


# Ryadel

Algorithm and JavaScript function for Dobble (Spot it!) game  
A JavaScript function to generate combinatorial geometric series for games such as Dobble (Spot it!) and to better understand the math logic behind them

---

 [ryadel.com/en/dobble-spot-it-algorithm-math-function-javascript](https://ryadel.com/en/dobble-spot-it-algorithm-math-function-javascript)

Ryan

August 17,  
2019



Have you ever heard of **Dobble**? If you have children, I bet you did! Anyway, it's a *pattern recognition* board game for children (but also great for all ages) where the players must identify common shapes across multiple cards as quickly as possible.

The game has 55 cards, each of which presents 8 different shapes arranged in random order. The game is made so that each pair of cards has one (and only one) shape in common, which players must find as soon as possible. The game was released in 2009 under the name **Spot it!**, while **Dobble** is an alternative market name mostly used in European countries (including Italy, where it was published by Asmodee around 2013). For more information about this game, we strongly advise to take a look at its page on the [BoardGameGeek community website](#).



The **dobble()** function in the JavaScript code below can be used to generate the maximum number of series (cards) with a given **N** number of elements: however, the algorithm will only work if **N** has a value of any prime number +1. It could be 8 (as in *Dobble*) or also 3, 4, 6, 12, 14, 18, 20, and so on. If an invalid **N** value is used, the function will show a warning – and the **outputTests()** testing function will have some failures.

It goes without saying that each number contained in the various series is meant to represent a unique *shape* among those present on our cards: the number 1 for the *heart*, the number 2 for the *tree*, and so on. For the sake of simplicity, we won't make the numbers-to-shapes mapping in this post, being it a rather trivial task.

```
// -----
// JavaScript function to generate combinatorial geometric series
// for board games like Dobble / Spot it!
//
// Each generated series is expected to have the following features:
//
// - a given number (N) of different elements, as long as N is a prime number +1
// - one (and only one) element in common with each other series;
//
// Released on GNU - General Public Licence v3.0
// https://www.gnu.org/licenses/gpl-3.0.en.html
//
// Darkseal, 2018-2019
// https://www.ryadel.com/en/dobble-spot-it-algorithm-math-function-javascript
// -----
//
function dobble() {
    var N = 12;    // number of symbols on each card
    var nC = 0;    // progressive number of cards
    var sTot = []; // array of series (cards)

    // check if N is valid (it must be a prime number +1)
    if (!isPrime(N-1)) {
        document.write("<pre>ERROR: N value (" + N + ") is not a prime number +1:");
        document.write(" some tests will fail.</pre>");
    }

    // Generate series from #01 to #N
    for (i=0; i <= N-1; i++) {
        var s = [];
        nC++;
        s.push(1);
        for (i2=1; i2 <= N-1; i2++) {
            s.push((N-1) + (N-1) * (i-1) + (i2+1));
        }
        sTot.push(s);
    }

    // Generate series from #N+1 to #N+(N-1)*(N-1)
    for (i= 1; i<= N-1; i++) {
        for (i2=1; i2 <= N-1; i2++) {
            var s = [];
```

```

    nC++;
    s.push(i+1);
    for (i3=1; i3<= N-1; i3++) {
        s.push((N+1) + (N-1) * (i3-1) + ( ((i-1) * (i3-1) + (i2-1)) ) % (N-1));
    }
    sTot.push(s);
}
}

// Print the series to screen
outputSeries(sTot);

// perform 1000 test and print the results to screen
outputTest(sTot, 1000);
}

function isPrime(num) {
    for(var i = 2; i < num; i++) {
        if(num % i === 0) return false;
    }
    return num > 1;
}

function pad(n, width, z) {
    z = z || '0';
    n = n + '';
    return n.length >= width ? n : new Array(width - n.length + 1).join(z) + n;
}

function outputSeries(sTot) {
    var nPad = sTot.length.toString().length;
    var cnt = 0;
    document.write("<div>Printing "+ sTot.length + " series of "+ sTot[0].length + " elements each.
</div>");
    document.write("<pre>");
    for (var i in sTot) {
        cnt++;
        var sLog = "#" + pad(cnt,nPad) + ":";
        for (var i2 in sTot[i]) {
            sLog += " " + pad(sTot[i][i2], nPad);
        }
        document.write(sLog + "\n");
    }
    document.write("</pre>");
}

// test function
// compares n pairs of different series randomly taken from sTot
// and outputs the results.
function outputTest(sTot, n) {
    var nSucc = 0;
    var nFail = 0;
    var err = "";
    for (i = 0; i < n; i++) {

