Design Tic-Tac-Toe

Design a Tic-tac-toe game that is played between two players on a n x n grid.

You may assume the following rules:

1. A move is guaranteed to be valid and is placed on an empty block.
2. Once a winning condition is reached, no more moves is allowed.
3. A player who succeeds in placing n of their marks in a horizontal, vertical, or diagonal row wins the game.

Example:

```
Given n = 3, assume that player 1 is "X" and player 2 is "O" in the board.

TicTacToe toe = new TicTacToe(3);

toe.move(0, 0, 1); -> Returns 0 (no one wins)
|X| | |
| | | |    // Player 1 makes a move at (0, 0).
| | | |

toe.move(0, 2, 2); -> Returns 0 (no one wins)
|X| |O|
| | | |    // Player 2 makes a move at (0, 2).
| | | |

toe.move(2, 2, 1); -> Returns 0 (no one wins)
|X| |O|
| | | |    // Player 1 makes a move at (2, 2).
| | |X|

toe.move(1, 1, 2); -> Returns 0 (no one wins)
|X| |O|
| |O| |    // Player 2 makes a move at (1, 1).
| | |X|

toe.move(2, 0, 1); -> Returns 0 (no one wins)
|X| |O|
| |O| |    // Player 1 makes a move at (2, 0).
|X| |X|

toe.move(1, 0, 2); -> Returns 0 (no one wins)
|X| |O|
|O|O| |    // Player 2 makes a move at (1, 0).
|X| |X|

toe.move(2, 1, 1); -> Returns 1 (player 1 wins)
|X| |O|
|O|O| |    // Player 1 makes a move at (2, 1).
|X|X|X|
```

Follow up:
Could you do better than O(n^2) per `move()` operation?

1. Could you trade extra space such that `move()` operation can be done in O(1)?

2. You need two arrays: int rows[n], int cols[n], plus two variables: diagonal, anti_diagonal.

Solution 1

Initially, I had not read the Hint in the question and came up with an O(n) solution. After reading the extremely helpful hint; a much easier approach became apparent. The key observation is that in order to win Tic-Tac-Toe you must have the entire row or column. Thus, we don't need to keep track of an entire n^2 board. We only need to keep a count for each row and column. If at any time a row or column matches the size of the board then that player has won.

To keep track of which player, I add one for Player1 and -1 for Player2. There are two additional variables to keep track of the count of the diagonals. Each time a player places a piece we just need to check the count of that row, column, diagonal and anti-diagonal.

Also see a very similar answer that I believe had beaten me to the punch. We came up with our solutions independently but they are very similar in principle. Aeonaxx's soln

```java
public class TicTacToe {
private int[] rows;
private int[] cols;
private int diagonal;
private int antiDiagonal;

/** Initialize your data structure here. */
public TicTacToe(int n) {
    rows = new int[n];
    cols = new int[n];
}

/** Player {player} makes a move at ({row}, {col}).
    @param row The row of the board.
    @param col The column of the board.
    @param player The player, can be either 1 or 2.
    @return The current winning condition, can be either:
            0: No one wins.
            1: Player 1 wins.
            2: Player 2 wins. */
public int move(int row, int col, int player) {
    int toAdd = player == 1 ? 1 : -1;

    rows[row] += toAdd;
    cols[col] += toAdd;
    if (row == col)
    {
        diagonal += toAdd;
    }

    if (col == (cols.length - row - 1))
    {
        antiDiagonal += toAdd;
    }

    int size = rows.length;
    if (Math.abs(rows[row]) == size ||
        Math.abs(cols[col]) == size ||
        Math.abs(diagonal) == size  ||
        Math.abs(antiDiagonal) == size)
    {
        return player;
    }

    return 0;
}
```

}

written by bdwalker original link here

Solution 2

new version:

```
class TicTacToe {
private:
    //count parameter: player 1 + : player 2: -
    vector<int> rowJudge;
    vector<int> colJudge;
    int diag, anti;
    int total;
public:
    /** Initialize your data structure here. */

    TicTacToe(int n):total(n), rowJudge(n), colJudge(n),diag(0),anti(0){}

    int move(int row, int col, int player) {
        int add = player == 1 ? 1 : -1;
        diag += row == col ? add : 0;
        anti += row == total - col - 1 ? add : 0;
        rowJudge[row] += add;
        colJudge[col] += add;
        if(abs(rowJudge[row]) == total || abs(colJudge[col]) == total || abs(diag)
== total || abs(anti) == total)
            return player;
        return 0;
    }
};
```

old version:

```
class TicTacToe {
private:
    //status:
    // 0: no one fill
    // 1 or 2: player fill
    //-1 : invalid
    //pair:
    //first:player, second:count
    vector<pair<int,int>> rowJudge;
    vector<pair<int,int>> colJudge;
    pair<int,int> diag, anti;
    int total;
public:
    /** Initialize your data structure here. */
    TicTacToe(int n):total(n), rowJudge(n), colJudge(n){}

    /** Player {player} makes a move at ({row}, {col}).
        @param row The row of the board.
        @param col The column of the board.
        @param player The player, can be either 1 or 2.
        @return The current winning condition, can be either:
                0: No one wins.
                1: Player 1 wins.
                2: Player 2 wins. */
```

```
                2: Player 2 wins. */
    int move(int row, int col, int player) {
        if(rowJudge[row].first == 0 || rowJudge[row].first == player){
            rowJudge[row].first = player;
            rowJudge[row].second++;
            if(rowJudge[row].second == total){
                return player;
            }
        }
        else {
            rowJudge[row].first = -1;
        }

        if(colJudge[col].first == 0 || colJudge[col].first == player){
            colJudge[col].first = player;
            colJudge[col].second++;
            if(colJudge[col].second == total){
                return player;
            }
        }
        else {
            colJudge[col].first = -1;
        }


        if(row == col){
            if(diag.first == 0 || diag.first == player){
                diag.first = player;
                diag.second++;
                if(diag.second == total){
                    return player;
                }
            }
            else{
                diag.first = -1;
            }
        }
        if(row + col == total - 1){
            if(anti.first == 0 || anti.first == player){
                anti.first = player;
                anti.second++;
                if(anti.second == total){
                    return player;
                }
            }
            else{
                anti.first = -1;
            }
        }
        return 0;
    }
};
```

written by sxycwzwzq original link here

Solution 3

```
public class TicTacToe {

    private int[] rows;
    private int[] cols;
    private int size;
    private int diagonal;
    private int anti_diagonal;
    /** Initialize your data structure here. */
    public TicTacToe(int n) {
        size = n;
        rows = new int[n];
        cols = new int[n];
    }

    /** Player {player} makes a move at ({row}, {col}).
        @param row The row of the board.
        @param col The column of the board.
        @param player The player, can be either 1 or 2.
        @return The current winning condition, can be either:
                0: No one wins.
                1: Player 1 wins.
                2: Player 2 wins. */
    public int move(int row, int col, int player) {
        int add = player == 1 ? 1 : -1;
        if(col == row){
            diagonal += add;
        }
        if(col == size - 1 - row){
            anti_diagonal += add;
        }
        rows[row] += add;
        cols[col] += add;
        if(Math.abs(rows[row]) == size || Math.abs(cols[col]) == size|| Math.abs(
diagonal) == size || Math.abs(anti_diagonal) == size){
            return player;
        }
        return 0;
    }
}
```

written by lei11 original link here