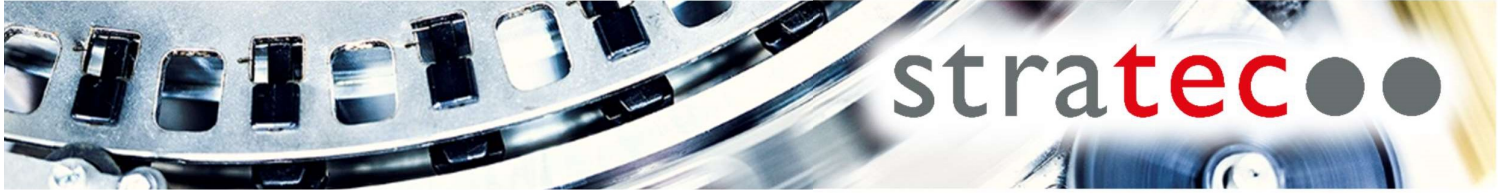


Build Your Robot 2019

The Software Challenge

Before delving into the software challenge, it's worth mentioning a few guidelines regarding the participants' submissions.

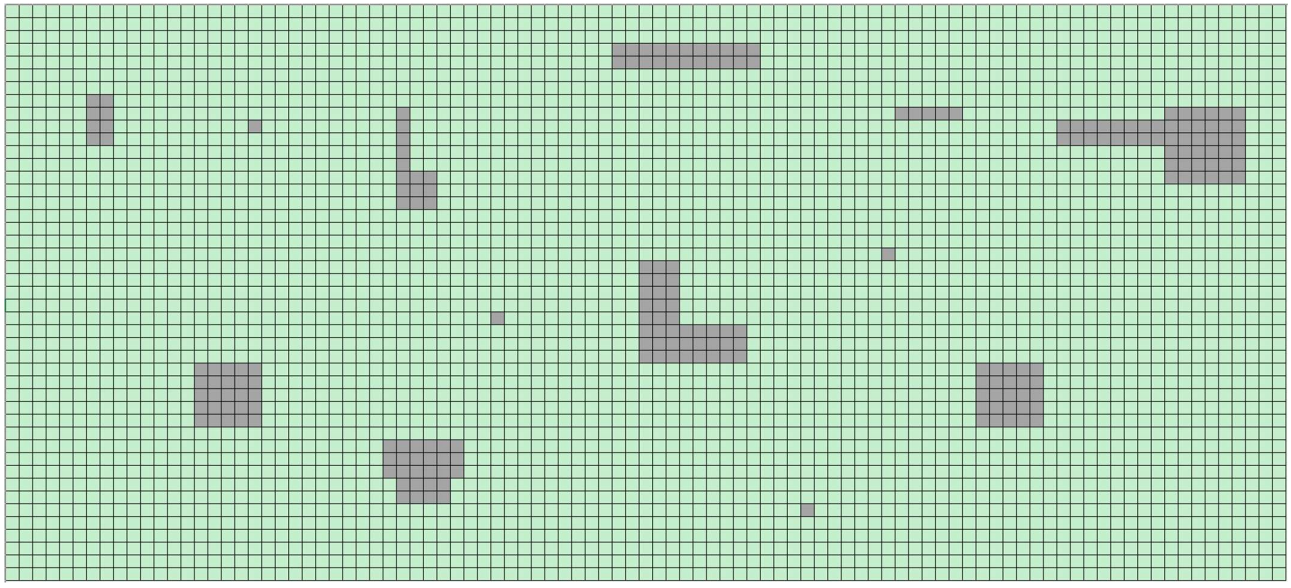
First off, the solution to the problem may be implemented in any programming language that is deemed fitting by the contestant.

Secondly, the submission must contain, as a bare minimum, the complete code that you wrote to handle the given problem, presented in any of the common formats in which computer code is shared. If you choose to send us the complete solution or application as well, it's a welcome addition.

Lastly, the way in which you tackle the problem is entirely up to you. You may use whichever algorithm or resource you wish, as long as it's necessary to your approach in solving the challenge. Ideally, any external packages used will be contained in your submission. Third party frameworks or packages are accepted, and the only limitation placed on external resources is this: running your code must not require a network connection.

Stating The Problem

You are given a grid. The grid, or matrix, looks like this:



This grid is littered with contiguous entities (objects) of varying shapes and sizes.