```

```

var i1 = Math.floor(Math.random() * (sTot.length - 1));
var i2 = 0;
do {
  i2 = Math.floor(Math.random() * (sTot.length - 1));
}
while (i1 == i2);
var s1 = sTot[i1];
var s2 = sTot[i2];
var nEquals = 0;
for (var p1 in s1) {
  for (var p2 in s2) {
    if (s1[p1] == s2[p2]) {
      nEquals++;
    }
  }
}
if (nEquals == 1) {
  nSucc++;
}
else {
  nFail++;
  err += "FAILURES #" + nFail + ": Series #" + s1 + " and series #" + s2 + " do have " + nEquals + "
numbers in common. +\n";
}
}
document.write("<pre>");
document.write("Test result (after " + n + " tests):\n");
document.write("- SUCCESS #: " + nSucc + "\n");
document.write("- FAILURE #: " + nFail + "\n");
if (nFail > 0) {
  document.write("<pre>" + err + "</pre>");
}
}

```

The version in this article might be obsolete: the most up-to-date build is available on [GitHub](#).

To try this function and/or change its values to generate different series, check out [this JSFiddle](#).

As we can see, the function performs two cycles: the first constructs a serial number equal to the number of elements (**N**) by placing the element number 1 in the first position and then filling them with the other elements in ascending order by following a *horizontal* progression (from left to right); such logic will make sure that all series produced so far will have the element number 1 (and only that one) in common. This operation, as can be seen from the image, also constructs a matrix that contains all the elements.

#01:	01	02	03	04	05	06	07	08
#02:	01	09	10	11	12	13	14	15
#03:	01	16	17	18	19	20	21	22
#04:	01	23	24	25	26	27	28	29
#05:	01	30	31	32	33	34	35	36
#06:	01	37	38	39	40	41	42	43
#07:	01	44	45	46	47	48	49	50
#08:	01	51	52	53	54	55	56	57

The second cycle builds all the remaining series by placing the elements number 2, 3, 4, 5 and so on (up to **N**) in the first position and then filling them with the other elements in ascending order: however, this time the progression develops *vertically* (from top to bottom) to be sure to “intersect” once (and only once) any series generated during the previous cycles.

#01:	01	02	03	04	05	06	07	08
#02:	01	09	10	11	12	13	14	15
#03:	01	16	17	18	19	20	21	22
#04:	01	23	24	25	26	27	28	29
#05:	01	30	31	32	33	34	35	36
#06:	01	37	38	39	40	41	42	43
#07:	01	44	45	46	47	48	49	50
#08:	01	51	52	53	54	55	56	57
#09:	02	09	16	23	30	37	44	51
#10:	02	10	17	24	31	38	45	52
#11:	02	11	18	25	32	39	46	53
#12:	02	12	19	26	33	40	47	54
#13:	02	13	20	27	34	41	48	55

Also, whenever the number in the first position changes, the vertical progression path is incrementally shifted to the right of the matrix starting from the number in position 3, so that each subsequent series will only “intersect” each vertical column of the matrix once (and only once).



By looking at how the above cycles work, we can easily understand how the maximum number of different series that this function can produce is equal to  $N + (N-1) * (N-1)$ . In the case of **Dobble**, where **N** is equal to 8, it means that we can have up to a maximum of  $8 + (8-1) * (8-1) = 57$  different cards: that's 2 more than the number of cards contained in the Italian edition! If the shapes on each card were 10, we could have up to 91; if they were 12, up to 133; and so on. Just remember that **N** must be a prime number +1, otherwise you'll end up with a certain amount of invalid series (that will be shown by the testing function output).

## Conclusion

---

That's about it: I hope that this article could feed the curiosity of those interested about the maths behind this beautiful game.

If you like this script, feel free to give me a virtual hug by starring the [GitHub project](#) and/or putting a like on our Facebook/Twitter pages!

