2048!

2048 is a very programmer-friendly game. It is played on a 4x4 matrix board. Each tile of the board is either empty, or contains a positive integer that is a power of two. (You know how programmers love their powers of 2! ;D)

The game progresses in turns. Each turn, one of four directions is chosen by the player. (Up, Left, Right, or Down) When a direction is selected, all non-empty tiles on the board shift in the chosen direction. All legal moves change the value of the board. When tiles shift, there can be a few possible outcomes:

1. If the tile is at the edge of the board for the chosen direction, it does not move.
2. If a tile slides into another tile with any other value than its own, it stops just like it would if it reached the end of the board.
3. If the tile slides into another tile with the same value, the two tiles are combined into a single tile with their combined value (2 & 2 becomes 4, 4 & 4 becomes 8 etc.)

If you would like to try out the game yourself, the original can be played here.

Let's look at a few examples of moves (with #'s representing empty tiles):

```
Right is selected:       Down is selected:
 #2## => ###2             #2## => ####
 4#4#    ###8             4#4#    ####
 48##    ##48             48##    #2##
 ####    ####             ####    884#

 Up is selected:         Left is selected:
 #2## => 824#             #2## => 2###
 4#4#    #8##             4#4#    8###
 48##    ####             48##    48##
 ####    ####             ####    ####
```

Okay, so that's simple. So what makes a move legal? Simply put, if a chosen direction would not modify the board, that is not a legal move! In other words, if all non-empty tiles on the board match case #1 OR case #2 for a given direction that move is not legal. Example:

```
Moving right is not legal for these boards!
(Because shifting right would not change the board at all)
###2  ###2
###2  ##24
###2  #248
###2  ####
```

The game ends when the board is full *AND* there are no legal moves remaining. (A full board where tiles can be collapsed is not the end of the game)

One interesting edge case to consider is a row or column with at least 3 equal values. How does this row collapse? The answer is, it collapses on the side closer to the direction the tiles are sliding *first*. An example is shown below. Notice that if there are four equal values, the two collapsed tiles don't collapse a second time in the same move. That action would require another move.

```
Right is selected:      Down is selected:
4444 => ##88            4444 => ####
2444    #248            2444    #4##
#444    ##48            #444    4888
2244    ##48            2244    4288
```

Your task will be to implement a function that accepts:

- A two dimensional array of integers (representing a game board) where empty tiles are represented as 0's.
- A direction (represented as pre-defined integers)

and calculates the next board position.

You must return the new board state from the updateBoard method. (Your function can modify the initial board value as much as you like, but the test suite will validate what you return, so don't forget to return it!)

------------------HAVE FUN!-----------------------

NOTES: For output simplicity, the basic tests handle only single digit numbers, but your solution should be able to handle up to any size (at least up to 2048, after all!)

Here I have represented empty tiles as #'s for simplicity. You should represent them in code as a 0.

If the provided board and direction is not a legal move (all tiles meet case 1 or 2 above) then the move does not affect the game board, and the input should be returned.

The result of a combination cannot match with another tile on the same move. In other words, `2222 =>` collapses to `##44` and *not* `###8`

Provided for you is a debugging function `printBoard(board)` that, given a game board array, displays that board to the output window.

In the real 2048, after each move, a 2 tile is added to a random empty space. For this kata we will not concern ourselves with this portion of the game rules because the kata would not be able to be accurately tested.

You can assume all input passed to your function will be correct. (You will receive a 2 dimensional array, always 4x4 and your direction will always be 0,1,2, or 3 as it is defined)