Conway's Game of Life

The programming challenge explain in https://github.com/careernowbrands/full-stack-engineer/blob/master/challenges/coding-challenge-1.md is defined as next:

> *The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead, or "populated" or "unpopulated". Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:*
>
> 1. *Any live cell with fewer than two live neighbours dies, as if caused by underpopulation.*
> 2. *Any live cell with two or three live neighbours lives on to the next generation.*
> 3. *Any live cell with more than three live neighbours dies, as if by overpopulation.*
> 4. *Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.*
>
> *The initial pattern constitutes the seed of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed—births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a tick (in other words, each generation is a pure function of the preceding one). The rules continue to be applied repeatedly to create further generations.*

1. The Grid

   The grid is our initial config, here is where we can know what are the *neighbours* for any cell and what is its position.

   For instance, a grid 7 x 5

1	**2**	**3**	**4**	**5**	**6**	**7**
8	**9**	**10**	**11**	**12**	**13**	**14**
15	**16**	**17**	**18**	**19**	**20**	**21**
22	**23**	**24**	**25**	**26**	**27**	**28**
29	**30**	**31**	**32**	**33**	**34**	**35**

If x = 7 and y = 5, and there are a couple of fors like

```
for ( j = 0; j < y; j++ ) {
        for ( i = 1; i <= x; i++ ) {
                (j * x) + i
        }
}
```

1	2	3	4	5	6	7	(j * x) + i
8	9	10	11	12	13	14	(j * x) + i
15	16	17	18	19	20	21	(j * x) + i
22	23	24	25	26	27	28	(j * x) + i
29	30	31	32	33	34	35	(j * x) + i

Also, according to the x and y values can be change the position

7 X 5

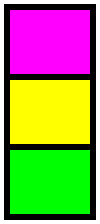
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35

5 x 5

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

For instance, in the 7x7 grid the 7 cell has just 3 neighbours but in the 5x5 table the 7 cell has 8 neighbours. The positions could have different amounts of neighbours

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35



Corners (3 neighbours)

Borders (4 neighbours)

Internals (8 neighbours)

Also, to identify who are the neighbours for each cell is necessary have a rules, or do a difference between the positions.

For the internals cell the neighbours and their positions are:

tl	**t**	**tr**
nl	**n**	**nr**
bl	**b**	**br**

In this case we can find the positions easily, these are:

n = Any internal position in the grid (Is the position base**)**
nl = n-1 (left neighbour**)**
nr = n+1 (right neighbour**)**
t = n-x (top neighbour**)**
tl = t-1 (top left neighbour**)**
tr = t+1 (top right neighbour**)**
b = n+x (bottom neighbour**)**
bl = b-1 (bottom left neighbour**)**
br = b+1 (bottom right neighbour**)**

For example, in the 7 x 5 grid, the neighbours of the position 17 according to the above definitions

x = 7, **y** = 5, **n** = 17
nl = 17 - 1 = 16
nr = 17 + 1 = 18
t = 17 - 7 = 10
tl = 10 - 1 = 9
tr = 10 + 1 = 11
b = 17 + 7 = 24
bl = 24 - 1 = 23
br = 24 + 1 = 25

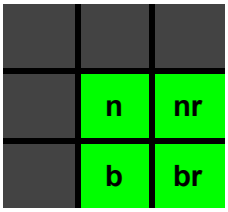
9	**10**	**11**
16	**17**	**18**
23	**24**	**25**

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35

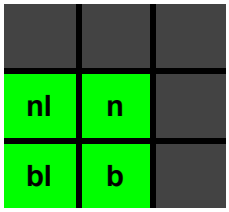
For the corners is different because depend of the corners the neighbours would have different positions. There are four corners positions and the numbers are

top-left: tl = 1
top-right : tr = x
bottom-left: bl = (x * y) - (x - 1)
bottom-right: br = (x * y)

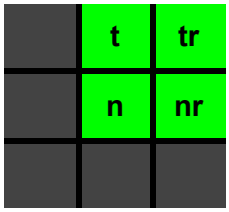
For each corner there are different neighbours.



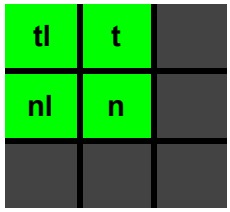
tl



tr

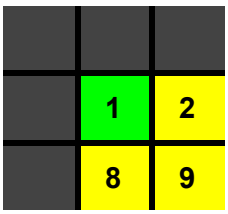


bl

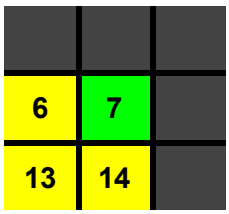


br

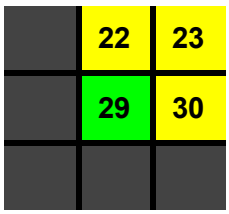
For the 7x5 example .



