# Life at the edges

We all have the "board index" reporter block working, which translates a 2-D coordinate into a place in our 1-D list.

In the Game of Life, the rules for each cell depend on counting the number of live cells around it. Consider a cell at row **R**, column **C**. Then we need to know if the eight cells around it are alive or dead:

| | | |
|---|---|---|
| R - 1, C - 1 | R – 1, C | R – 1, C + 1 |
| R, C – 1 | **R, C** | R, C + 1 |
| R + 1, C – 1 | R + 1, C | R + 1. C + 1 |

This is pretty easy: to count the number of live cells around us, we can create a "sum" variable and add to the sum every time we see a live cell. In fact, because we store our dead cells as 0s, we can just add the values of our neighbors together since adding zero doesn't change our sum,

Seems like we could just do this:

(board index (r – 1) (c – 1)) + (board index (r – 1) (c)) + (board index (r – 1) (c + 1)) + …

That's great, **except** when the cell we care about is on the border. If our cell is at row 1, column 1, then it only has 3 neighbors instead of 8. Any cell n row 1 is missing at least three neighbors above it.

How should we deal with this? The game authors proposed two solutions:

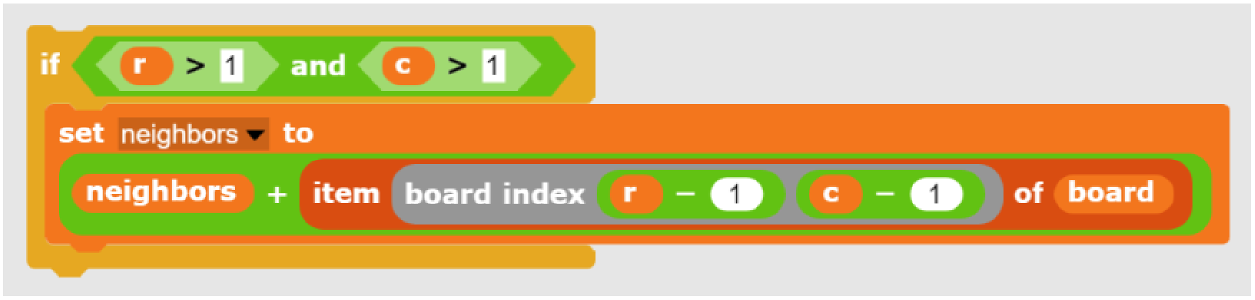
1. Any "neighbors" not on the board are considered "dead"
2. OR wrap around, so that the left neighbors of the first column are actually the last column.

I suggest you stick with #1 for convenience. If you feel like you're an advanced student, pick #2.

How do you make this work in SNAP? There are many possible ways to account for this. I'll propose two ways, and you can choose one.

## Idea 1: use eight "if" blocks

You need to guard against the border cases. You can do this with "if" blocks. For example, the following block tests if the upper left diagonal neighbor is on the board, and if so, adds its value.

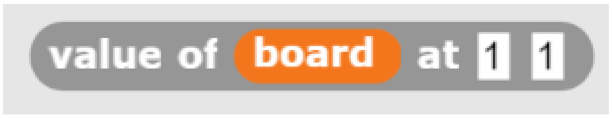


```
if <(r) > 1> and <(c) > 1>
  set neighbors to (neighbors) + (item (board index ((r) − 1) ((c) − 1)) of (board))
```

You can see that this code will get a little bit hairy because you'll need to make sure that your "if" statements and the values passed in to "board index" are correct. There will be eight of these statements. Advanced students will note that you can reduce the number of "if" statements.

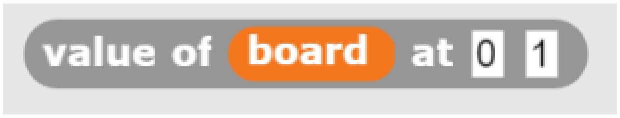
## Idea 2: use a "helper block"

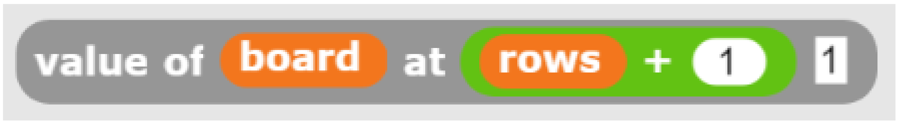
If you don't like writing these "if" blocks over and over, consider using a "helper block." Since we are only adding ones and zeroes together, imagine if we made a block that gave us the board value at a particular row and column. This block also reports zero if the row or column are out of bounds.



This returns the value of the board at (1, 1). It uses the "board index" reporter to do this.

But it also knows to return zero if the parameters are less than 1 or bigger than the #rows or #columns. For example, both of the following report 0 because the row parameter is out of range.





So how do you use this? Basically, neighbor count becomes a giant sum of calling (value of (board) at (r, c)) eight times. It relies on value of … returning zero if it receives "bad" parameters.

If you choose to go down this way, make sure you test your "value of…" reporter for positions that are on the board and off the board.