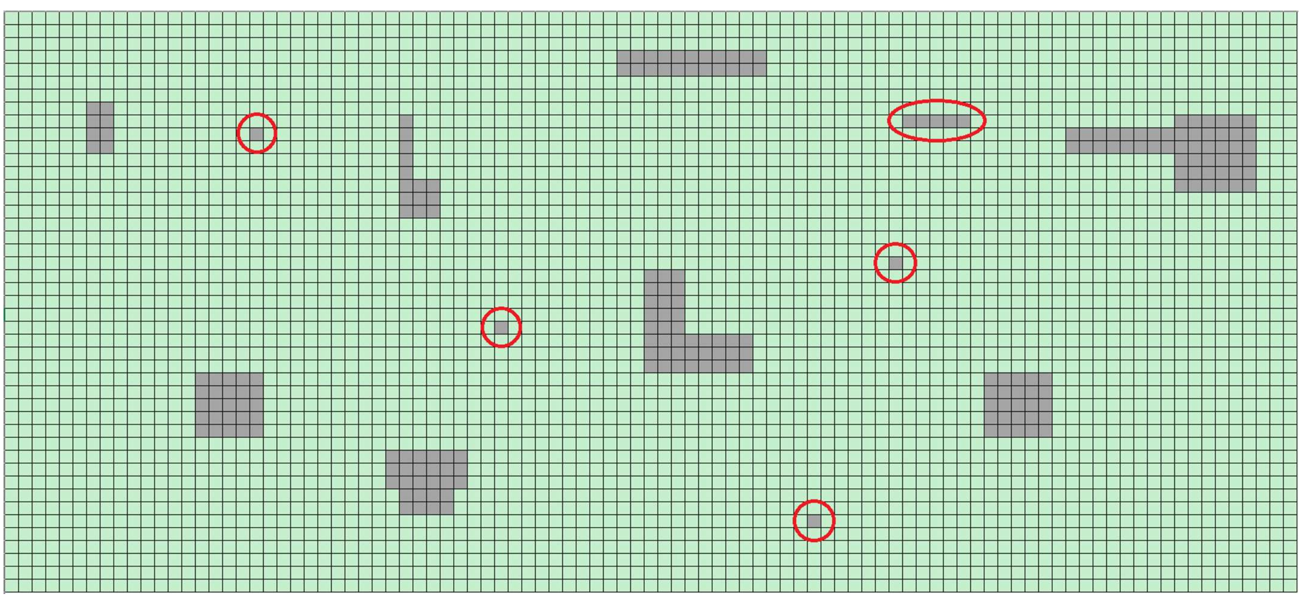
Upon closer inspection, these objects are represented by collections of individual cells. On this grid, the difference between an empty cell and a populated cell is binary. An empty (or green, in this case) cell contains a value of 0. A full (or gray) cell contains a value of 1.

The image below represents a zoomed-in area around the object in the upper-left of the grid.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This binary distinction is important, as the input for the code that you write will consist of CSV (comma-separated values) files exported from this grid.

As can be seen in the overview picture, the grid contains objects that are one-dimensional (the line in the upper-right of the grid) or made up of only one cell. These objects are to be considered noise and will be ignored when solving the challenge. The image below has markers around the "noise" objects.



The main aspects of this challenge will revolve around locating the valid objects on the grid and, later on, classifying them.

The output of your code will consist (mostly) of simple text, either printed on a console window or written to a file.

There are five levels to this challenge, each level building on the previous one and increasing in difficulty.

Level 1

You are given the .csv file taken from the grid shown in the pictures above. This file is called "The_Basics.csv".

Using that as an input, your program must output the number of non-noise objects present on the grid.

In this case, the text output will simply read:

8

Level 2

Using the same input file as before, "The_Basics.csv", your code must print out the following information:

- On the first line, the number of given objects.
- On the following lines (one line for each set), the pairs of X and Y coordinates for the bounding boxes surrounding each of the objects, as well as the bounding boxes' dimensions.

