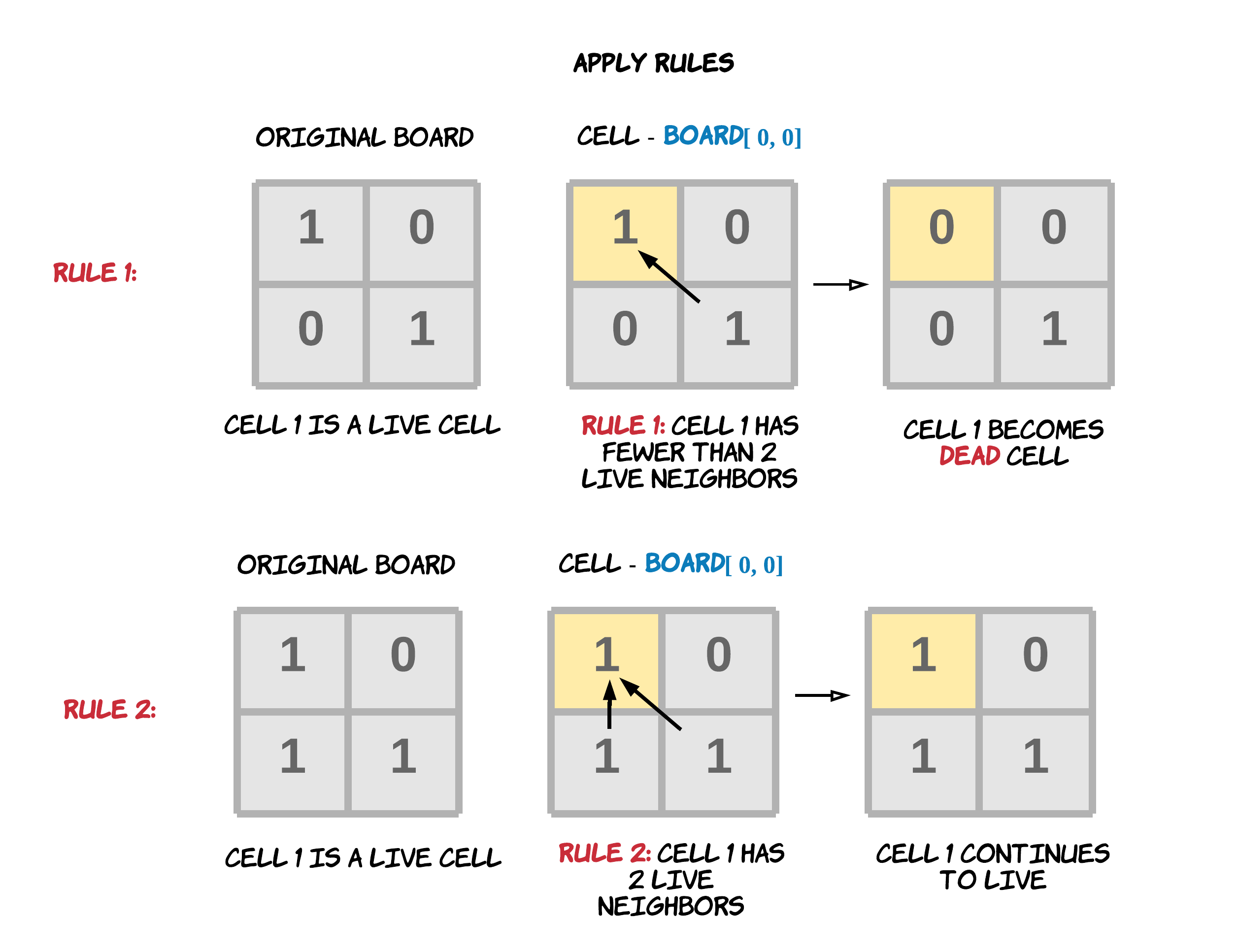


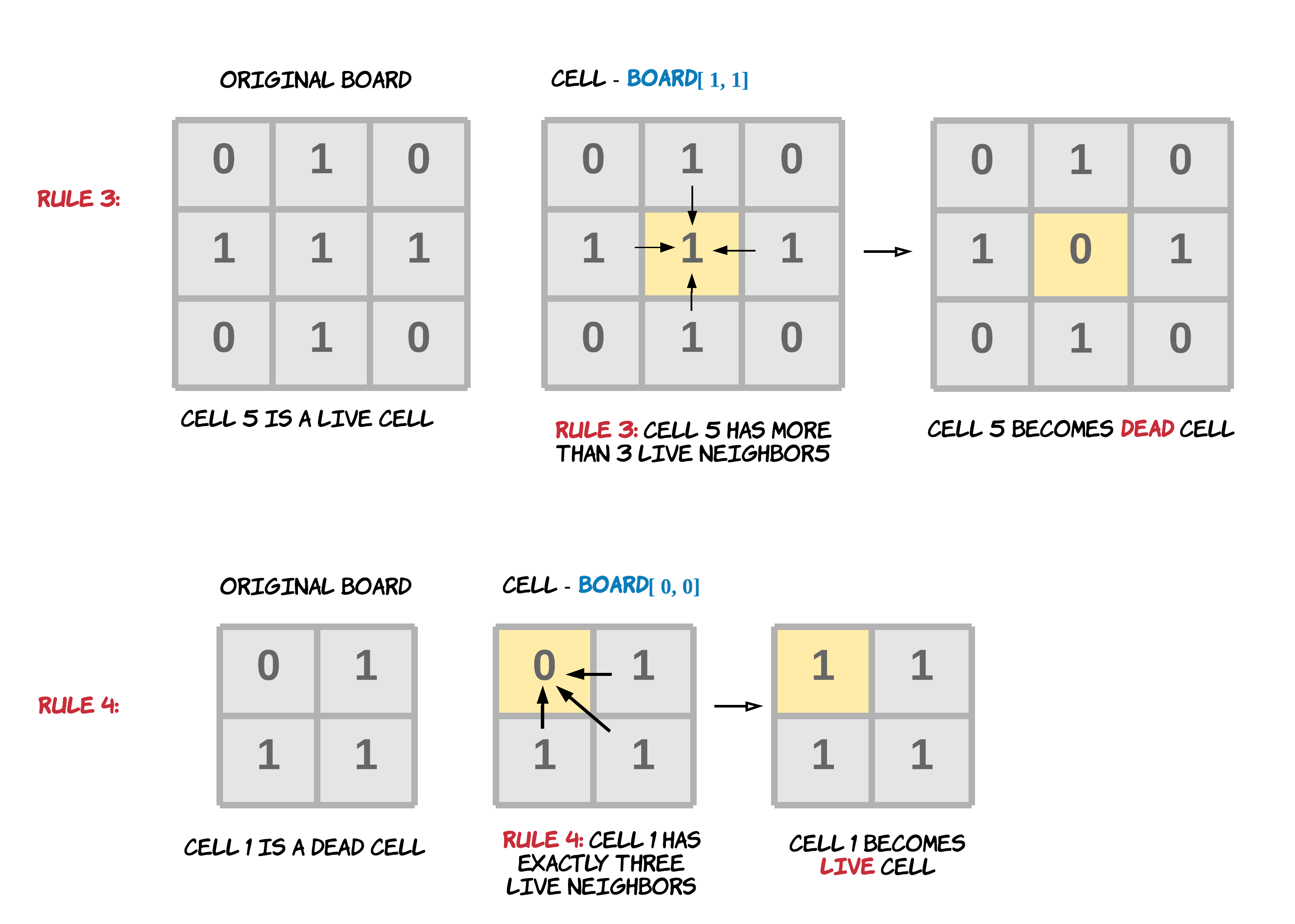
生命游戏

[力扣官方题解](https://leetcode-cn.com/u/leetcode-solution/)发布于 2 天前16.1k**官方**C++JavaPython数组

