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:

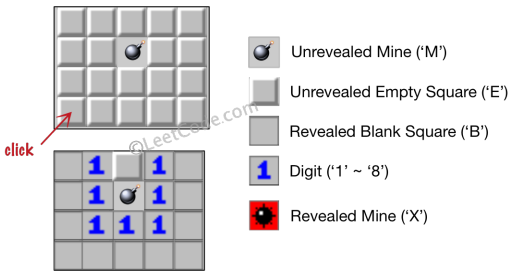
```
[['E', 'E', 'E', 'E', 'E'],
 ['E', 'E', 'M', 'E', 'E'],
 ['E', 'E', 'E', 'E', 'E'],
 ['E', 'E', 'E', 'E', 'E']]

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:

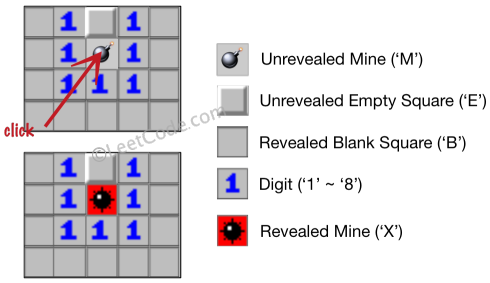
```
[['B', '1', 'E', '1', 'B'],
 ['B', '1', 'M', '1', 'B'],
 ['B', '1', '1', '1', 'B'],
 ['B', 'B', 'B', 'B', 'B']]

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

Make sure you know the rule of updating the board.

1. When clicking on the unrevealed mine, just update the current cell to 'X'.
2. When clicking on the cell with mines nearby (8 neighbors including diagonals), just update the count of mines in the neighborhood.
3. When clicking on the cell with 0 mine nearby, all the 8 neighbors also need to be checked and updated. And since once a cell is visited, its character will be changed for sure, the board will record the visited cells itself so no worry about revisiting.

ps: I know the format is somewhat stupid lol.

```java
public char[][] updateBoard(char[][] board, int[] click) {
    int x = click[0];
    int y = click[1];
    if(board[x][y] == 'M') board[x][y] = 'X';
    else if(countmines(board,x,y)>0) board[x][y] = (char)(countmines(board,x,y) +
'0');
    else update(board,x,y);
    return board;
}
private void update(char[][] board, int i, int j){
    if(i<0||i>=board.length||j<0||j>=board[0].length) return;
    if(board[i][j]=='E'){
        if(countmines(board,i,j)==0) {
            board[i][j] = 'B';
            update(board,i,j-1);
            update(board,i-1,j);
            update(board,i,j+1);
            update(board,i+1,j);
            update(board,i-1,j-1);
            update(board,i+1,j+1);
            update(board,i+1,j-1);
            update(board,i-1,j+1);
        }
        else{
            board[i][j] = (char)(countmines(board,i,j) + '0');
        }
    }
}

private int countmines(char[][] board,int i, int j){ // just count mines in the ne
ighborhood.
    int count = 0;
    if(i-1>=0&&board[i-1][j]=='M')count++;
    if(i+1<board.length&&board[i+1][j]=='M')count++;
    if(j-1>=0&&board[i][j-1]=='M')count++;
    if(j+1<board[0].length&&board[i][j+1]=='M')count++;
    if(i-1>=0&&j-1>=0&&board[i-1][j-1]=='M')count++;
    if(i+1<board.length&&j+1<board[0].length&&board[i+1][j+1]=='M') count++;
    if(i-1>=0&&j+1<board[0].length&&board[i-1][j+1]=='M') count++;
    if(i+1<board.length&&j-1>=0&&board[i+1][j-1]=='M') count++;
    return count;
}
```

written by Nakanu original link here