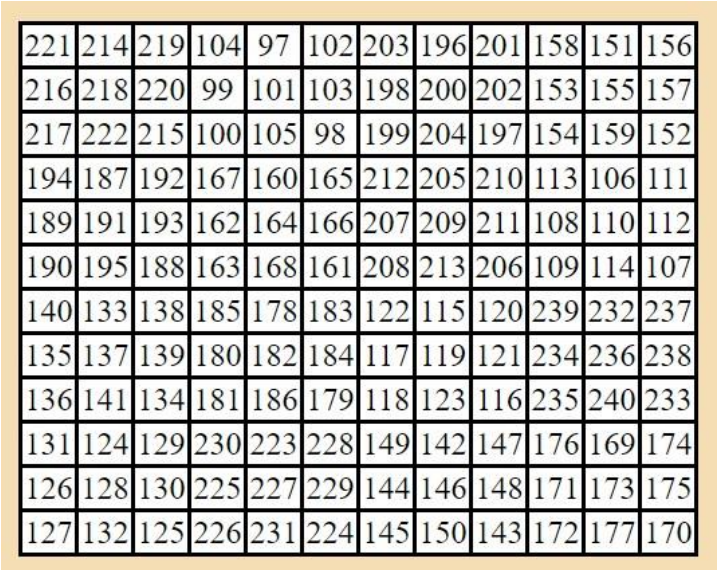
How to make ANY Magic Square



221	214	219	104	97	102	203	196	201	158	151	156
216	218	220	99	101	103	198	200	202	153	155	157
217	222	215	100	105	98	199	204	197	154	159	152
194	187	192	167	160	165	212	205	210	113	106	111
189	191	193	162	164	166	207	209	211	108	110	112
190	195	188	163	168	161	208	213	206	109	114	107
140	133	138	185	178	183	122	115	120	239	232	237
135	137	139	180	182	184	117	119	121	234	236	238
136	141	134	181	186	179	118	123	116	235	240	233
131	124	129	230	223	228	149	142	147	176	169	174
126	128	130	225	227	229	144	146	148	171	173	175
127	132	125	226	231	224	145	150	143	172	177	170

Judith Opdebeeck

How to make ANY Magic Square

Contents

Preface

At some point in the year 2013, I came across an episode of the show "Scam School", with magician Brian Brushwood. It would be the beginning of a journey that would last several years. If you haven't heard of it, Scam School (which is called Scam Nation these days) is a weekly show on YouTube, which teaches proposition bets, riddles, and magic tricks to use at the bar. In this episode, Brian taught a simple method to force the number 34, and a simple way to write down a magic square, which would add up to that number 34 in all sorts of ways.[1] Go watch the video, because I promise it is more impressive than I make it seem now.

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

It was a nice trick, but in my opinion it was too limited. This is because one is forced to use a 4 by 4 square, and is limited to the number 34. This is of course an essential part of how the trick works, and those limitations also keep the trick simple, which is why it's so good. But I couldn't help wondering if it

[1] BRUSHWOOD, B., [SCAM SCHOOL], *Look Like a GENUIS SAVANT! – An Easy, Convincing Magic Square Cheat*, video, 14 July 2010, (https://youtu.be/JKYUckzfnBw), viewed July 24, 2015.

might be possible to expand it. That thought inspired me to put my mathemagical skills to the test, and attempt to figure out a method to make not just this magic square, but ANY magic square. That is, a method to make a magic square of any named size, adding up to any named sum. I knew it was an ambitious plan, but I was hooked! I decided to focus on making a magic square at all first, and to leave the sum-part for later. I quickly discovered that finding one method for every size would be impossible.

As it turns out, there are in fact three categories, each one having its own technique for constructing the magic square. First, there are the odd numbers, which are very simple. The even numbers however, are more complicated. There are the so-called “singly even numbers”, numbers divisible by two, but not by four, on one hand, and on the other hand the “doubly even numbers”, that are divisible by four. Each of those categories has its own technique, which will be explained in detail. This will be part one.

The second part will be a step up from that, and will be about making the square add up to any number. The techniques from part one will be the foundations for part two, so spend some time with them, before moving on.

The third and last part of this book will be a detailed look into the inner workings of the Magic Square Generator, which is a site I made, that does all of this automatically. Figuring out how to automate this entire process came with its own difficulties,

and at first I wasn't even sure if it could be done, so I am very proud that I got it to work. Now let's get into it.

Prologue: The Scam School trick

Before I get into the general creation of magic squares, it seems only appropriate to start this book with the trick that started the whole thing. Watching the video is of course a must, but it is advised to read this as well, because I will be explaining the math behind it as well. The trick consists of two parts: the forcing of the number, and the making of the magic square. For the first part, write down a four by four grid, and fill it out with all the numbers from 1 to 16, as such:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Then, ask your spectators to select four of the numbers. Each time however, it is important to eliminate all the other numbers in the column and row of the chosen number. This means that in every row and column, there will be only one selected number. Other than that, they are free to select any number they want. An example:

		3	
5			
			12
	14		

Then ask one of the spectators to add up all the selected numbers, the result of which will be the random number. In this case that would be: 3+5+12+14=34. The secret is of course that it is not random at all. In fact, it will always be 34. Let me explain why that is. Here is that same grid again:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Another way to write that though, is this:

1+0	2+0	3+0	4+0	0
1+4	2+4	3+4	4+4	4
1+8	2+8	3+8	4+8	8
1+12	2+12	3+12	4+12	12
1	2	3	4	

Essentially, every column and every row has a number associated with it, and the value of each field is simply the sum of the numbers corresponding to its row and column. This means that when one number is chosen from every row and column, it actually comes down to the same sum every time: 0+4+8+12+1+2+3+4=34. Thus, it does not matter what numbers are selected, as the sum will always be the same. Before we move on, it may be important to note that this technique works for every size, meaning you can do it with a five-by-five, ten-by-ten, or hundred-by-hundred, and always get the required number. Needless to say, that number will of course vary according to the size of your square. The second part is making the magic square. Here is the full thing again:

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

To make it, you need to see it as four groups of four consecutive numbers. These groups will all be filling the grid in parabola-shapes: two big ones, two little ones; two mountains, two valleys; two starting left, two starting right:

	3	2	
4			1

5			8
	6	7	

	10	11	
9			12

16			13
	15	14	

Fill out the entire grid like that, and you will get the magic square. This specific magic square is rather special, because not only do all the rows and columns, and both diagonals, add up to 34, but also the four numbers in the corners, and in every quadrant, and every other group of two-by-two squares. Unfortunately, I cannot explain why this layout does all these things, but it does.

This magic square is only the start. Now it is time to have a look at the techniques and the logic that will allow us to make magic

squares of any size. This book is called “How to make any Magic Square”, and please take that title very seriously. Although for practical reasons I advise to keep the square smaller than twenty-by-twenty, it is important to realize that in theory, should you wish to do so, it is possible to make a magic square of one-hundred-by-one-hundred and infinitely bigger. I have, however, never done it, as it would require hours of my time, and several square meters of paper.

As I said, there are three categories of numbers: odd, singly even and doubly even. All of these have a technique that works for any number in that category, although there are special cases.

A little bit of History

Magic squares have fascinated mankind for millennia, which is why I feel it would be irresponsible of me, not to talk a little bit about their history. If nothing else, the story of magic squares throughout the ages, can provide a basis for your presentation of the trick, but I hope you can also understand better why they are actually called magic, since it is more than their mathematical curiosity alone.

Throughout the centuries, scholars of varying origins and backgrounds have tried to figure out the secrets of magic squares. The Islamic scholars of medieval Persia and Arabia are known to have made several magic squares of different sizes,[2] but it is unclear when exactly the first magic squares were invented. The Chinese appear to have been the first.[3] The first Chinese magic squares are found in a book called Ta Tai Li-chi, a compendium about ancient rituals and philosophy, written in the first century. In this book, the three-by-three magic square is explained by making the analogy with the nine halls of the cosmic temple, where early emperors performed rituals. Because of this, anything to do with this square has been called the "Nine Halls calculations".

[2] *Wikipedia: Magic square*, 2018, (https://en.wikipedia.org/wiki/Magic_square). Viewed May 12, 2018.

[3] CAMMANN, S., 'The Evolution of Magic Squares in China', *Journal of the American Oriental Society*, 80(2), 1960, 116-124.