**分析**

在讲具体解法之前，请先根据下面的图片理解题目中描述的细胞遵循的生存定律，这有助于我们后面的讲解。

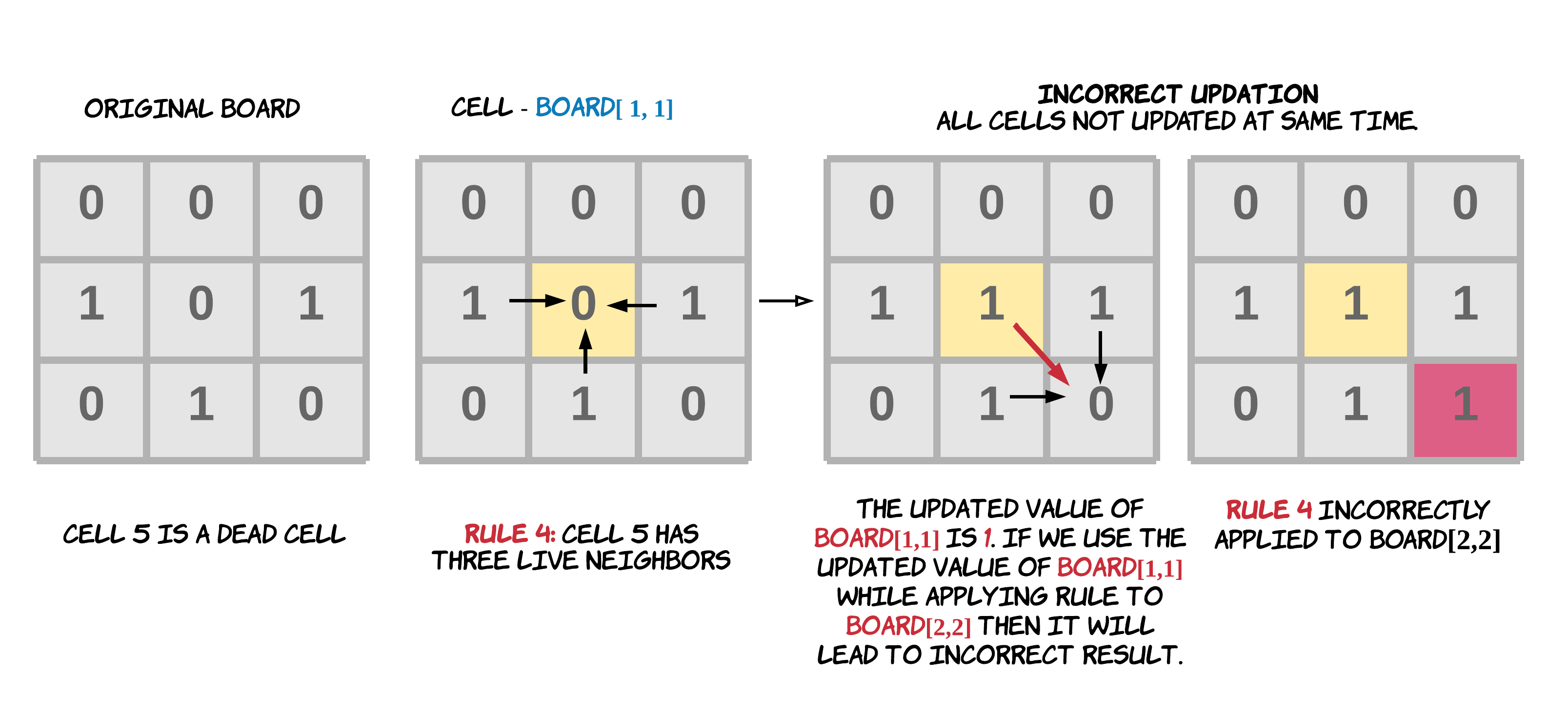




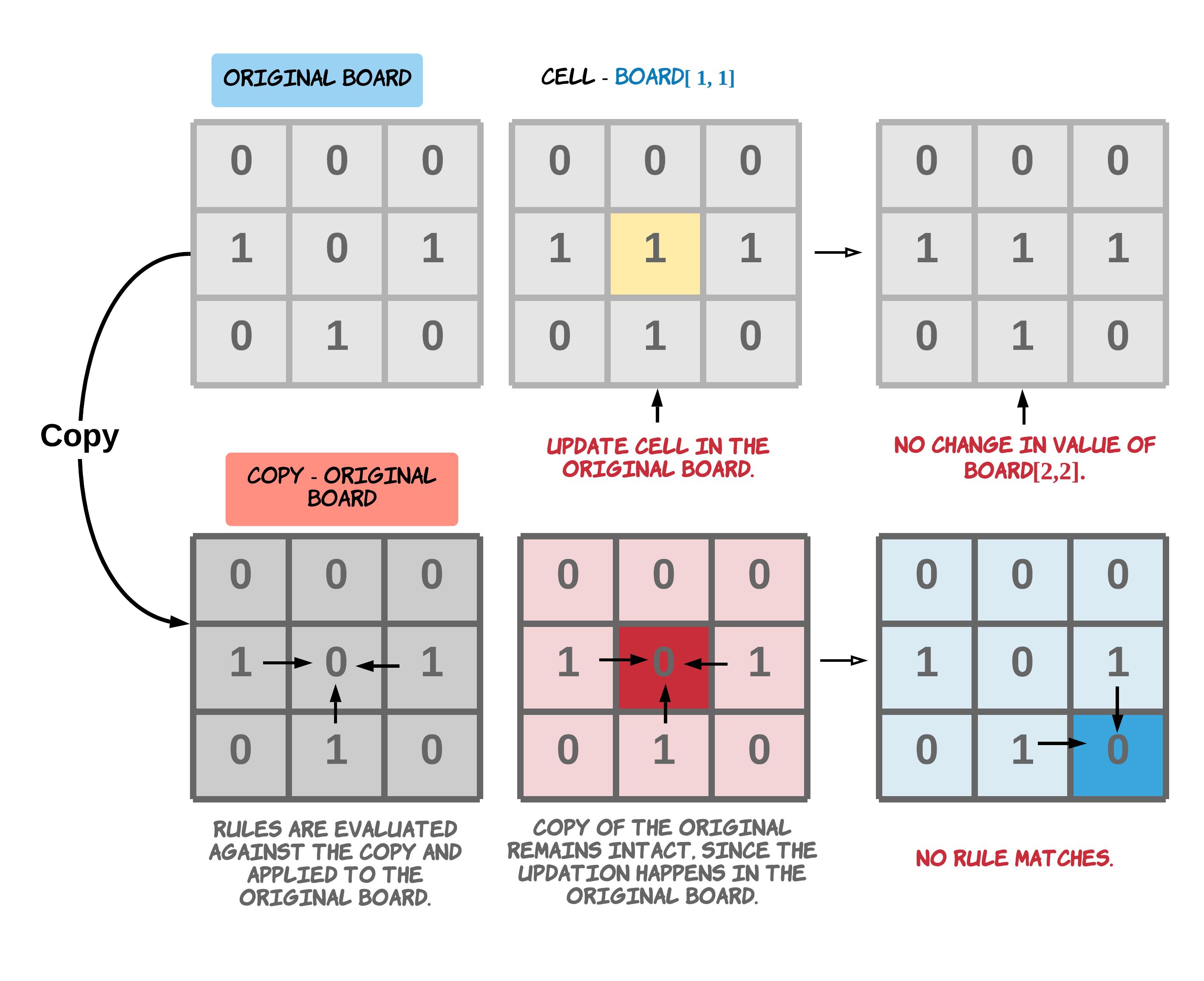
**方法一：复制原数组进行模拟**

**思路**

这个问题看起来很简单，但有一个陷阱，如果你直接根据规则更新原始数组，那么就做不到题目中说的 **同步** 更新。假设你直接将更新后的细胞状态填入原始数组，那么当前轮次其他细胞状态的更新就会引用到当前轮已更新细胞的状态，但实际上每一轮更新需要依赖上一轮细胞的状态，是不能用这一轮的细胞状态来更新的。



如上图所示，已更新细胞的状态会影响到周围其他还未更新细胞状态的计算。一个最简单的解决方法就是复制一份原始数组，复制的那一份永远不修改，只作为更新规则的引用。这样原始数组的细胞值就不会被污染了。



**算法**

* 复制一份原始数组；
* 根据复制数组中邻居细胞的状态来更新 board 中的细胞状态。
* C++
* Java
* Python3

class Solution {

public:

void gameOfLife(vector<vector<int>>& board) {

int neighbors[3] = {0, 1, -1};

int rows = board.size();

int cols = board[0].size();

// 创建复制数组 copyBoard

vector<vector<int> >copyBoard(rows, vector<int>(cols, 0));

// 从原数组复制一份到 copyBoard 中

for (int row = 0; row < rows; row++) {

for (int col = 0; col < cols; col++) {

copyBoard[row][col] = board[row][col];

}

}

// 遍历面板每一个格子里的细胞

for (int row = 0; row < rows; row++) {

for (int col = 0; col < cols; col++) {

// 对于每一个细胞统计其八个相邻位置里的活细胞数量

int liveNeighbors = 0;

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (!(neighbors[i] == 0 && neighbors[j] == 0)) {

int r = (row + neighbors[i]);

int c = (col + neighbors[j]);

