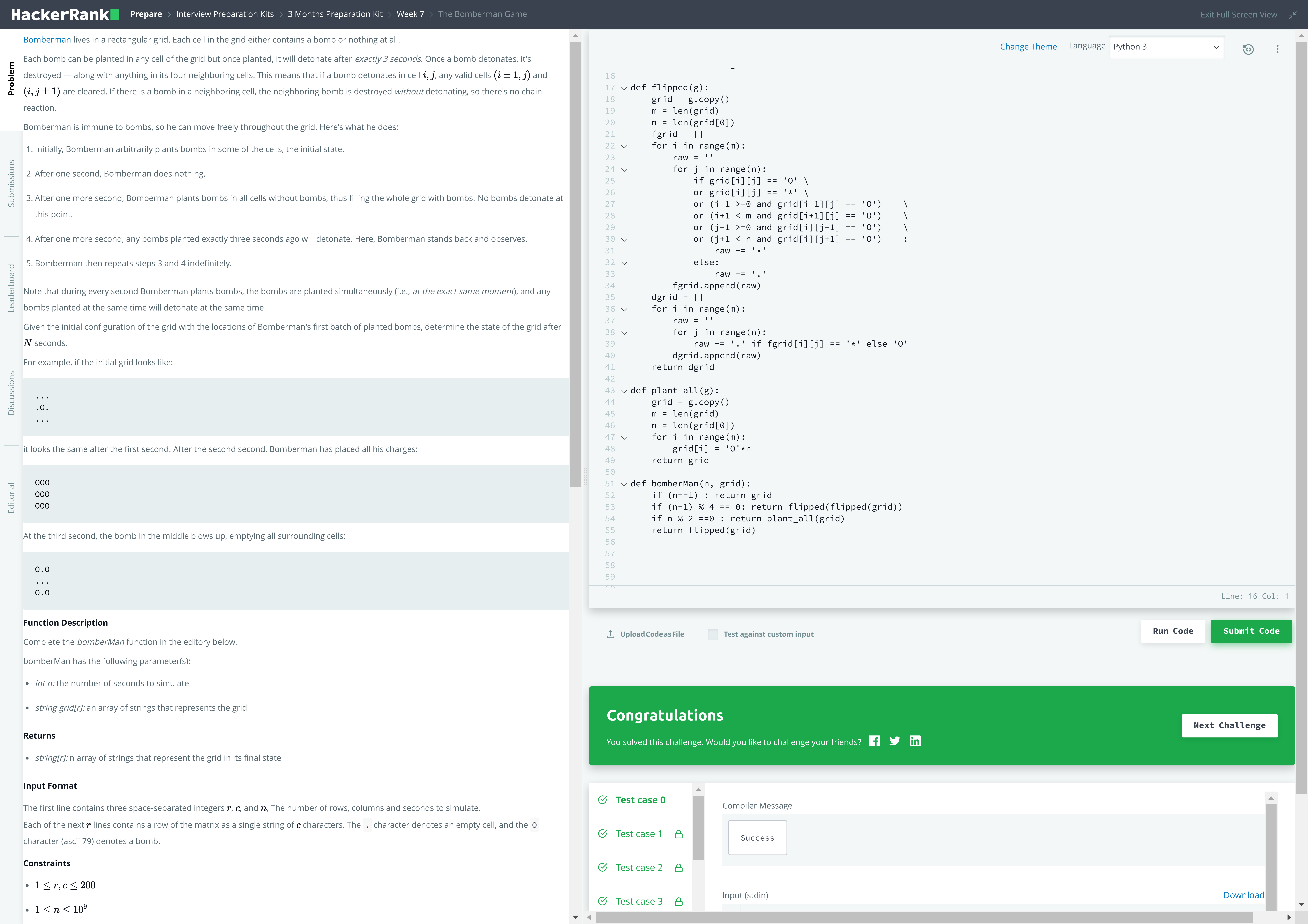

Bomberman lives in a rectangular grid. Each cell in the grid either contains a bomb or nothing at all.

Each bomb can be planted in any cell of the grid but once planted, it will detonate after *exactly 3 seconds*. Once a bomb detonates, it's destroyed — along with anything in its four neighboring cells. This means that if a bomb detonates in cell i, j, any valid cells $(i \pm 1, j)$ and $(i, j \pm 1)$ are cleared. If there is a bomb in a neighboring cell, the neighboring bomb is destroyed *without* detonating, so there's no chain reaction.

Bomberman is immune to bombs, so he can move freely throughout the grid. Here's what he does:

1. Initially, Bomberman arbitrarily plants bombs in some of the cells, the initial state.
2. After one second, Bomberman does nothing.
3. After one more second, Bomberman plants bombs in all cells without bombs, thus filling the whole grid with bombs. No bombs detonate at this point.
4. After one more second, any bombs planted exactly three seconds ago will detonate. Here, Bomberman stands back and observes.
5. Bomberman then repeats steps 3 and 4 indefinitely.

Note that during every second Bomberman plants bombs, the bombs are planted simultaneously (i.e., *at the exact same moment*), and any bombs planted at the same time will detonate at the same time.

Given the initial configuration of the grid with the locations of Bomberman's first batch of planted bombs, determine the state of the grid after N seconds.

For example, if the initial grid looks like:

```
...
.O.
...
```

it looks the same after the first second. After the second second, Bomberman has placed all his charges:

```
OOO
OOO
OOO
```

At the third second, the bomb in the middle blows up, emptying all surrounding cells:

```
O.O
...
O.O
```

Function Description

Complete the *bomberMan* function in the editory below.

bomberMan has the following parameter(s):

- *int n:* the number of seconds to simulate
- *string grid[r]:* an array of strings that represents the grid

Returns

- *string[r]:* n array of strings that represent the grid in its final state

Input Format

The first line contains three space-separated integers r, c, and n, The number of rows, columns and seconds to simulate.

Each of the next r lines contains a row of the matrix as a single string of c characters. The `.` character denotes an empty cell, and the `O` character (ascii 79) denotes a bomb.

Constraints

- $1 \leq r, c \leq 200$
- $1 \leq n \leq 10^9$

```python

def flipped(g):
    grid = g.copy()
    m = len(grid)
    n = len(grid[0])
    fgrid = []
    for i in range(m):
        raw = ''
        for j in range(n):
            if grid[i][j] == 'O' \
            or grid[i][j] == '*' \
            or (i-1 >=0 and grid[i-1][j] == 'O')    \
            or (i+1 < m and grid[i+1][j] == 'O')    \
            or (j-1 >=0 and grid[i][j-1] == 'O')    \
            or (j+1 < n and grid[i][j+1] == 'O')    :
                raw += '*'
            else:
                raw += '.'
        fgrid.append(raw)
    dgrid = []
    for i in range(m):
        raw = ''
        for j in range(n):
            raw += '.' if fgrid[i][j] == '*' else 'O'
        dgrid.append(raw)
    return dgrid

def plant_all(g):
    grid = g.copy()
    m = len(grid)
    n = len(grid[0])
    for i in range(m):
        grid[i] = 'O'*n
    return grid

def bomberMan(n, grid):
    if (n==1) : return grid
    if (n-1) % 4 == 0: return flipped(flipped(grid))
    if n % 2 ==0 : return plant_all(grid)
    return flipped(grid)
```

Line: 16 Col: 1

Congratulations

You solved this challenge. Would you like to challenge your friends?

Compiler Message

```
Success
```

Input (stdin)