4	9	2
3	5	7
8	1	6

The Nine Halls magic square his also known as the Lo Shu magic square, because of a legend involving an ancient emperor and a turtle. According to legend, during the reign of the emperor Yu, who is said to have reigned between 2200 and 2100 BC, China was hit by a terrible deluge[4]. In the aftermath of the flood, the emperor was found walking along one of the big rivers, when he spotted a turtle with an interesting shell. Basically, on the shell were dots, arranged like the Nine Halls magic square.

It's clear from these origins, that the Lo Shu square was primarily, if not exclusively, ritualistic in nature, rather than an object of mathematical study. The Lo Shu square and it's variations appeared in philosophical writings, and they became the basis for a school of feng shui, called Flying Stars[5], where the movement and placement of the numbers in the grid, represent the principles of yin and yang in space. With this

[4] 'Wikipedia: Lo Shu Square', 2015, (https://en.m.wikipedia.org/wiki/Lo_Shu_Square), Viewed July 24, 2015.

[5] So ATP, Lee E, Li KL, Leung DKS. Luo Shu: Ancient Chinese Magic Square on Linear Algebra. *SAGE Open*. April 2015.

interpretation of the square as a basis, Chinese philosophers developed tools for divination.

The Chinese were the first we know of, but they were not the only culture to discover the concept of magic squares. Since the Vedic period (ca. 1500-500 BC), magic squares have been used in India, as well as the Islamic world, for rituals and as talismans or charms[6]. Variations on three-by-three and four-by-four magic squares, appear in several ways, going from floor decoration, over inscriptions in temples, to alchemy. In the first century AD for example, an alchemist by the name of Nagarjuna wrote a book called Kaksaputa, all about four-by-four magic squares. Notable is the so-called Jaina square, a four-by-four square in the Parshvanatha temple. This is notable, because due to its lay-out, there is a plethora of ways to group the numbers, which all give the magic sum of 34.

7	12	1	14
2	13	8	11
16	3	10	5
9	6	15	4

So what about Europe? One would probably think that the Greeks were the first to play around with magic squares, and in a way, they were. There is no evidence of ancient Greek

[6] CAMMAN, S., 'Islamic and Indian Magic Squares. Part I', *History of Religions*, 8(3), 1969, 181-209.

scholars studying magic squares, but Greek culture lived on through the Eastern Roman Empire, also known as the Byzantine Empire. I feel justified in saying the Greeks were technically first, in a way, because it was a Byzantine scholar by the name of Manuel Moschopoulos, who is credited with the introduction of magic squares in Europe.[7] Around 1315, Manuel wrote a treatise about them, for which he was most likely, like many Byzantine scholars, influenced by the Arabs. Western scholars at the time were more focused on studying scripture, than mathematical curiosities, and the Church was a little too dogmatic for the esoteric interpretations of magic squares, to really make their way to Europe.

But when the renaissance came around, pursuing knowledge for the sake of knowledge became not only possible, but a norm. Around 1500, Fra Luca Pacioli, a mathematician and friend of Leonardo da Vinci, wrote a book called *De Viribus Quantitatis*, in which he covers, amongst other things, magic squares. In this section, he shows the 7 planetary magic squares, which he says are vital in astronomy, although he does not actually say why or how. We can find the answers in magical handbooks, such as the *Liber de Angelis*, written around 1440. By far the most influential of these magical handbooks, was *De Occulta Philosophia*, written in 1510 by a guy called Heinrich Cornelius Agrippa. Each of the planets had certain characteristics and aspects of life, it was believed to

[7] SWANEY, M., *Mark Swaney on the History of Magic Squares*, (http://www.ismaili.net/mirrors/Ikhwan_08/magic_squares.html), Viewed July 24, 2015.

influence, and because each planet had a specific magic square assigned to it, these squares could be used in rituals. Showing you each of the planetary magic squares, would be unnecessary, and quite frankly a waste of space, but I cannot not show you the Jupiter square[8], since it closely resembles the square used in the Scam School episode.

1	15	14	4
12	6	7	9
8	10	11	5
13	3	2	16

Depending on what the ritual or spell is to accomplish, the square corresponding to the planet in charge of those areas, could be written in a particular way on a particular material, all of which had symbolic meaning. Another way magic squares were used for magic, perhaps the most popular one, was by making a so-called sigil.[9]

These days, though magic squares have not lost their symbolic and esoteric interpretations, they are mainly used recreationally, as entertainment for nerds, or as elements in

[8] STAPLETON, H., 'The antiquity of alchemy', *Ambix*, 5(1&2), 1953, 1-43.

[9] BENITEZ-FRANCES, L., *Reflections from the Black Stone: Sigil Creation with Planetary Magic Squares*, September 16, 2014 (http://voces-magicae.com/2014/09/16/sigil-creation-with-planetary-magic-squares/). Viewed May 12, 2018.

fiction. But they can in fact have practical uses, believe it or not. Due to their remarkable properties, magic squares have algebraic and calculatory applications.[10] One way magic squares can be applied practically, is by interpreting the values as masses.[11] That way, you can find the optimal way to lay out objects, so that the whole is perfectly balanced, which can be used in engineering. Ironically, this use is not unlike the ancient way of feng shui.

But I feel I cannot end this section on the modern uses of magic squares, without at least mentioning what is probably the most popular applications of magic squares: the Sudoku. While Sudoku's are not technically magic squares, they do share some key characteristics. The goal of a Sudoku is to arrange the numbers in such a way, so that every row, column, and each of the smaller three-by-three squares, has every number from one to nine in it, without repeats or gaps. While this is not the same as being a magic square, this principle is one of the keys to making them. Interestingly,

[10] LOLY, P., CAMERON, I., TRUMP, W., SCHINDEL, D., 'Magic square spectra', *Linear Algebra and its Applications*, 430(10), 2009, 2659-2680.

[11] FAHIMI, P., JAVADI, R., 'An Introduction to Magic Squares and Their Physical Applications', Submitted to *Parabola Journal*, 2015 (https://www.researchgate.net/publication/297731505_An_Introduction_to_Magic_Squares_and_Their_Physical_Applications). Viewed May 13, 2018.

Sudoku's are also used in digital communication.[12] Using algorithms, a message can be turned into a solved Sudoku, which is send. Then, if the message arrives incomplete, the AI can solve the Sudoku, and decode the message. So in a way, magic squares help us in running our modern society.

[12] PARKER, M., [STANDUPMATHS], *Matt Parker: Stand-up Maths Routine (about barcodes)*, video, December 6, 2011 (https://youtu.be/RP8jepN3zMc). Viewed May 13, 2018.

Part I: Making the Magic Square.

Odd numbers

The odd numbers are the easiest magic squares to construct, because the method is so simple to understand, and there is virtually no memorization needed. This method, the so-called "Siamese method" is also one of the oldest techniques for constructing magic squares. And because this works for every square of odd dimensions, it covers fifty percent of all cases you'll come across.

The Siamese method

The Siamese method works by filling out the grid using all the consecutive numbers from one up until the grid is full.[13] In fact, all of the following methods work like that. The big difference lies in where the numbers go. For the Siamese method, start by locating the middle field of the very top row, and write 1 in it.

		1		

[13] HOSPEL, T., *The math behind the Siamese method of generating magic squares*, (http://thospel.home.xs4all.nl/siamese.html). Viewed July 24, 2015.

Next, move to the column to the right of it, and the row above it. This means we go to the field that is directly touching the top right corner of the first field. In this case however, we see that there is no row above it. So to solve this problem, we wrap around the column, to get to the other side. In short, we count the bottom row as being the one above the top row. Likewise, we consider the leftmost column to be right next to the rightmost column. So move to the field to the top right of the 1, and write 2.

		1		
			2	

Now move the same way, and write 3.

		1		
				3
			2	

Now, wrap around horizontally, and continue filling out the next numbers.

		1		
	5			
4				
				3
			2	

When you reach the last number of this “diagonal”, five in this case, you see that the field to the top right of the 5 already has a number in it, so the next number, in this case that’s 6, cannot go there. So in order to continue, we do not move to the top right, but rather we move one down.