// 查看相邻的细胞是否是活细胞

if ((r < rows && r >= 0) && (c < cols && c >= 0) && (copyBoard[r][c] == 1)) {

liveNeighbors += 1;

}

}

}

}

// 规则 1 或规则 3

if ((copyBoard[row][col] == 1) && (liveNeighbors < 2 || liveNeighbors > 3)) {

board[row][col] = 0;

}

// 规则 4

if (copyBoard[row][col] == 0 && liveNeighbors == 3) {

board[row][col] = 1;

}

}

}

}

};

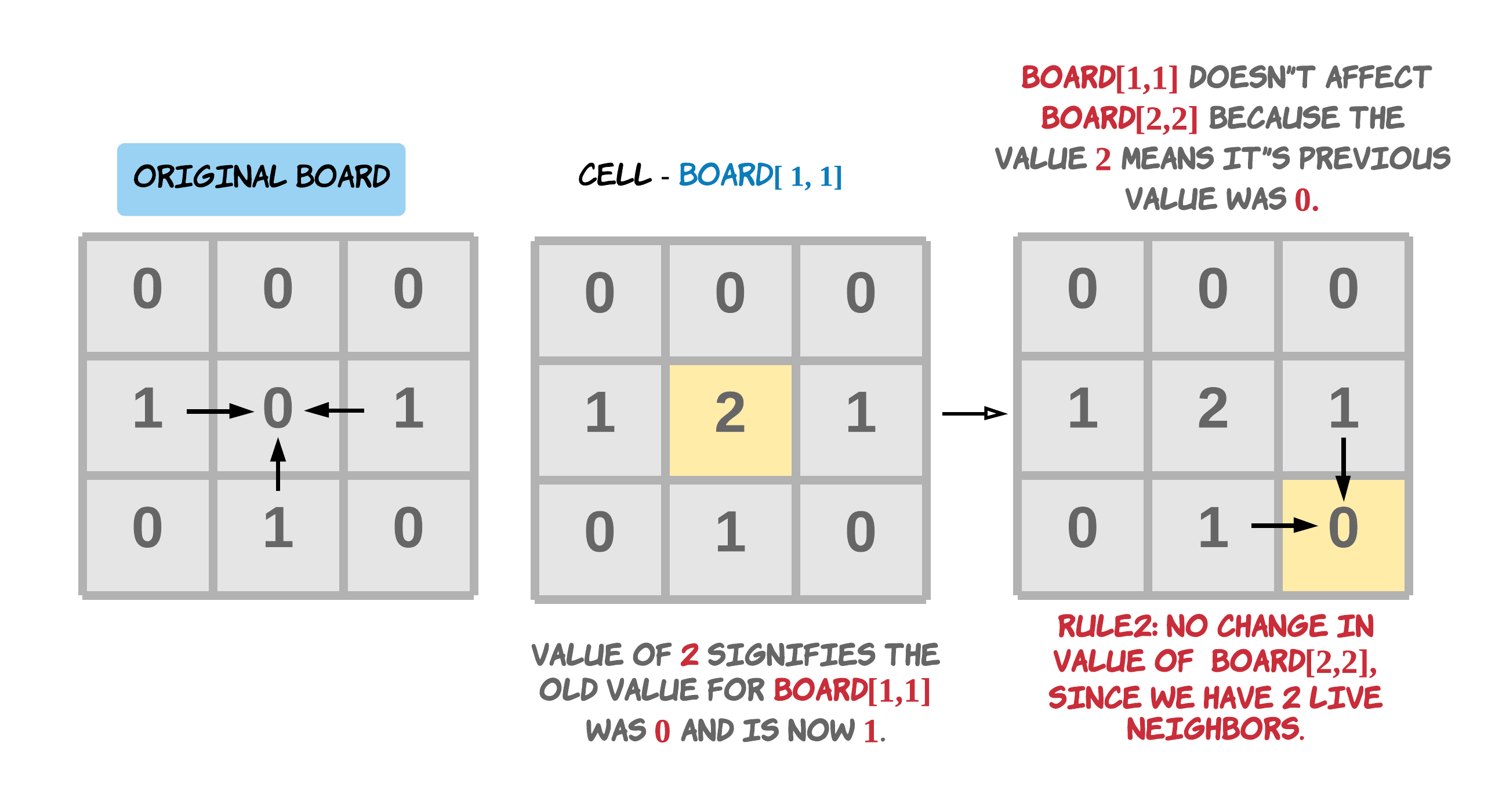
**复杂度分析**

* 时间复杂度：O(mn)*O*(*mn*)，其中 m*m* 和 n*n* 分别为 board 的行数和列数。
* 空间复杂度：O(mn)*O*(*mn*)，为复制数组占用的空间。