The text output will look like this:

8

(5, 6) W: 4 H: 6

(13, 27) W: 7 H: 7

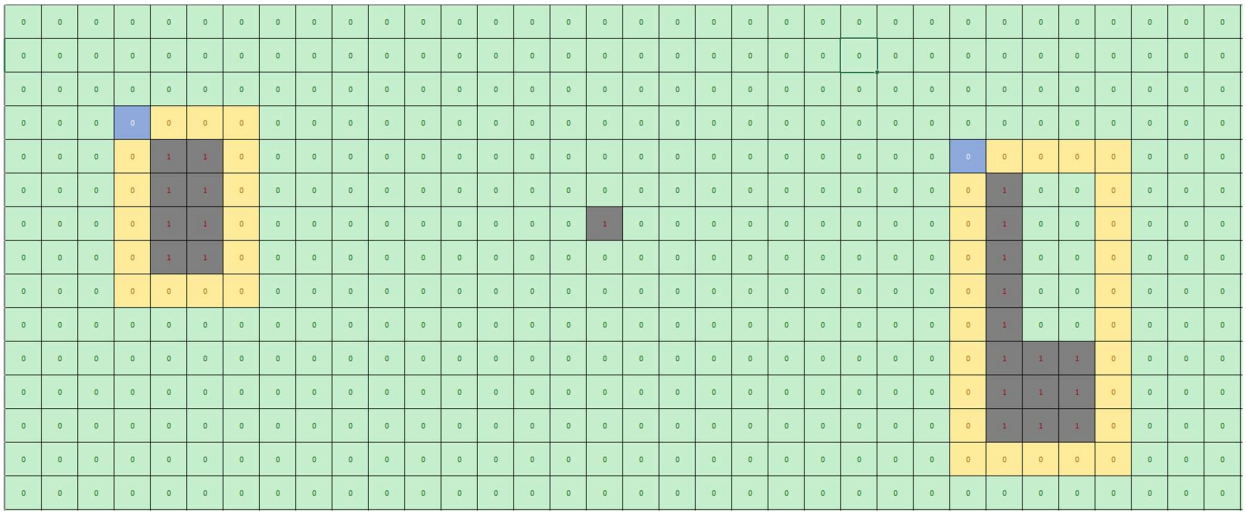
And so on…

The rules for this level are as follow:

- The uppermost, leftmost cell in the grid is considered to be the point of origin, coordinates (0, 0).
- The X axis is considered to be the left – right direction, and the Y axis is the up – down direction on the grid.
- The coordinates given for the bounding boxes consider their upper-left corners as the origins. It's those corners that are to be printed as the output coordinates.

The image below (a snapshot of the upper left corner of the grid) shows these aspects in greater detail.
The bounding boxes are shown in yellow, and their origin is shown in blue.
The boxes are rectangular and they completely encase the objects on the grid, whether or not they themselves are rectangular.

- The bounding boxes will have their widths measured on the X axis, and their heights measured on the Y axis.