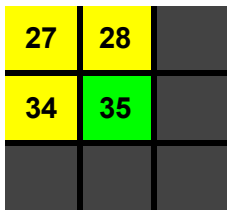
tl



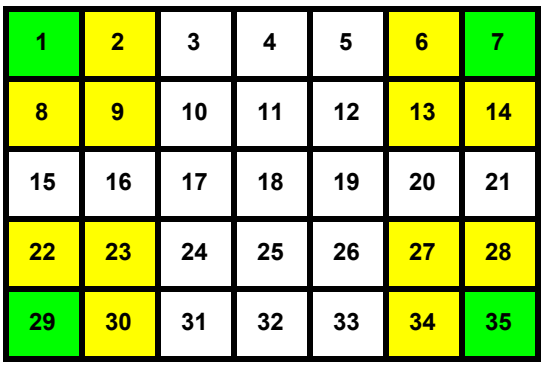
tr



bl

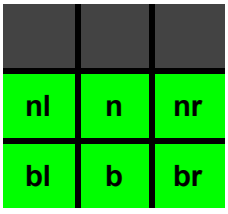


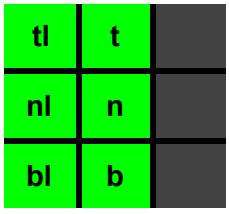
br

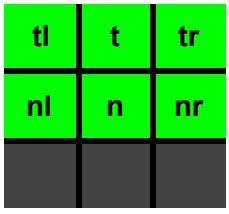


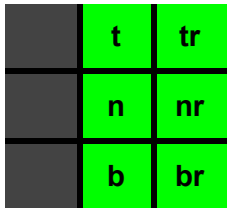
And, there are other positions for the borders. There are 4 kind of borders

top-border: tb = positions between top-left (1) and top-right (x) -> (1 < i < x)
right-border: rb = positions between x and x*y where i % x = 0 -> (1 < i < x*y)
bottom-border: bb = position between bottom-left and bottom-right
left-border: positions between x and (x*y)-(x-1) where i % x = 1 -> (1 < i < bl)





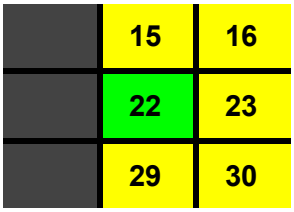




tb rb bb lb

Let's do a example with a position in the left-border. The position 22 for a 7x5 grid

n = 22; x = 7; y = 5;
n % x = 22 % 7 = 1; // It is in the left border
(x*y)-(x-1) = (7 * 5) - (7 - 1) = 35 - 6 = 29;
1 < 22 < 29; // 22 is a left border



1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35

In total there are 9 different kinds of positions. 4 corners, 4 borders and the internal cells.

Corners:

top-left: **tl** = 1

top-right : **tr** = x

bottom-left: **bl** = (x * y) - (x - 1)

bottom-right: **br** = (x * y)

Borders:

top-border: **tb** = positions between top-left and top-right -> (tl < i < tr)

right-border: **rb** = positions between tr and br where i % x = 0 -> (tr < i < br)

bottom-border: **bb** = position between bottom-left and bottom-right -> (bl < i < br)

left-border: **lb** = positions between tl and bl where i % x = 1 -> (tl < i < bl)

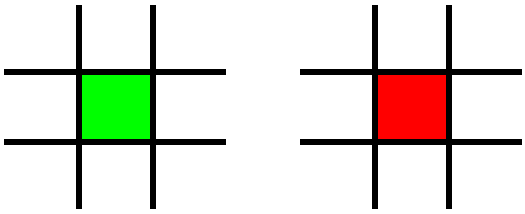
Internals: **in** = All positions different to corners or borders

According to the analysis the grid has positions and in each position there would be a cell with a specific features.

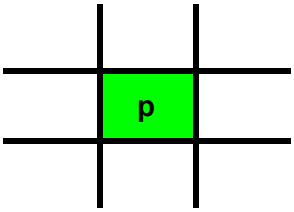
2. The Cell

The cell is a object with a few attributes

- Status (It could be alive or dead)



- Position (The position in the grid)



- Neighbours (Must be a array, It could be 3, 5 or 8 elements, accorder to the position kind)