		1		
	5			
4	6			
				3
			2	

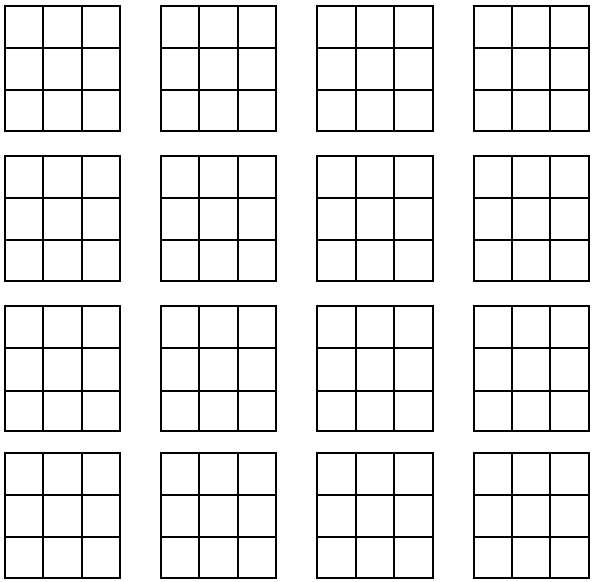
Then, continue filling out the numbers like before, wrapping around when necessary, and moving down when blocked.

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

Continue that way until the entire grid is filled. In the end, you will have a magic square, which adds up to the same number, in this case 65, in every row, column, and both diagonals. This is the Siamese method, and it is advised to learn this one very well, because it will come back in most of the other methods.

The Picture Principle

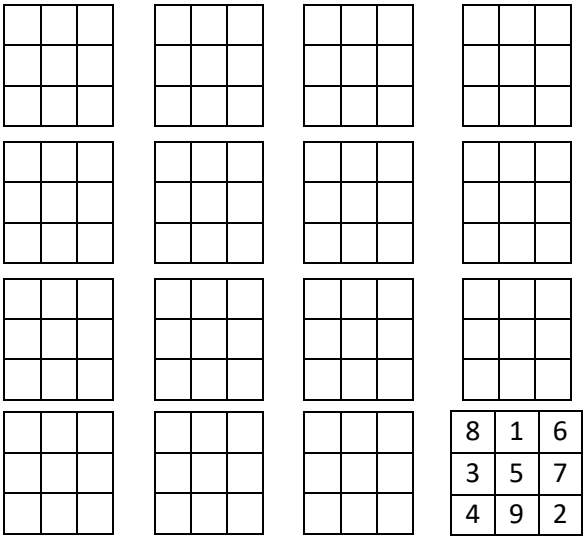
Before moving on to the even numbers, it is necessary to address an important principle which is the basis of most other methods. This was my own development, and though it's certainly possible that this already exists elsewhere, I haven't seen it to this extent. I call it the Picture Principle, because it uses a Big Picture and a Small Picture to make the Total Picture. The idea is to make a magic squares out of magic squares, by dividing a big square into a lot of smaller ones. A twelve-by-twelve grid for example, can be divided into a four-by-four grid of three-by-three grids.



The small grids, or the Small Pictures, will be filled with consecutive numbers, while the big grid, the Big Picture, determines in what order the Small Pictures will be filled. In most cases, the Big Picture will be made with its method for making a magic square. In this case, we can use the Scam School square for the Big Picture.

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

We see that the 1 is in the bottom right field, so that tells us that the very bottom right three-by-three will be the first to be filled. This will be done with the method we know for the three-by-three, which is the Siamese method.



Then, go to the grid where in the Big Picture is the two, and fill its Small Picture. Continue that way until the entire square is filled. This works, because the Small Pictures will make little magic squares, all rows and columns of which add up to the same. But the sum differs per little square. So by arranging them using the Big Picture, those sums are arranged to also add up as well. Basically we make a magic square out of magic squares, and the logic still works.

143 136 141 138 140 142 139 144 137	26 19 24 21 23 25 22 27 20	17 10 15 12 14 16 13 18 11	116 109 114 111 113 115 112 117 110
44 37 42 39 41 43 40 45 38	89 82 87 84 86 88 85 90 83	98 91 96 93 95 97 94 99 92	71 64 69 66 68 70 67 72 65
80 73 78 75 77 79 76 81 74	53 46 51 48 50 52 49 54 47	62 55 60 57 59 61 58 63 56	107 100 105 102 104 106 103 108 101
35 28 33 30 32 34 31 36 29	134 127 132 129 131 133 130 135 128	125 118 123 120 122 124 121 126 119	8 1 6 3 5 7 4 9 2

Singly even numbers

Singly even numbers are those numbers that are divisible by two, but not by four. Six for example, is singly even, whereas eight is not. There is probably a proper term for this, but I call them singly even. For this category, there are in fact two methods that I know of: the Strachey method, and the Radar method. For the purposes of this book however, the Strachey method is simply not suited, so there's not really any point in covering it here. But first, I need to talk about two-by-two magic squares.

Two-by-two

A proper two-by-two magic square is impossible. There are two ways to prove this, the first is to write down every possible arrangement for a two-by-two grid, and check whether any of them is a magic square. I have in fact done this, and as it turns out, there is no way. That is the empirical proof. But there is a more algebraic way to prove it. Let us just fill it with letters for a minute.

A	B
C	D

For this to be a magic square, the sum of every row, column, and both diagonals have to equal the same. This means that A+B=C+D=A+C=B+D=A+D=B+C. Simplified, that also means that A+C=A+B. A eliminates itself, leaving us with C=B. And this

can be done to prove that A=B=C=D. But that would mean that there are four repeating numbers, which is against the rules of magic squares. Thus, a two-by-two magic square is impossible. So the only thing you can do, if you find yourself having to make one, is this:

2	2
2	2

This is however more of a gag then it is a serious magic square. So if they insist, show them the proof above.

The Radar method

The Radar method is a method that I developed myself, after months of trial and error. It utilizes the Picture Principle, except that for the Small Picture something extra is required. The Big Picture is a grid half the size of the total square. Thus, it is by definition an odd-by-odd grid. But this provides a problem, which is that this leaves the Small Picture to be two-by-two grids, and a two-by-two magic square is impossible. We get around this by altering the way the Small Pictures are filled in, according to its position in the Big Picture. The way we determine how to alter it, is by using an extra tool called a radar, hence the name. The radar is a grid the same size as the Big Picture, filled with letters which represent the way to fill in the Small Picture. The most basic radar is the three-by-three one.

U	N	S
X	Z	N
S	E	A

The best way to memorize this, is by the words they make: Uns, which is German for “us”, and Sea, which is English. The middle row, you just have to repeat a couple of times. What those letters mean will be explain further. Now to expand this to bigger sizes. Let us start with the five-by-five radar.

First, copy the nine letters from the basic radar to their respective positions in the bigger grid.

U		N		S
X		Z		N
S		E		A

Continue filling the diagonals with the letter corresponding to that corner, and do the same with the top half of the middle column and the right half of the middle row.

U		N		S
	U	N	S	
X		Z	N	N
	S		A	
S		E		A

This leaves only the left half of the middle row, the bottom half of the middle column, and the “pieces of the pie”. Let us start with the pieces of the pie. You need to see those as eight pieces in a circle. We will fill them in, going around the circle, with S en Si, alternating. It does not matter on which one you start, or whether you start with S or Si. Personally, I like to start in the right piece on the bottom row, with S.

U	S	N	Si	S
Si	U	N	S	S
X		Z	N	N
S	S		A	Si
S	Si	E	S	A

Notice that, apart from the letters we copied from the basic radar, every row and column, except for the middle ones, have both an S and an Si. These compensate for each other, and make the whole thing work. This leaves the left half of the middle row, and the bottom half of the middle column. Notice

that the diagonal these pieces are on, consists of four pieces, with an S on one side, and Si on the other. We will continue this trend, by filling in the rest so the diagonal alternates S-Si-S-Si.

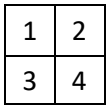
U	S	N	Si	S
Si	U	N	S	S
X	S	Z	N	N
S	S	Si	A	Si
S	Si	E	S	A

To expand it further, it works the same way, only the pieces of the pie will be bigger, resembling actual pieces of pie, and the halves of the middle row and column will be longer.

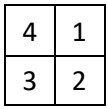
U	S	S	S	N	Si	Si	Si	S
Si	U	S	S	N	Si	Si	S	S
Si	Si	U	S	N	Si	S	S	S
Si	Si	Si	U	N	S	S	S	S
X	S	S	S	Z	N	N	N	N
S	S	S	S	Si	A	Si	Si	Si
S	S	S	Si	Si	S	A	Si	Si
S	S	Si	Si	Si	S	S	A	Si
S	Si	Si	Si	E	S	S	S	A

So that is the radar. The way of filling in the Small Pictures depends on its position on the radar. It is not that hard to memorize and construct the radar, but the important thing is of course what the letters stand for. Most of the letters are

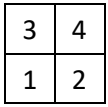
chosen because they look like the line between the consecutive numbers. N and Si are exceptions to this rule. N stands for Normal, which is the way we read. Si is the inverse of S.