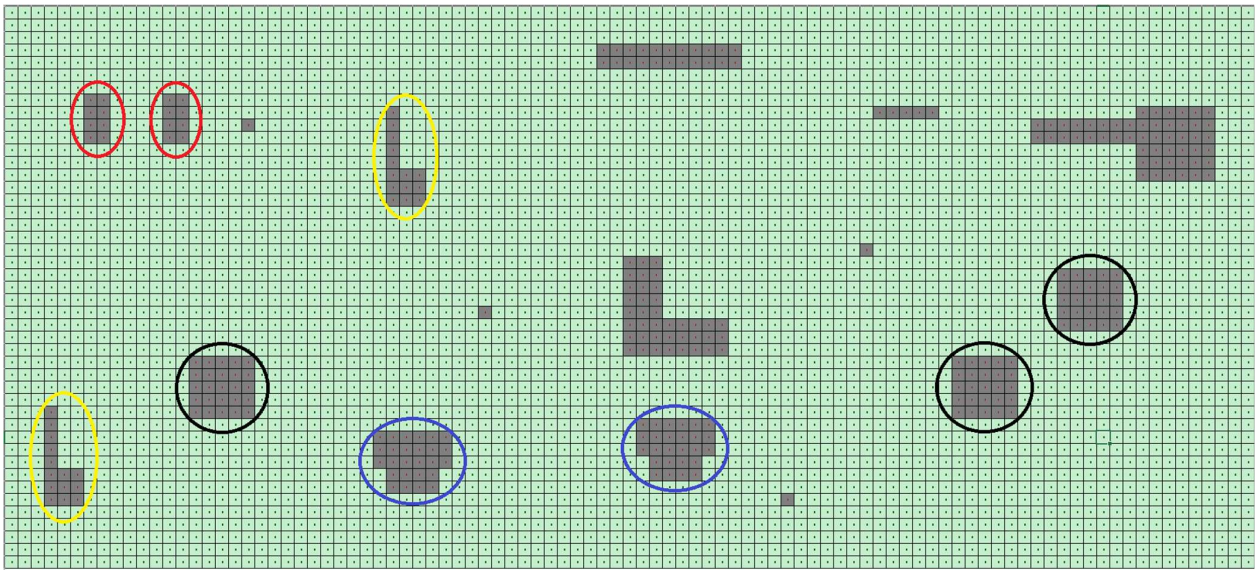


Level 3

This level uses a different input file, "Duplicates.csv".

Like in the previous level, the goal is to pinpoint the origins and dimensions of the bounding boxes surrounding the objects.

This time, however, some of the objects are duplicated:



For this level, the output of your code will be similar to the one in the level above, with the first line showing the total number of objects, and the following lines showing the coordinates and dimensions of the objects' bounding boxes. This time, however, the duplicated objects will be grouped onto the same line:

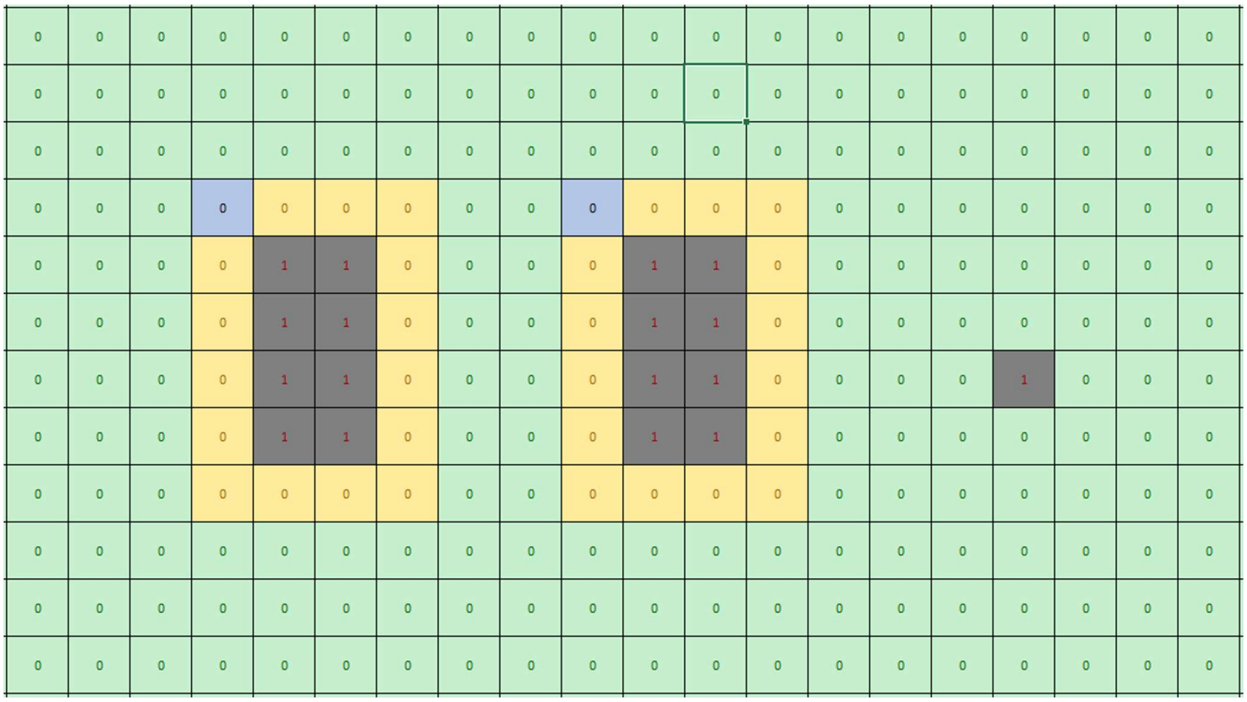
8

(5, 6) W: 4 H: 6 – this object is also found at (11, 6)

(13, 27) W: 7 H: 7 - this object is also found at… and at…

And so on…

The image below shows this in more detail:



Level 4

The input file for this level is called "Duplicates_Advanced.csv".

The output for this level is almost identical to the output of the previous level, and the proposition is very similar.

