Problem 289: Game of Life

Problem Information

Difficulty: Medium
Acceptance Rate: 72.03%
Paid Only: No
Tags: Array, Matrix, Simulation

Problem Description

According to [Wikipedia's article](https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life): "The **Game of Life** , also known simply as **Life** , is a cellular automaton devised by the British mathematician John Horton Conway in 1970."

The board is made up of an `m x n` grid of cells, where each cell has an initial state: **live** (represented by a `1`) or **dead** (represented by a `0`). Each cell interacts with its [eight neighbors](https://en.wikipedia.org/wiki/Moore_neighborhood) (horizontal, vertical, diagonal) using the following four rules (taken from the above Wikipedia article):

1. Any live cell with fewer than two live neighbors dies as if caused by under-population. 2. Any live cell with two or three live neighbors lives on to the next generation. 3. Any live cell with more than three live neighbors dies, as if by over-population. 4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

The next state of the board is determined by applying the above rules simultaneously to every cell in the current state of the `m x n` grid `board`. In this process, births and deaths occur **simultaneously**.

Given the current state of the `board`, **update** the `board` to reflect its next state.

Note that you do not need to return anything.

Example 1:

![](https://assets.leetcode.com/uploads/2020/12/26/grid1.jpg)

Input: board = [[0,1,0],[0,0,1],[1,1,1],[0,0,0]] **Output:** [[0,0,0],[1,0,1],[0,1,1],[0,1,0]]

Example 2:

![](https://assets.leetcode.com/uploads/2020/12/26/grid2.jpg)

Input: board = [[1,1],[1,0]] **Output:** [[1,1],[1,1]]

Constraints:

* `m == board.length` * `n == board[i].length` * `1 <= m, n <= 25` * `board[i][j]` is `0` or `1`.

Follow up:

* Could you solve it in-place? Remember that the board needs to be updated simultaneously: You cannot update some cells first and then use their updated values to update other cells. * In this question, we represent the board using a 2D array. In principle, the board is infinite, which would cause problems when the active area encroaches upon the border of the array (i.e., live cells reach the border). How would you address these problems?

Code Snippets

C++:

```
class Solution {
public:
void gameOfLife(vector<vector<int>>& board) {

}
};
```

Java:

```
class Solution {
public void gameOfLife(int[][] board) {

}
}
```

Python3:

```
class Solution:
def gameOfLife(self, board: List[List[int]]) -> None:
"""
Do not return anything, modify board in-place instead.
"""
```