Minesweeper

Let's play the minesweeper game (Wikipedia, online game)!

You are given a 2D char matrix representing the game board. **'M'** represents an **unrevealed** mine, **'E'** represents an **unrevealed** empty square, **'B'** represents a **revealed** blank square that has no adjacent (above, below, left, right, and all 4 diagonals) mines, **digit** ('1' to '8') represents how many mines are adjacent to this **revealed** square, and finally **'X'** represents a **revealed** mine.

Now given the next click position (row and column indices) among all the **unrevealed** squares ('M' or 'E'), return the board after revealing this position according to the following rules:

1. If a mine ('M') is revealed, then the game is over - change it to **'X'**.
2. If an empty square ('E') with **no adjacent mines** is revealed, then change it to revealed blank ('B') and all of its adjacent **unrevealed** squares should be revealed recursively.
3. If an empty square ('E') with **at least one adjacent mine** is revealed, then change it to a digit ('1' to '8') representing the number of adjacent mines.
4. Return the board when no more squares will be revealed.

Example 1:

Input:

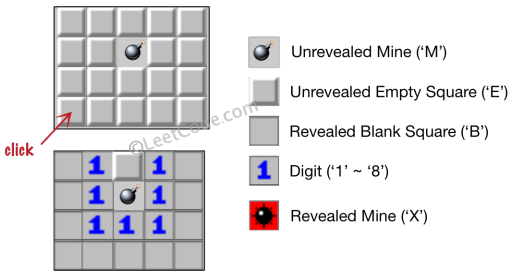
```
[['E', 'E', 'E', 'E', 'E'],
 ['E', 'E', 'M', 'E', 'E'],
 ['E', 'E', 'E', 'E', 'E'],
 ['E', 'E', 'E', 'E', 'E']]

Click : [3,0]
```

Output:

```
[['B', '1', 'E', '1', 'B'],
 ['B', '1', 'M', '1', 'B'],
 ['B', '1', '1', '1', 'B'],
 ['B', 'B', 'B', 'B', 'B']]
```

Explanation:



Example 2:

Input:

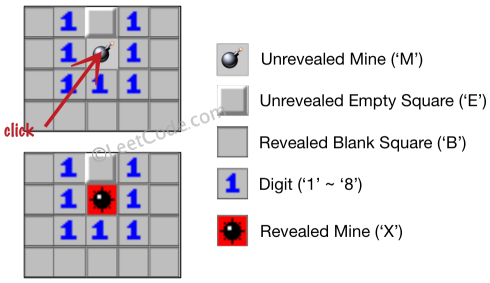
```
[['B', '1', 'E', '1', 'B'],
 ['B', '1', 'M', '1', 'B'],
 ['B', '1', '1', '1', 'B'],
 ['B', 'B', 'B', 'B', 'B']]

Click : [1,2]
```

Output:

```
[['B', '1', 'E', '1', 'B'],
 ['B', '1', 'X', '1', 'B'],
 ['B', '1', '1', '1', 'B'],
 ['B', 'B', 'B', 'B', 'B']]
```

Explanation:



Note:

1. The range of the input matrix's height and width is [1,50].
2. The click position will only be an unrevealed square ('M' or 'E'), which also means the input board contains at least one clickable square.
3. The input board won't be a stage when game is over (some mines have been revealed).
4. For simplicity, not mentioned rules should be ignored in this problem. For example, you **don't** need to reveal all the unrevealed mines when the game is over, consider any cases that you will win the game or flag any squares.

Solution 1

This is a typical `Search` problem, either by using `DFS` or `BFS`. Search rules:

1. If click on a mine (`'M'`), mark it as `'X'`, stop further search.
2. If click on an empty cell (`'E'`), depends on how many surrounding mine:
   2.1 Has surrounding mine(s), mark it with number of surrounding mine(s), stop further search.
   2.2 No surrounding mine, mark it as `'B'`, continue search its `8` neighbors.

DFS solution.

```
public class Solution {
    public char[][] updateBoard(char[][] board, int[] click) {
        int m = board.length, n = board[0].length;
        int row = click[0], col = click[1];

        if (board[row][col] == 'M') { // Mine
            board[row][col] = 'X';
        }
        else { // Empty
            // Get number of mines first.
            int count = 0;
            for (int i = -1; i < 2; i++) {
                for (int j = -1; j < 2; j++) {
                    if (i == 0 && j == 0) continue;
                    int r = row + i, c = col + j;
                    if (r < 0 || r >= m || c < 0 || c < 0 || c >= n) continue;
                    if (board[r][c] == 'M' || board[r][c] == 'X') count++;
                }
            }

            if (count > 0) { // If it is not a 'B', stop further DFS.
                board[row][col] = (char)(count + '0');
            }
            else { // Continue DFS to adjacent cells.
                board[row][col] = 'B';
                for (int i = -1; i < 2; i++) {
                    for (int j = -1; j < 2; j++) {
                        if (i == 0 && j == 0) continue;
                        int r = row + i, c = col + j;
                        if (r < 0 || r >= m || c < 0 || c < 0 || c >= n) continue
;
                        if (board[r][c] == 'E') updateBoard(board, new int[] {r,
c});
                    }
                }
            }
        }

        return board;
    }
}
```