The input data for this level contains a grid with objects, some duplicated. This time, however, some of the duplicated objects are also rotated relative to each other. Your code must, nonetheless, recognize the duplicates. The output will look like this:

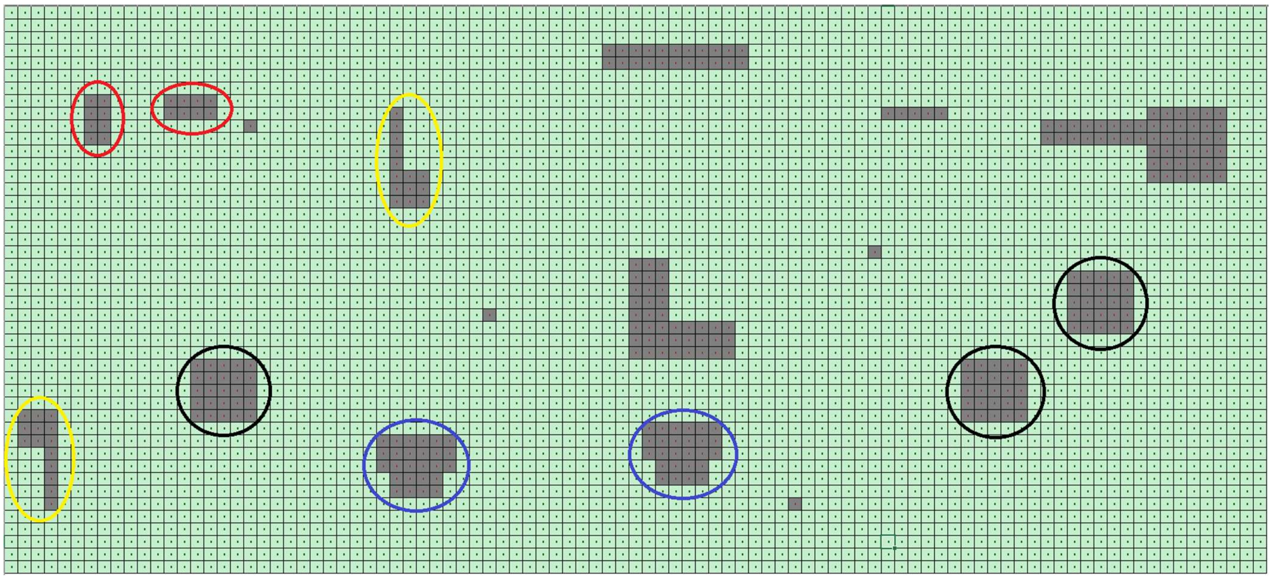
8

(5, 6) W: 4 H: 6 – this object is also found at (11, 6), rotated by 90 degrees

(13, 27) W: 7 H: 7 – this object is also found at…

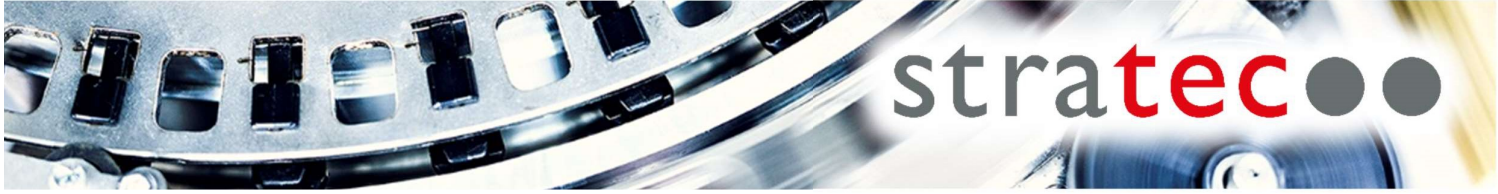
And so on…

This overview image of the grid shows us that two of the four duplicates have been rotated in a clockwise direction by 90 and 180 degrees, respectively. For simplicity's sake, the rotation between objects will only be done in the clockwise direction, by multiples of 90 degrees (0, 90, 180, 270). Furthermore, perfectly square objects may be excluded from the rotation measurements, as they will look the same in any of the possible orientations.



The following image describes the bounding box situation for the upper-left corner of the grid:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Level 5

The bonus level.

For this level, all you have to do is design a user interface capable of displaying all the information you consider to be relevant when importing the input data and solving the previous four levels' challenges.

Good luck!