**方法二：使用额外的状态**

**思路**

方法一中 O(mn)*O*(*mn*) 的空间复杂度在数组很大的时候内存消耗是非常昂贵的。题目中每个细胞只有两种状态 live(1) 或 dead(0)，但我们可以拓展一些复合状态使其包含之前的状态。举个例子，如果细胞之前的状态是 0，但是在更新之后变成了 1，我们就可以给它定义一个复合状态 2。这样我们看到 2，既能知道目前这个细胞是活的，还能知道它之前是死的。



**算法**

* 遍历 board 中的细胞。
* 根据数组的细胞状态计算新一轮的细胞状态，这里会用到能同时代表过去状态和现在状态的复合状态。
* 具体的计算规则如下所示：
  + 规则 1：如果活细胞周围八个位置的活细胞数少于两个，则该位置活细胞死亡。这时候，将细胞值改为 -1，代表这个细胞过去是活的现在死了；
  + 规则 2：如果活细胞周围八个位置有两个或三个活细胞，则该位置活细胞仍然存活。这时候不改变细胞的值，仍为 1；
  + 规则 3：如果活细胞周围八个位置有超过三个活细胞，则该位置活细胞死亡。这时候，将细胞的值改为 -1，代表这个细胞过去是活的现在死了。可以看到，因为规则 1 和规则 3 下细胞的起始终止状态是一致的，因此它们的复合状态也一致；
  + 规则 4：如果死细胞周围正好有三个活细胞，则该位置死细胞复活。这时候，将细胞的值改为 2，代表这个细胞过去是死的现在活了。
* 根据新的规则更新数组；
* 现在复合状态隐含了过去细胞的状态，所以我们可以在不复制数组的情况下完成原地更新；
* 对于最终的输出，需要将 board 转成 0，1 的形式。因此这时候需要再遍历一次数组，将复合状态为 2 的细胞的值改为 1，复合状态为 -1 的细胞的值改为 0。
* C++
* Java
* Python3

class Solution {

public:

void gameOfLife(vector<vector<int>>& board) {

int neighbors[3] = {0, 1, -1};

int rows = board.size();

int cols = board[0].size();

// 遍历面板每一个格子里的细胞

for (int row = 0; row < rows; row++) {

for (int col = 0; col < cols; col++) {

// 对于每一个细胞统计其八个相邻位置里的活细胞数量

int liveNeighbors = 0;

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (!(neighbors[i] == 0 && neighbors[j] == 0)) {

// 相邻位置的坐标

int r = (row + neighbors[i]);

int c = (col + neighbors[j]);

// 查看相邻的细胞是否是活细胞

if ((r < rows && r >= 0) && (c < cols && c >= 0) && (abs(board[r][c]) == 1)) {

liveNeighbors += 1;

}

}

}

}

// 规则 1 或规则 3

if ((board[row][col] == 1) && (liveNeighbors < 2 || liveNeighbors > 3)) {

// -1 代表这个细胞过去是活的现在死了

board[row][col] = -1;

}

// 规则 4

if (board[row][col] == 0 && liveNeighbors == 3) {

// 2 代表这个细胞过去是死的现在活了

board[row][col] = 2;

}

}

}

// 遍历 board 得到一次更新后的状态

for (int row = 0; row < rows; row++) {

for (int col = 0; col < cols; col++) {

if (board[row][col] > 0) {

board[row][col] = 1;

} else {

board[row][col] = 0;

}

}

}

}

};

**复杂度分析**

* 时间复杂度：O(mn)*O*(*mn*)，其中 m*m*，n*n* 分别为 board 的行数和列数。
* 空间复杂度：O(1)*O*(1)，除原数组外只需要常数的空间存放若干变量。

下一篇：C++ 位运算原地操作，击败双百。

© 著作权归作者所有

81

条评论

最热



编辑

预览

评论

精选评论(3)

[[](https://leetcode-cn.com/u/lucien_z/)Lucien](https://leetcode-cn.com/u/lucien_z/)

