

# 2048 Instructions

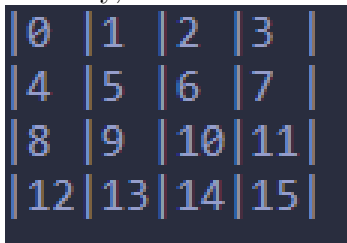
Dante

July 29, 2019

## 1 Intro

We are using a list of 16 numbers to map the 4x4 grid  
the 0th number of the list is the one at the top left corner  
the 1st number of the list is the one to the right of the 0th number  
...

and the 15th number is the one at the bottom right corner  
Visually, it looks something like the following:



0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

## 2 Part 1

Moving up/down/left/right uses the same algorithm, just in different directions.

Furthermore, when moving down for example, the columns are independent from each other.

So, instead of writing a function that moves all 4 columns/rows at once, we break it down into a function that only moves 1 row/col.

Let's analyse the process of moving 1 column DOWN.  
 Consider the following examples(0 would be an empty block in the actual game):

0	->	0		0	->	0		2	->	0
0		0		2		0		2		0
2		0		0		0		0		0
2		4		2		4		0		4

Apparently, different arrangements of two 0s and two 2s all yield the same outcome, which is one 4 at the bottom, and three 0s on top of the 4.  
 Here is another set of examples:

0	->	0		2	->	0		2	->	0		2	->	0
2		2		0		2		4		2		4		2
4		4		4		4		0		4		8		4
8		8		8		8		8		8		0		8

If you change the order of 2, 4 and 8, the outcome would be different.  
 As long as the order of non-zero numbers stay the same, the outcome would stay the same. What we can conclude from this is that the positions of 0s do NOT matter. So when we implement the "move" function, it would be ideal to NOT consider the number 0 whatsoever.

\* You can use as many helper functions as you need

*let move lst =*

Use the last 3 functions as helper functions to implement this function,  
 so that given a list that looks something like [4;4;8;2], return [8;8;2;0]

*let c\_s lst =*

Use some of the functions you have implemented as helper functions to implement this function,  
 so that given a list that looks something like [4;4;8;2], return 8(only the two 4s combined, so 8 points)

*let step { points = p; cells = lst } dir =*

given the current level and the direction, return the next level after moving in the given direction once.

update the score(points) and the numbers in the grid(cells)

### 3 Part 2

Everytime a successful move is made, a new number is added randomly to an empty spot in the grid

You want to randomly generate a number that is either 2 or 4, and put it in a random empty spot.

*let rec insert\_rand lst n =*

given the list of all 16 numbers, replace the nth 0 with a random new number

### 4 Part 3

connect the algorithms to the GUI

*let end\_state { points=p; cells = cs } =*

technically there is no win or lose state

the game ends when you cannot make any moves

*let match\_grid lst coor =*

match each number in the list to a coordinate