BFS solution. As you can see the basic logic is almost the same as DFS. Only added a

queue to facilitate BFS.

```
public class Solution {
    public char[][] updateBoard(char[][] board, int[] click) {
        int m = board.length, n = board[0].length;
        Queue<int[]> queue = new LinkedList<>();
        queue.add(click);

        while (!queue.isEmpty()) {
            int[] cell = queue.poll();
            int row = cell[0], col = cell[1];

            if (board[row][col] == 'M') { // Mine
                board[row][col] = 'X';
            }
            else { // Empty
                // Get number of mines first.
                int count = 0;
                for (int i = -1; i < 2; i++) {
                    for (int j = -1; j < 2; j++) {
                        if (i == 0 && j == 0) continue;
                        int r = row + i, c = col + j;
                        if (r < 0 || r >= m || c < 0 || c < 0 || c >= n) continue
;
                        if (board[r][c] == 'M' || board[r][c] == 'X') count++;
                    }
                }

                if (count > 0) { // If it is not a 'B', stop further DFS.
                    board[row][col] = (char)(count + '0');
                }
                else { // Continue BFS to adjacent cells.
                    board[row][col] = 'B';
                    for (int i = -1; i < 2; i++) {
                        for (int j = -1; j < 2; j++) {
                            if (i == 0 && j == 0) continue;
                            int r = row + i, c = col + j;
                            if (r < 0 || r >= m || c < 0 || c < 0 || c >= n) cont
inue;
                            if (board[r][c] == 'E') {
                                queue.add(new int[] {r, c});
                                board[r][c] = 'B'; // Avoid to be added again.
                            }
                        }
                    }
                }
            }
        }

        return board;
    }
}
```

written by shawngao original link here

Solution 2

```
public class Solution {
    public char[][] updateBoard(char[][] board, int[] click) {
        int x = click[0], y = click[1];
        if (board[x][y] == 'M') {
            board[x][y] = 'X';
            return board;
        }

        dfs(board, x, y);
        return board;
    }

    int[] dx = {-1, 0, 1, -1, 1, 0, 1, -1};
    int[] dy = {-1, 1, 1, 0, -1, -1, 0, 1};
    private void dfs(char[][] board, int x, int y) {
        if (x < 0 || x >= board.length || y < 0 || y >= board[0].length || board[
x][y] != 'E')  return;

        int num = getNumsOfBombs(board, x, y);

        if (num == 0) {
            board[x][y] = 'B';
            for (int i = 0; i < 8; i++) {
                int nx = x + dx[i], ny = y + dy[i];
                dfs(board, nx, ny);
            }
        } else {
            board[x][y] = (char)('0' + num);
        }

    }

    private int getNumsOfBombs(char[][] board, int x, int y) {
        int num = 0;
        for (int i = -1; i <= 1; i++) {
            for (int j = -1; j <= 1; j++) {
                int nx = x + i, ny = y + j;
                if (nx < 0 || nx >= board.length || ny < 0 || ny >= board[0].leng
th)    continue;
                if (board[nx][ny] == 'M' || board[nx][ny] == 'X') {
                    num++;
                }
            }
        }
        return num;
    }
}
```

written by tankztc original link here

Solution 3

Maintain a todo list (***stack***) of squares left to reveal. For each square S to be revealed, if S contains a mine, put an X in it. If there are mines adjacent to S, then put that number of mines in it. If there are no mines adjacent to S, then put a B in it, and add adjacent squares to S that are unrevealed (*in 'ME'*) that were not previously added (***nei*** *not in seen*) to your todo list.

```
def updateBoard(self, A, click):
    click = tuple(click)
    R, C = len(A), len(A[0])

    def neighbors(r, c):
        for dr in xrange(-1, 2):
            for dc in xrange(-1, 2):
                if (dr or dc) and 0 <= r + dr < R and 0 <= c + dc < C:
                    yield r + dr, c + dc

    stack = [click]
    seen = {click}
    while stack:
        r, c = stack.pop()
        if A[r][c] == 'M':
            A[r][c] = 'X'
        else:
            mines_adj = sum( A[nr][nc] in 'MX' for nr, nc in neighbors(r, c) )
            if mines_adj:
                A[r][c] = str(mines_adj)
            else:
                A[r][c] = 'B'
                for nei in neighbors(r, c):
                    if A[nei[0]][nei[1]] in 'ME' and nei not in seen:
                        stack.append(nei)
                        seen.add(nei)
    return A
```

written by awice original link here