（编辑过）14 小时前

我又来帮大家捋一捋思路了。这题原理没啥复杂，关键就是规则挺麻烦，怎么能捋清规则以及合理设计复合状态呢？且看我来捋一捋。

一. 普遍意义上，如果规则极其繁复，如何简化这些规则呢？

关于逻辑表达式的简化可能会用上“卡诺图”，有兴趣的朋友请自行查一查。

二. 这道题的规则如何简化？

1. 原来是活的，周围有2-3个活的，成为活的

2. 原来是死的，周围有3个活的，成为活的

3. 其他都是死了

三. 复合状态如何设计？

说真的，状态稍微多一点，这种用特定的新数字表示新状态的做法容易把自己搞懵，又是2又是-1的，自己的代码隔一天看，自己都捋不顺。

我咋干的？首先遍历的时候我不让每一个格子都看一眼周围的八个格子去更新自己状态。我换个角度，遍历的时候，如果这个格子里是活的，我就让它去“影响”周围的八个格子。这样一来，大批原来是死了的格子就省了很多检查的时间。怎么“影响”？简单，我给被影响的格子里的数字加10。这样一来，个位存着这格子原来的状态，十位就存着它周围有多少个活格子了。比如遍历完了之后一个格子里是41，那就表示它原来自己是1，然后被周围的四个活格子加了四个10，于是周围有四个活细胞。

等之后再遍历一遍，更新到最新状态就完事了。代码：

class Solution:

def gameOfLife(self, board: List[List[int]]) -> None:

m=len(board)

n=len(board[0])

def affect(x,y):

for i in [x-1,x,x+1]:

for j in [y-1,y,y+1]:

if i<0 or i>=m or j<0 or j>=n or (i==x and j==y):

continue

board[i][j]+=10

for i in range(m):

for j in range(n):

if board[i][j]%10==1:

affect(i,j)

def calculate(x,y):

value=board[x][y]

if value//10==3:

board[x][y]=1

elif value//10==2 and value%10==1:

board[x][y]=1

else:

board[x][y]=0

for i in range(m):

for j in range(n):

calculate(i,j)

72踩查看 16 条回复回复

[[](https://leetcode-cn.com/u/modest-antonelli/)JensonSuen](https://leetcode-cn.com/u/modest-antonelli/)

1 天前

4月是不是没有easy的题目了，弟弟我瑟瑟发抖

36踩查看 7 条回复回复

[[](https://leetcode-cn.com/u/bai-ma-hu-xiao-long-wang/)前端业务员](https://leetcode-cn.com/u/bai-ma-hu-xiao-long-wang/)

15 小时前

四月的卡我怕是打不动了哦

19踩回复

评论(81)

[[](https://leetcode-cn.com/u/andyliu-2/)AndyLiu](https://leetcode-cn.com/u/andyliu-2/)

15 小时前

第二个牛批啊，经过上个月的训练，这回找每个点的八邻域我已经学会不用一堆if判断了，这回又学会了不用复制列表解决，我害搁这deepcopy呢

9踩查看 3 条回复回复

[[](https://leetcode-cn.com/u/leetcode_cnnn/)隐士2020](https://leetcode-cn.com/u/leetcode_cnnn/)

13 小时前

看来leetcode题目的长度是一天比一长

8踩回复

[[](https://leetcode-cn.com/u/july-13/)july](https://leetcode-cn.com/u/july-13/)

7 小时前

最后的状态设计可以考虑使用位来实现，这样会比较容易记忆；老的状态就是0和1，其实只占用了最低位，而对于int有32位，我们完全可以利用最低位的前一位来记录现在的状态，如果需要旧的状态就是 board[i][j] & 1;如果需要新的状态就是 board[i][j] & 2 ；

3踩查看 1 条回复回复

[[](https://leetcode-cn.com/u/user5134/)HzDeZh](https://leetcode-cn.com/u/user5134/)

4 小时前

发现有一些小伙伴看不懂neighbors，稍微修改了一下，简化了neighbors，供大家参考。

class Solution {

public void gameOfLife(int[][] board) {

int m = board.length;

int n = board[0].length;

for(int i = 0; i < m; i++) {

for(int j = 0; j < n; j++) {

int count = board[i][j] == 1 ? -1 : 0;

for(int r = -1; r < 2; r++) {

for(int c = -1; c < 2; c++) {

int rr = i + r;

int cc = j + c;

if ((rr < m && rr >= 0) && (cc < n && cc >= 0) && (Math.abs(board[rr][cc]) == 1)) {

count++;