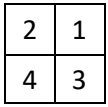
N



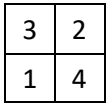
U



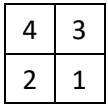
S



Si



X



Z

2	1
3	4

E

3	2
4	1

A

Now that we have the radar, we can start constructing the actual magic square. This is done with the Picture Principle, using the radar for the Small Pictures. To do so, get your grid, and in your mind, separate it into the Small Pictures. The Big Picture will follow the Siamese method. For the arrangement of the Small Pictures, check the position of it in the Big Picture, and compare that in your mind to the radar to see what letter it is, thus how to arrange it. An example:

68	65	95	96	1	2	30	29	59	60
67	66	93	94	3	4	32	31	57	58
90	89	20	17	25	26	55	56	63	64
92	91	19	18	27	28	53	54	61	62
15	14	23	24	52	51	77	78	85	86
13	16	21	22	50	49	79	80	87	88
39	40	47	48	74	73	83	82	10	9
37	38	45	46	76	75	84	81	12	11
43	44	70	69	98	97	7	8	35	34
41	42	72	71	99	100	5	6	36	33

Doubly even numbers

If singly even numbers the even numbers that are not divisible by four, it stands to reason that the doubly even numbers, are divisible by four. Once again, there is probably a proper term for this category, but this is what I call it. To make a doubly even magic square, I only ever use the Picture Principle. The Big Picture will be a four-by-four, and the Small Picture is of course whatever is left. The important thing to keep in mind here, is that this means the Small Picture's size could potentially be any number at all. So the size of the Small Picture might be an odd number, or a singly even number, or even a doubly even number. The point I'm trying to make, is simply that this could potentially get very very complex.

It also means that how complex a magic square is, has little to nothing to do with it's size. A twentyfour-by-twentyfour magic square for example, is more complex than a hundred-by-hundred magic square, since the former has a singly even number as its Small Picture, while the latter has an odd number as its Small Picture.

The Jaina Big Picture

I have previously used the Scam School square as a four-by-four Big Picture, but there is also a way, in which the four consecutive numbers of each of the four groups, never share a row or column.[14] This feature will be vital in the second part, where we will manipulate the magic sum of the magic square. For this reason, it is the one I use for the Big Picture of the doubly evens. You'll notice that this is essentially the same lay-out as the Jaina magic square.

14	1	12	7
11	8	13	2
5	10	3	16
4	15	6	9

Here's how to actually construct this thing. Start filling it out like this:

	1		7
	8		2
5		3	
4		6	

[14] MISMAG822, *Magic Square Tutorial*, video, January 12, 2010 (https://youtu.be/NVx9xfOl10o). Viewed July 24, 2015.

Notice that the numbers in the columns always add up to nine. Now, we have all the numbers from one to eight. So let us do the next four. To do so, we shall start in the bottom right, and go up, making sure none of them share a row or column.

	1	12	7
11	8		2
5	10	3	
4		6	9

Then, 13 will go right underneath 12. The last three are filled in on the outer ring, starting in the top left, going counterclockwise.

14	1	12	7
11	8	13	2
5	10	3	16
4	15	6	9

A four-by-four radar

What if you need to make an eight-by-eight magic square? We can't just use the Picture Principle as it's used on the other doubly even sizes, because the Small Picture's size will be two, which is impossible. And we can't use the radar we use for singly even squares, because that radar is meant for Big Pictures of odd sizes, and it can't just be adapted to a size of four, without losing its effectiveness. The solution is to use a custom radar, specifically for eight.

Z	N	Z	N
N	Z	N	Z
Z	N	Z	N
N	Z	N	Z

As you can see, it is really just a checkerboard, alternating N and Z. Using this radar, and the same Picture Principle, making the eight-by-eight magic square could not be easier.

This concludes Part I, about the making of the squares. It is advised to now wait a bit before going to Part II, and to first repeat and practice the techniques explained above, so you have them down when starting Part II. This is because, as I said, these techniques will be used extensively in Part II. So practice, and prepare to go to the next level, where everything is even harder. See you on the other side!

Part II: Manipulating the Sum.

The basic technique for manipulating what the sum will be, is very simple in theory, but it could be tricky to actually do it in the moment. Once you have created the magic square, it already has a sum, so all you need to do is distribute whatever the difference is between the sum it has, and the sum you want, throughout every row and column. Keep in mind that the sum is calculated based on the rows, columns, and diagonals, not the entire square, so distributing it evenly across the entire square, would not give the correct sum.

Since the numbers are filled in sequentially, all you need to do to distribute this difference, is to add it to the starting number, and it will automatically be added to the rest of the square as well. So when you get the size and the sum for the magic square, you simply need to calculate what that first number will be. With a random sum, this will usually not be 1, like it was in Part I. We calculate this with a formula, which will be explained in detail. Once you have this first number calculated, make the magic square using the appropriate method from Part I. The great thing about this technique, is that the value of the sum is completely independent from the actual method used to create the magic square. As such it is the same method, regardless of the given size.

The big problem with this technique is that it will take several calculations, which can get rather complex. Possibly too complex to do all in your head. Not to mention that depending on the sum you're given, you could very well end up dealing

with negative numbers and fractions. There are ways to simplify the calculating, and practice goes a long way. But in some cases, it may just be simpler to cheat a little bit. I will show you a technique I call the Correction, which concentrates the fractions in a way that usually makes the math a lot simpler. The resulting magic square will unfortunately not be perfect, as one of the diagonals will not share the magic sum. It will also not use purely sequential numbers, like every magic square we made so far. Though every number will still be different, meaning it doesn't go against the definition of a magic square, this does still feel dirty to me, so I prefer not to use it, if I can avoid it. But let's calculate our first number properly first.

The formula

$$F=(S(- R))/X - Y$$

Allow me to explain. F represents the first number, S is the required sum, R is the potential Rest, X is the size of the square, and Y is the Differential. The Rest will only come into play if you choose to resort to the Correction, so usually it will be 0. I still include it in the formula, because it is important to account for it in calculating the first number, since otherwise you will try to include it twice. The Differential is a number specific to X.

Basically, you need to divide the required sum by X, the size of the square. If S is not divisible by X, you may end up with a rest that does not provide a good fraction to work with, so you'll need to simply remember this number. Ignore the existence of

this Rest as you calculate the first number, but keep it in mind as you approach the Correction. After subtracting R, divide the number by X. Finally, subtract the Differential number from it, and you have your first number.

This is based on the technique for controlling the sum of a five-by-five which is used by James Grime in his demonstration video.[15] Why we do this, and why it works, is harder to understand. Subtracting R is of course to make it divisible, and not to have to work with those stupid decimals. Dividing by X is because before that, we have what has to become the sum of every row/column, but we need to know what to write in each field, which every row/column has X of. So the result of that division is what we were to write in every field, if repeating numbers were allowed. But they are not, so we have to make sure that this is the average value of the fields, by subtracting the exact right number.

What that number is, is originally determined by the outcome of these same calculations for the normal magic square from Part I. Whatever the difference is between that result, and one, which is the number we start within those squares, will be the Differential. But you won't need to run all these calculations to find the Differential.

[15] SINGINGBANANA, *Response: Magic Square Tutorial*, video, January 18, 2010 (http://youtu.be/tTCzcBs2b5Q). Viewed July 24, 2015.
KASABIANFAN44, *5x5 magic square tutorial*, video, October 16, 2012 (http://youtu.be/IXcJpWWgin8). Viewed July 24, 2015.

The Differential

The Differential varies for each size, but it's easy enough to calculate with the formula. The bad news is that it is a formula.

$$Y=((X^2) - 1)/2$$

As they did in the previous formula, Y represents the Differential, and X represents the size of the magic square. The main reason I separate this into a different formula, rather than making it part of the formula for the first number, is because that would complicate the formula, and you can in fact calculate this separately, before you even ask for what the sum will be.

So to calculate the Differential, you first square the size of the magic square. I can hardly imagine someone reading this book, to not know the squares of the single digit numbers by heart, but multiple digit numbers are harder to square. An easy way to multiply multiple digit numbers by themselves, is to simply multiply it by each of the digits, though remember to multiply by the appropriate multiple of 10, depending on its place, and add each of those products together. Subtract 1 from this, and then divide that by 2. If the subtraction makes it odd, subtract 2 for easy calculations, and simply remember to add a half to the quotient. This gives you the Differential you will need to subtract to get your first number.

