i, j)`	Called when a cell (td) is clicked
`cellMarked``(elCell)`	Called on right click to mark a cell as suspected to have a mine
`checkGameOver()`	Game ends when all mines are marked and all the other cells are shown
`expandShown``(board, elCell, i, j)`	When user clicks an empty place (0 negs), we need to open not only that cell, but also its neighbors.

	TIP: At this point you might find yourself giving each cell an id (or a class) that looks like that: "*cell-3-2*" (3 and 2 are just examples) NOTE: start with a basic implementation that only opens the two-level neighbors BONUS: if you have the time later, try to work more like the real algorithm.

Here are the **globals** you might be using:

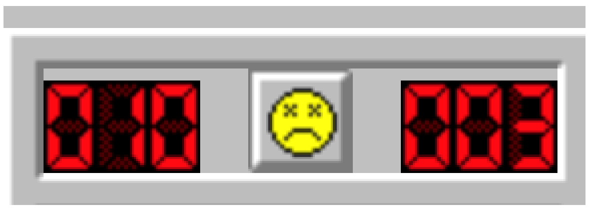
`gBoard - Matrix contains cell objects:`
`{`
`    minesAroundCount: 4,`
`    isShown: true,`
`    isMine: false,`
`    isMarked: true,`
`}`
	The model

`gLevel = {`
`    SIZE: 4,`
`    MINES: 2`
`};`
	This is an object by which the board size is set (in this case: 4*4), and how many mines to put

`gState = {`
`    isGameOn: false,`
`    shownCount: 0,`
`    markedCount: 0,`
`    secsPassed: 0`
`}`
	This is an object in which you can keep and update the current state:
	isGameOn - boolean, when true we let the user play
	shownCount: how many cells are shown
	markedCount: how many cells are marked (with a flag)
	secsPassed: how many seconds passed

Next Steps

1. Make sure the first clicked cell is never a mine (like in the real game)
   HINT: place the mines and count the neighbors only on first click.
2. Keep the best score in [local storage]() (per level) and show it on the page
3. Add this section:



Implement the following states on the smiley:

- Normal
- Sad & Dead – stepped on a mine
- Sunglasses – Victory

4. Make it look great