}

}

}

if(board[i][j] == 1 && (count < 2 || count > 3))

board[i][j] = -1;

if(board[i][j] == 0 && count == 3)

board[i][j] = 2;

}

}

for(int i = 0; i < m; i++) {

for(int j = 0; j < n; j++) {

if(board[i][j] > 0){

board[i][j] = 1;

}else{

board[i][j] = 0;

}

}

}

}

}

2踩查看 1 条回复回复

[[](https://leetcode-cn.com/u/shou-lu/)飞翔的企鹅](https://leetcode-cn.com/u/shou-lu/)

13 小时前

只给出了提示的第一点的解释。原谅我，第二点“原则上，面板是无限的，但当活细胞侵占了面板边界时会造成问题。你将如何解决这些问题？”我看不懂，这是想说什么

1踩查看 3 条回复回复

[[](https://leetcode-cn.com/u/wulinlw/)wulinlw](https://leetcode-cn.com/u/wulinlw/)

11 小时前

abs(board[newx][newy])==1 太妙了，活着的细胞死去标记-1，abs后就不会影响后面判断 有些细胞死了，但他还活着

1踩查看 2 条回复回复

[[](https://leetcode-cn.com/u/li-li-jie-xin-ku/)粒粒皆辛苦](https://leetcode-cn.com/u/li-li-jie-xin-ku/)

13 小时前

neighbors[3] = {-1, 0, 1}没看懂，哪个大佬给解释以下

赞踩查看 3 条回复回复

[[](https://leetcode-cn.com/u/xuan-xiao-10/)喧嚣](https://leetcode-cn.com/u/xuan-xiao-10/)

12 小时前

int r = (row + neighbors[i]);

int c = (col + neighbors[j]);

这俩步怎么理解

赞踩查看 2 条回复回复

[[](https://leetcode-cn.com/u/momomarihoo/)momomarihoo](https://leetcode-cn.com/u/momomarihoo/)

12 小时前

您好！ 法一中 neighbors = [(1,0), (1,-1), (0,-1), (-1,-1), (-1,0), (-1,1), (0,1), (1,1)] 这个的输出结果是什么呢？ 为什么没有规则二呢

赞踩查看 2 条回复回复

[[](https://leetcode-cn.com/u/ichol-2/)Ichol](https://leetcode-cn.com/u/ichol-2/)

18 小时前

想问问给copy赋值不能直接 int[][] copy =board吗？一定要遍历赋值吗？

赞踩查看 4 条回复回复

[[](https://leetcode-cn.com/u/luiew/)真是令人头大](https://leetcode-cn.com/u/luiew/)

8 小时前

请问int neighbors[3] = {0, 1, -1}; 是什么意思？

赞踩查看 1 条回复回复

[[](https://leetcode-cn.com/u/zhuobo/)Drinkwater](https://leetcode-cn.com/u/zhuobo/)

1 天前

今晚先写复制一个原来数组用于判断的方法吧，不符合题目要求！明早再看看大佬怎么写原地方法

class Solution(object):

def gameOfLife(self, board):

"""

:type board: List[List[int]]

:rtype: None Do not return anything, modify board in-place instead.

"""

copy\_board = [[0 for \_ in range(len(board[0]))] for \_ in range(len(board))]

for i in range(len(board)):

for j in range(len(board[0])):

if board[i][j] == 1:

copy\_board[i][j] = 1

for i in range(len(copy\_board)):

for j in range(len(copy\_board[0])):

board[i][j] = self.check\_and\_update(copy\_board, i, j)

def check\_and\_update(self, board, i, j):

count = 0

# 8 directions in the board

directions = [(0, 1), (1, 0), (0, -1), (-1, 0), (1, 1), (-1, -1), (1, -1), (-1, 1)]

for di, dj in directions:

new\_i, new\_j = i + di, j + dj

if new\_i >= 0 and new\_i < len(board) and new\_j >= 0 and new\_j < len(board[0]):

if board[new\_i][new\_j] == 1:

count += 1

if board[i][j] == 1:

if count == 2 or count == 3:

return 1

else:

return 0

else:

if count == 3:

return 1

return 0

赞踩查看 1 条回复回复分享举报