The problem with this formula is that while it's easy enough in itself, it becomes harder to do in your head during a performance, when you combine it with the formula for the first number. The reason is that that formula started as something a little more complicated.

$$F=((S-(((X^2+1)/2)X))/X)-1$$

Here the Predicted Sum is calculated, which is the sum a magic square of the requested size would have, assuming it's filled with the consecutive numbers from 1 to the size squared. I'll get back to that, but I simply inserted the formula for that into this formula. The thing is that that makes the formula very difficult to read, and so I simplified it, into the formula's for the first number and the Differential. That is a lot easier to read, but also a lot harder to actually do in your head. But the math in your head can be made a lot simpler, by simply taking one step back in the simplification.

$$F= (S – (((X^2) – 1)/2)X)/X$$

So what this means, is that we'll subtract the Differential before we do any division. To make this work, we simply multiply the Differential by the size. For the even sizes we can actually skip halving it, and instead multiply it by half the size. Subtract that from the requested sum, and divide that by the size, to get the first number. This makes the calculation more of a step by step process, and means you won't have to remember the Differential while you go through the division, which makes it a lot easier to do in your head.

3: 12

4: 30

5: 60

6: 105

7: 168

8: 252

9: 360

10: 495

11: 660

12: 858

13: 1092

14: 1365

15: 1680

16: 2040

17: 2448

18: 2907

19: 3420

20: 3990

The Correction

Sometimes you will calculate the first number, and end up with a fraction you can't simply translate to a decimal number, so you may want to cheat a little. The idea is to concentrate the Rest on one place in each row etc., rather than distributing it. We add it once there is one diagonal left to be filled out. This way every row and column gets the Correction, without interfering with each other. Don't take this diagonal too literally though. The order in which the fields are filled in, is determined by the method you learned in Part I, so it won't be an actual diagonal. Rather, it will be the moment when there is only one blank space left per row/column. If you are using the Picture Principle, this refers to the last diagonal in the Big Picture. In other words, if you have a Big Picture of X-by-X, you go on until there are X Small Pictures left blank. That is when you add the Correction to the next number you fill in, and continue as you were. Of course, if you use the Picture Principle, you need to still distribute it across the Small Picture, so the actual number you add, will be the Rest, divided by the size of the Small Picture.

This seems simple enough, but unfortunately the math can get rather difficult to do off the top of your head. This is because the Rest will always, by definition, not be divisible by X. In most cases, the number you're dividing will not be easy to divide by. There are no real tricks to make this easier, except practice. I can however say that if you know what one over all the numbers is, the math will be a lot easier. So if you get a fraction like 3/5, you know that 1/5 is 0,2 and therefore the fraction has

to be 0,6. Another example could be 2/7. Because 1/7 is about 0,1428 this means that the fraction is about 0,2856. This will not be applicable to every situation though, so if you want to do it right, you may need to do it on paper, or even use a calculator, and just do it properly.

Like I said, this will make a grid that is not technically a magic square, since one of the diagonals will be missing the sum. You will also need to make a leap in your numbers at some point, which is not forbidden, but I still feel it taints the magic square. On top of that, you'll have a Rest to keep track of throughout making the square, and the division with the Small Picture, may not provide an easier fraction at all. So I personally prefer not using this if I can avoid it, but sometimes it can be easier, so it's here if you need it.

The Predicted Sum

Before I conclude Part II, I would like to have a quick word about the sum. Because every magic square has a Predicted Sum, a sum it is supposed to have, assuming it is filled with sequential numbers starting on 1. A four-by-four for example, will have 34 as its Predicted Sum, a twelve-by-twelve has 870, and so on. How big the numbers will be that you will work with, like the Rest, the Differential, or the first number, depends on the sum they chose. If the difference between that sum, and the Predicted Sum, is too big, those numbers will be pretty big as well. So it may be a good idea to know what that Predicted Sum is, in order to keep things workable. Therefore, following will be a list of the supposed sums, and a possible interval for every size from 1 to twenty.

3: 15

4: 34

5: 65

6: 111

7: 175

8: 260

9: 369

10: 505

11: 671

12: 870

13: 1105

14: 1379

15: 1695

16: 2056

17: 2465

18: 2925

19: 3439

20: 4010

Memorizing this would be hard and pointless. Luckily, it turns out there is a formula for the Predicted Sum as well[16].

$$((X^2+1)/2)X$$

First, you square the size of the magic square, and add one to it. Then half it, and multiply it by the size. This will give you the sum you can expect from a magic square of this size, and it can inform whether or not the sum they ask for, is too high or too low. There is no technical reason to do this, but it can be difficult to intuitively guess how high these sums can go, so if they just pick a number they think of, you may need to dive deep into the negatives, which just isn't very satisfying.

[16] *Wikipedia: Magic Square*, 2015 (https://en.m.wikipedia.org/wiki/Magic_square), Viewed July 24, 2015.

An example

All this theory, and all these techniques are very interesting, but it may be rather confusing what it really is you need to actually do in a performance. So I will break it down step by step, using an example. First, ask a spectator to name any number between one and twenty. Let's say they name 8. Give that spectator some paper, and a pen, and tell them to draw an eight-by-eight grid.

Meanwhile, I am already doing some calculations. Because this doesn't really matter too much in a grid of this small size, I don't actually bother with calculating the Predicted Sum. Instead, I calculate the Differential. 8 squared is 64. I subtract 1, but because I will then half it, I actually subtract 2, to make 62, and remember to add ,5 to the result. After halving it, I end up with 31,5 as the Differential, so I will remember that as I move on.

Now I ask a different spectator to choose the sum. Let us say 265 is named. Now I can calculate what the first number has to be. Turns out 265 is not divisible by eight, but 264 is. That means the Rest is 1. In this case 1/8 is simply 0,125, so in this situation I could easily go with that. But for the demonstration, it may be interesting to use the Correction instead, so I remember 1 as my Rest. Having subtracted the Rest from the sum, I then divide 264 by eight, which gives 33. Now I just have to subtract the Differential, which I already know to be 31,5. So I will subtract this 31,5 from the 33, giving me 1,5 as the first number.

Then I can begin constructing the magic square. The size is eight, so I will use the Picture Principle, with a Big Picture of four, and Small Pictures of twos. I will fill the Big Picture using the Jaina square, since it allows me to actually have a diagonal to concentrate the Correction onto, and the Small Pictures using the Checkerboard-Radar.

		1,5	2,5	48,5	47,5	25,5	26,5
		3,5	4,5	46,5	45,5	27,5	28,5
41,5	42,5	32,5	31,5			8,5	7,5
43,5	44,5	30,5	29,5			6,5	5,5
20,5	19,5	37,5	38,5	12,5	11,5		
18,5	17,5	39,5	40,5	10,5	9,5		
13,5	14,5			21,5	22,5	36,5	35,5
15,5	16,5			23,5	24,5	34,5	33,5

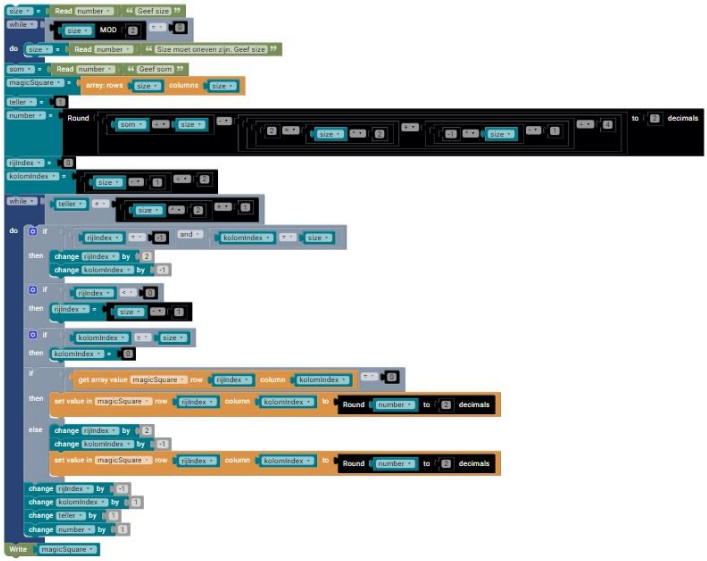
Filling this in is pretty mindless work, so meanwhile, I have plenty of time to calculate the Correction. The Rest is 1, and the Small Pictures are two-by-twos, so I have to divide 1 by 2, which gives ½, or ,5. This means I only have to add 0,5 to the next number I write down, which would be 49,5, but will now be 50.

