LC 529. Minesweeper

Question

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

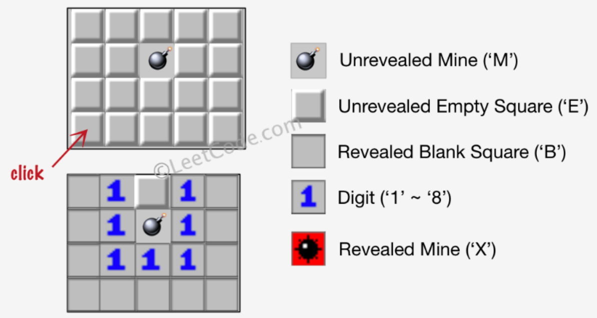
You are given a 2D char matrix representing the game board. **'M'** represents an **unrevealed** mine, **'E'** represents an **unrevealed** empty square, **'B'** represents a **revealed** blank square that has no adjacent (above, below, left, right, and all 4 diagonals) mines, **digit** ('1' to '8')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



Solution

```python
class Solution:
    def updateBoard(self, board: List[List[str]], click: List[int]) -> 
List[List[str]]:
        #Solution 2
        if not board or not board[0]:
            return board
        m = len(board)
        n = len(board[0])
        directions = [(0,1), (0,-1), (1,0), (-1,0), (1,1), (-1,-1), (1,-1), 
(-1,1)]
        x, y = click
        if board[x][y] == 'M':
            board[x][y] = 'X'
        elif board[x][y] == 'E':
            count = 0
            for dx, dy in directions:
                new_x = x + dx
                new_y = y + dy
                if 0 <= new_x < m and 0 <= new_y < n and board[new_x][new_y] == 
'M':
                    count += 1
            if count > 0:
                board[x][y] = str(count)
            else:
                board[x][y] = 'B'
                for dx, dy in directions:
                    new_x = x + dx
                    new_y = y + dy
                    if 0 <= new_x < m and 0 <= new_y < n and board[new_x][new_y] 
== 'E':
                        self.updateBoard(board, [new_x, new_y])
        return board


        #Solution
        if not board:
            return []

        m, n = len(board), len(board[0])
        i, j = click[0], click[1]

        # If a mine ('M') is revealed, then the game is over - change it to 'X'.
        if board[i][j] == 'M':
            board[i][j] = 'X'
            return board
        # run dfs to reveal the board
        self.dfs(board, i, j)
        return board

    def dfs(self, board, i, j):
        if board[i][j] != 'E':
            return

        m, n = len(board), len(board[0])
        directions = [(-1,-1), (0,-1), (1,-1), (1,0), (1,1), (0,1), (-1,1),
(-1,0)]

        mine_count = 0

        for d in directions:
            ni, nj = i + d[0], j + d[1]
            if 0 <= ni < m and 0 <= nj < n and board[ni][nj] == 'M':
                mine_count += 1

        if mine_count == 0:
            board[i][j] = 'B'
        else:
            board[i][j] = str(mine_count)
            return

        for d in directions:
            ni, nj = i + d[0], j + d[1]
            if 0 <= ni < m and 0 <= nj < n:
                self.dfs(board, ni, nj)
```