57	56	1,5	2,5	48,5	47,5	25,5	26,5
55	54	3,5	4,5	46,5	45,5	27,5	28,5
41,5	42,5	32,5	31,5	50	51	8,5	7,5
43,5	44,5	30,5	29,5	52	53	6,5	5,5
20,5	19,5	37,5	38,5	12,5	11,5	62	63
18,5	17,5	39,5	40,5	10,5	9,5	64	65
13,5	14,5	61	60	21,5	22,5	36,5	35,5
15,5	16,5	59	58	23,5	24,5	34,5	33,5

All that is left to do, is to have them check all the sums, and hope I didn't make any mistakes. Since I used the Correction, I conveniently forget to mention that the diagonals are also supposed to add up to that same sum.

Part III: Automating the Process.

For a long time, this was where my quest for magic squares ended. I succeeded in my goal. I was able to create a magic square for any given size, with any given sum. Several years after I completed the first draft of this booklet, I enrolled in a programming course. One of the first assignments we got, even before we got to any actual syntax, we were told to build a little program that could make a magic square of any odd size. With the background I had, I felt compelled to make the program a little more interesting than what they asked for. It was easy enough to add a function that could calculate the first number, allowing the program to customize the magic sum of the created magic square. Unfortunately I was unable at the time, to also implement the even sizes, which would allow it to return a magic square of any size and any sum, since that would have taken so much more time and effort, than I could justify spending on what was originally a rather small assignment.

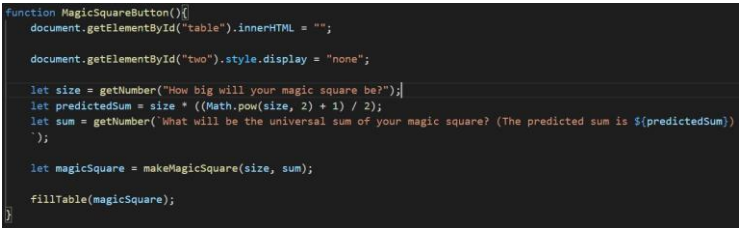


But it gave me an idea to expand on my original quest, and automate the process, by writing a JavaScript function that could do all this automagically. At first, I was not even sure if this would be possible, since some of the methods, specifically the Picture Principle, use a kind of relativistic thinking, that I wasn't sure I'd be able to automate. It took a while to figure out how to implement it in code, but I was able to find a different way of approaching the Picture Principle, and to streamline the method so it could be applied on any size a user asks for. In the end, I launched a little website for it, if only to allow for easy use, and of course shareability. In this chapter, I want to show you how I automated all of this, to allow a computer to do it.

https://magic-square-generator.netlify.app/

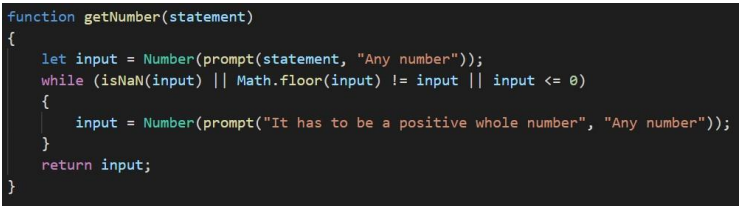
Front End

I won't go too deep into the front end of the website, because it is mostly just the interface, which allows the user to launch the function, input their wishes, and display the result. However, there are a couple of interesting things about it, that are worth highlighting.



```
function MagicSquareButton(){
    document.getElementById("table").innerHTML = "";

    document.getElementById("two").style.display = "none";

    let size = getNumber("How big will your magic square be?");
    let predictedSum = size * ((Math.pow(size, 2) + 1) / 2);
    let sum = getNumber(`What will be the universal sum of your magic square? (The predicted sum is ${predictedSum})
    `);

    let magicSquare = makeMagicSquare(size, sum);

    fillTable(magicSquare);
}
```

When the button is clicked, a single central function is launched, which does not itself make the magic square, but rather resets the page to prepare it to display the square that will be created. It then calls a separate function, which asks for a number, validates it, and if necessary, asks again until the input is valid, which it returns. As an extra feature, I made sure to calculate what the Predicted Sum would be for this size, which is then communicated to the user as a guideline.



```
function getNumber(statement)
{
    let input = Number(prompt(statement, "Any number"));
    while (isNaN(input) || Math.floor(input) != input || input <= 0)
    {
        input = Number(prompt("It has to be a positive whole number", "Any number"));
    }
    return input;
}
```

Once all the input has been validated and collected, it calls a separate function, which actually does all of the work, and which actually creates the magic square. Once that's done, it passes that magic square to another function, which takes care of displaying it in the page.

```
function fillTable(magicSuareArray)
{
    const table = document.getElementById("table");
    for(rowIndex = 0; rowIndex < magicSuareArray.length; rowIndex++)
    {
        let row = document.createElement("tr");
        for(columnIndex = 0; columnIndex < magicSuareArray.length; columnIndex++)
        {
            let cell = document.createElement("td");
            cell.innerHTML = magicSuareArray[rowIndex][columnIndex];
            row.appendChild(cell);
        }
        table.appendChild(row);
    }
}
```

Back End

You may have noticed that making a magic square only requires two numbers from the user, and everything else is simply following the logical steps and making the calculations, based on those two numbers, until you end up with a grid of numbers. This makes it perfectly suited for implementing it into a function like this one.

The first thing a human would do with these numbers, would be to calculate what the first number will be. But since this is nothing more than plugging these numbers into a formula, I decided it would be easier to make the general magic square for this size first, starting with zero, and add the first number to each number at the end. So the next first thing we would do with these numbers, would be to decide which method to use to construct the magic square, based on the category it belongs to. At first I thought I would need to implement a complex if-else structure to decide this, but the doubly even numbers mean that this could become overly complicated. An odd number is always odd, and a singly even number will by definition only take one division to become odd. But a doubly even number, after dividing by four, could theoretically still give any number, including a doubly even number. This means that, in order to make the Small Picture, we would have to run the entire function again, inside itself. Potentially multiple times. I realized that even if that was technically possible, it would probably still reuse variables, overwriting them in the process, breaking the code. So I knew I had to find a better way to do this.

Since the basic methods for the Picture Principle and the radars are essentially the same regardless of its category, I wanted to be able to reuse those functions with appropriate parameters depending on the input. In the end, I realized I could simplify the process immensely by dissolving the size-parameter into its essential factors. This allows the program to create the magic squares in predictable steps, without having to rerun functions inside themselves. So the first thing it does, is to run the size through a filter which keeps dividing it by four, until what's left cannot be divided by four, and keeps a count of how many times it was divided. This way, we know exactly how many times it will need to nest Small Pictures into four-by-fours. Of course, if the size is odd or singly even, this number will be zero. At that point, what's left can't possible be doubly even, but it could be either singly even or odd. So then it defines a Boolean variable, which is true if the remainder is divisible by two. If so, the remainder is divided by two, and we are left with the remainder, which is definitely odd. These three variables allow us to follow the same process every time.

```
let fourCount = 0;
let remain = size;
while(remain % 4 === 0)
{
    remain /= 4;
    fourCount++;
}
let remainEven = remain % 2 === 0 ? true : false;
remain /= remainEven ? 2 : 1;

let bigPictureArray = makeSiamese(remain);
```

The next step is to create our first basic magic square. This will be the smallest factor, so it is by definition of an odd size. It could be one, but one is also odd, so while this is a little obsolete, it can't break the code, so it's easier to simply run it. I made a separate function which uses the Siamese method to return a magic square of an odd size, starting with zero. I define it as the Big Picture, because I'm assuming this is part of a singly even magic square, in which case this will be the Big Picture. If it isn't, this will simply be carried to the next step.

```
function makeSiamese(siameseSize)
{
    let siameseSquare = setMultiArray(siameseSize, ".");

    let counter = 0;
    let rowindex = 0;
    let columnindex = (siameseSize - 1)/2;

    while(counter !== Math.pow(siameseSize, 2))
    {
        if(rowindex === -1 && columnindex === siameseSize)
        {
            rowindex += 2;
            columnindex--;
        }
        if(rowindex < 0)
        {
            rowindex = siameseSize - 1;
        }
        if(columnindex >= siameseSize)
        {
            columnindex = 0;
        }
        if(siameseSquare[rowindex][columnindex] == ".")
        {
            siameseSquare[rowindex][columnindex] = roundToTwo(counter);
        } else {
            rowindex += 2;
            columnindex--;
            siameseSquare[rowindex][columnindex] = roundToTwo(counter);
        }
        rowindex--;
        columnindex++;
        counter++;
    }

    return siameseSquare;
}
```

The Siamese magic square making function may be one of the least efficient of these functions, because despite being very simple and intuitive for a human, for a computer it's necessary to account for every eventuality. At the same time, this method is scalable to any odd size, meaning hardcoding it is

not an option. So the algorithm follows the exact same reasoning a human would. It starts at zero, in the middle of the top row. Then it simply checks the coordinates to make sure they are still in the grid, and if it isn't occupied yet, and makes the necessary changes. It then puts the counter in the location it landed in, and adjusts the coordinates and the counter, so they are ready for the next round. This Siamese magic square is then returned back to the main magic square making function, which can use it for the next step. The program then initiates two more grids, including a basic four-by-four magic square. This is always the same size, so the easiest option is to simply hardcode it.

```
let smallPictureArray;

let baseFourArray = [
    [13,0,11,6],
    [10,7,12,1],
    [4,9,2,15],
    [3,14,5,8]
];
```

In the beginning of all this, we turned the size of the magic square into three variables that describe it. Using these variables, we can now run through a simple if-else filter, to figure out which steps we should take, and which ones are impossible.

```
if(remainEven)
{
    if(remain === 1)
    {
        if(fourCount === 0)
        {
            let twoPara = document.getElementById("two");
            twoPara.style.display = "block";
            return setMultiArray(2, sum/2);
        } else {
            fourCount--;
            let radar = makeRadar(4);
            smallPictureArray = makePictureWithBigPicture(baseFourArray, radar);
        }
    } else {
        let radar = makeRadar(remain);
        smallPictureArray = makePictureWithBigPicture(bigPictureArray, radar);
    }
} else {
    smallPictureArray = bigPictureArray;
}
```

This constellation of conditions allows us to check every eventuality we need to check. If the Boolean is true, the remainder is one, and there are no fours, it means that the requested size is definitely two. Obviously we have access to that variable, so we could have just tested that, but that would not have been as efficient as this. So if it's two, it will display a paragraph in the page, which explains that a two-by-two magic square is impossible. It also calls an external function which makes two dimensional arrays with the dimensions in its parameter, and fills it with the placeholder in the second parameter. In this case it makes a two-by-two grid and fills it with half of the requested sum. It's not really a magic square, but it's the closest we get on a two-by-two. That grid is returned to the front end as the magic square, and we exit this function.

If there is at least one four, but the remainder is still one, and the Boolean is still true, that means we have a Small Picture of two, nested in a Big Picture of four. To deal with this, we will make an eight-by-eight magic square, starting at zero. In this case we roll back the count of fours by one, because one of them is used for the eight. We then call a separate function to create the radar. All this function needs, is for us to tell it the size of the radar. That size is the size of the Big Picture, so for an eight-by-eight square, that's four.

```
let radar = setMultiArray(radarSize, ".");
if(radarSize === 4)
{
    let bool = false;
    let counter = 1;
    for(let i = 0; i < 4; i++)
    {
        for(let j = 0; j < 4; j++)
        {
            radar[i][j] = bool ? "Z" : "N";
            bool = counter % 4 !== 0 ? !bool : bool;
            counter++;
        }
    }
} else {
```

The radar making function first creates a grid of the size of the radar, and fills it with placeholders. If that size is four, it establishes a Boolean and a counter, to keep track of where it is in the radar. It then iterates over the radar, and checks the Boolean to determine which of the two letters to put down. In preparation for the next round, it checks whether or not it is at

the end of a row, and if it is not, it flips the Boolean. If it is, the Boolean stays as it is. Then it adds one to the counter. That way the radar will end up in the same checkerboard pattern I laid out earlier. You may notice that the way this is actually coded, it will actually be flipped, in comparison to the one I showed before. The honest truth is that it doesn't matter as long as the checkerboard is there. This radar is returned to the main function, where it is passed to the function that will make a Total Picture using the radar, and the basic four-by-four square as the Big Picture.

```
function makePictureWithBigPicture(bigPicturePara, smallPicturePara)
{
    let radar = isNaN(smallPicturePara[0][0]);
    let multiplier = radar ? 2 : smallPicturePara.length;

    let picture = setMultiArray(multiplier*bigPicturePara.length, ".");
    for(let i = 0; i < picture.length; i++)
    {
        for(let j = 0; j < picture.length; j++)
        {
            let bpRow = bigPictureCoordinate(i, multiplier);
            let bpColumn = bigPictureCoordinate(j, multiplier);
            let spRow = smallPictureCoordinate(i, multiplier);
            let spColumn = smallPictureCoordinate(j, multiplier);

            let modifier = radar ? findNumberByRadar(smallPicturePara[bpRow][bpColumn], spRow, spColumn) :
            smallPicturePara[spRow][spColumn];

            picture[i][j] = (Math.pow(multiplier,2))*(bigPicturePara[bpRow][bpColumn]) + modifier;
        }
    }
    return picture;
}
```

The first thing this function does, is checking whether or not the Small Picture parameter is a radar. It does so by making a Boolean variable, and defining it by checking if the first cell of that grid is a letter. If so, it's a radar, if not, it's a regular Small Picture. It then defines a multiplier, which will be important to get the correct value from the Big Picture. Due to the order one follows with the Picture Principle, the actual value in any specific cell, is actually the corresponding number in the Big

Picture, multiplied by the size of the Small Picture squared, added to the corresponding number in the Small Picture. When using a radar, the size of the Small Picture will be two, meaning this multiplier has to be 2, rather than the size of the Small Picture, since the Small Picture the program has, is actually the radar, which is a different thing altogether.

52+0	52+1	0+3	0+2	44+0	44+1	24+3	24+2
52+2	52+3	0+1	0+0	44+2	44+3	24+1	24+0
40+3	40+2	28+0	28+1	48+3	48+2	4+0	4+1
40+1	40+0	28+2	28+3	48+1	48+0	4+2	4+3
16+0	16+1	36+3	36+2	8+0	8+1	60+3	60+2
16+2	16+3	36+1	36+0	8+2	8+3	60+1	60+0
12+3	12+2	56+0	56+1	20+3	20+2	32+0	32+1
12+1	12+0	56+2	56+3	20+1	20+0	32+2	32+3

It then makes a grid the size of the Total Picture, and iterates over it. For every cell, it uses a separate function to calculate it's corresponding coordinate in both the Big Picture and the Small Picture. To know the position in the Big Picture, simply divide the coordinate by the size of the Small Picture, and round it down. Because JavaScript counts from zero, this will give the correct coordinate. To get the position in the Small Picture, you take that same division, but you look only at the remainder, which will equal the coordinate in the Small Picture. It is important that we pass the correct size, rather than the size of the radar, so we use the multiplier for this.

```
function bigPictureCoordinate(coordinate, smallPictureSize)
{
    return Math.floor(coordinate/smallPictureSize);
}

function smallPictureCoordinate(coordinate, smallPictureSize)
{
    return coordinate % smallPictureSize;
}
```

The modifier is the number we get from the Small Picture. If the parameter is not a radar, we only need to get the number at the appropriate position in the parameter, but if it is a radar, we need something extra. We will tell an external function what letter we find in that position in the radar, and what the coordinates are in the Small Picture. The function that makes a new variable which represents the Small Picture. Based on the letter, it will simply find the correct case, define the Small Picture as the corresponding grid, and return the number in the correct position in that Small Picture.

```
let smallPicture;
switch (token)
{
    case "N":
        smallPicture = [
            [0, 1],
            [2, 3]
        ];
        break;
    case "U":
        smallPicture = [
            [3, 0],
            [2, 1]
        ];
        break;
    case "S":
        smallPicture = [
            [2, 3],
            [0, 1]
        ];
        break;
    case "Si":
        smallPicture = [
            [1, 0],
            [3, 2]
        ];
        break;
    case "X":
        smallPicture = [
            [2, 1],
            [0, 3]
        ];
        break;
    case "Z":
        smallPicture = [
            [3, 2],
            [1, 0]
        ];
        break;
    case "E":
        smallPicture = [
            [1, 0],
            [2, 3]
        ];
        break;
    case "A":
        smallPicture = [
            [2, 1],
            [3, 0]
        ];
}
let value = smallPicture[row][column];
return value;
```

Finally, the Picture making function will calculate the number it needs to fill in, fill it in, and keep going until the entire Total Picture is full. This Picture then becomes the next Small Picture, as we prepare for the next step. But before we continue the process, I have to talk about the rest of the conditional filter. What if the remainder isn't one? What if the Boolean isn't true?

Well, if the remainder isn't one, but the Boolean was true, it means that we have to make a singly even magic square. The method is much the same as what we did in the previous case, except we need a different radar, and a different Big Picture. The Big Picture will of course be the Siamese magic square we made earlier. For the radar, we need to call the same method, except the size we give it, is the remainder, which is also the size of the Big Picture.

```
} else {
    for(let i = 0; i < radarSize; i++)
    {
        for(let j = 0; j < radarSize; j++)
        {
            if(i < Math.floor(radarSize/2)){
                if(i === j){radar[i][j] = "U"}
                else if(i > j){radar[i][j] = "Si"}
                else if(j === Math.floor(radarSize/2)){radar[i][j] = "N"}
                else if(j < Math.floor(radarSize/2)){radar[i][j] = "S"}
                else if(j < ((radarSize - 1) - i)){radar[i][j] = "Si"}
                else radar[i][j] = "S";
            } else {
                if(j <= ((radarSize - 1) - i)){radar[i][j] = "S"}
                else if(i === Math.floor(radarSize/2)){radar[i][j] = "N"}
                else if(i === j){radar[i][j] = "A"}
                else if(i < j){radar[i][j] = "Si"}
                else if(j > Math.floor(radarSize/2)){radar[i][j] = "S"}
                else radar[i][j] = "Si";
            }

            radar[Math.floor(radarSize/2)][0] = "X";
            radar[radarSize - 1][Math.floor(radarSize/2)] = "E";
            radar[Math.floor(radarSize/2)][Math.floor(radarSize/2)] = "Z";
        }
    }
}
return radar;
```

Remember when I said the Siamese function was one of the least efficient of the functions? This one probably takes the cake. What is happening here, is that the program will iterate over the grid it made for the radar, and runs its coordinates through a conditional filter to determine what area it is in, and based on that it knows what letter to put in place. There are however three special letters that do belong in these areas, but also appear only once each, so it just does those last. That radar is returned into the main function, which passes it, along with the Siamese square it made in the beginning, straight on to the Picture making function. Following the exact same function we just went through, it makes a singly even magic

square, and makes that the Small Picture in preparation of the next step.

And if the Boolean is false, it means that the remainder was never even to begin with, and the Siamese square from before is the in fact the Small Picture we need. It may be a one-by-one square, but that still works as a Small Picture. So in that case it will simply define the Small Picture as being that previous square, and move on to the next step.

```
let newSmallPicture = smallPictureArray;
if(fourCount !== 0)
{
    for(let i = fourCount; i > 0; i--)
    {
        newSmallPicture = makePictureWithBigPicture(baseFourArray, smallPictureArray);
        smallPictureArray = newSmallPicture;
    }
}
```

The program now defines a new Small Picture, which starts by being the exact same as the other Small Picture. The reason is that the old one will be used to redefine the new one, by passing it to the Picture making function as the Small Picture. Then the old one will be redefined as the new one, ready to be the Small Picture in the next round. This will be repeated as many times as we counted fours in the beginning. That means it may be done several times, it may only be done once, or it may not be done at all. The important part is that the final magic square we end up it, is always the same variable, regardless of whether or not this loop was even run. After this loop, we end up with the final general magic square. What I mean by that, is that it is now the exact size we need it to be, and it will provide the foundation for the actual magic square.

But it is not quite done yet. Before it can be returned to the front end, it needs to add up to the requested sum.

```
let firstNumber = (sum/size) - (Math.pow(size, 2) - 1)/2;
let magicSquare = setMultiArray(smallPictureArray.length, ".");
for(let i = 0; i < magicSquare.length; i++)
{
    for(let j = 0; j < magicSquare.length; j++)
    {
        magicSquare[i][j] = roundToTwo(smallPictureArray[i][j] + firstNumber);
    }
}

return magicSquare;
```

This works the same way it always has, by calculating the first number. there are two main differences with the human approach. Firstly, the formula for the Differential is put straight into the formula for the first number, mostly because this is a computer. Calculating things is what it does, so there's no reason not to. That also means that it has no problems with fractions, so there will be no need for any Corrections. Instead, the program will simply calculate the first number as precisely as possible, and then iterate over the magic square adding it to every value, and rounding it to two places. This might cost us some accuracy, meaning that the actual sum will not be exactly the sum that was asked for. The reason is purely esthetic. A magic square with infinite decimals in every cell, is not as clear or pretty as one where the numbers are rounded to two decimals. That final magic square is then returned to the front end, to be displayed on the website.

Conclusion

The world of magic squares is infinitely bigger than this. People have come up with many variations on the theme, such as magic hexagons, magic squares with square or cube numbers, even a magic square with digital numbers, and many others. I will not go into what that all means, or how to make them, because it would take me too far. If you want to see this book in video format, you can find that over on my own channel[17].

I had a blast writing this, and I hope you have enjoyed it. I also hope that everything was clear, and that you understood everything. For now, I hope you enjoyed reading this book and learning these techniques, as much as I enjoyed writing it and discovering them. It has come a long way from the trick as it was taught on Scam School.

When I first saw that video, I could not have expected that I would actually succeed in figuring this out, let alone that I would one day automate the whole process. But it all happened, and I'm very proud of the result. I suppose there are still ways to make the process easier, and I'm sure that code can be streamlined, but this is where I will call it. My work is done.

Thanks for reading!

---

[17] Nerdycopia: https://www.youtube.com/channel/UCU4hD43TaTTJMJu5S_mmpqw

Bibliography

BENITEZ-FRANCES, L., *Reflections from the Black Stone: Sigil Creation with Planetary Magic Squares*, September 16, 2014 (http://voces-magicae.com/2014/09/16/sigil-creation-with-planetary-magic-squares/). Viewed May 12, 2018.

BRUSHWOOD, B., [SCAM SCHOOL], *Look Like a GENUIS SAVANT! – An Easy, Convincing Magic Square Cheat*, video, 14 July 2010, (https://youtu.be/JKYUckzfnBw), viewed July 24, 2015.

Wikipedia: Magic square, 2018, (https://en.wikipedia.org/wiki/Magic_square). Viewed May 12, 2018.

CAMMANN, S., 'The Evolution of Magic Squares in China', *Journal of the American Oriental Society*, 80(2), 1960, 116-124.

CAMMAN, S., 'Islamic and Indian Magic Squares. Part I', *History of Religions*, 8(3), 1969, 181-209.

FAHIMI, P., JAVADI, R., 'An Introduction to Magic Squares and Their Physical Applications', Submitted to *Parabola Journal*, 2015 (https://www.researchgate.net/publication/297731505_An_Introduction_to_Magic_Squares_and_Their_Physical_Applications). Viewed May 13, 2018.

HOSPEL, T., *The math behind the Siamese method of generating magic squares*, (http://thospel.home.xs4all.nl/siamese.html). Viewed July 24, 2015.

KASABIANFAN44, *5x5 magic square tutorial*, video, October 16, 2012 (http://youtu.be/IXcJpWWgin8). Viewed July 24, 2015.

LOLY, P., CAMERON, I., TRUMP, W., SCHINDEL, D., 'Magic square spectra', *Linear Algebra and its Applications*, 430(10), 2009, 2659-2680.
MISMAG822, *Magic Square Tutorial*, video, January 12, 2010 (https://youtu.be/NVx9xfOl10o). Viewed July 24, 2015.
PARKER, M., [STANDUPMATHS], *Matt Parker: Stand-up Maths Routine (about barcodes)*, video, December 6, 2011 (https://youtu.be/RP8jepN3zMc). Viewed May 13, 2018.
SINGINGBANANA, *Response: Magic Square Tutorial*, video, January 18, 2010 (http://youtu.be/tTCzcBs2b5Q). Viewed July 24, 2015.
So ATP, Lee E, Li KL, Leung DKS. Luo Shu: Ancient Chinese Magic Square on Linear Algebra. *SAGE Open*. April 2015.
STAPLETON, H., 'The antiquity of alchemy', *Ambix*, 5(1&2), 1953, 1-43.
SWANEY, M., *Mark Swaney on the History of Magic Squares*, (http://www.ismaili.net/mirrors/Ikhwan_08/magic_squares.html), Viewed July 24, 2015.
'Wikipedia: Lo Shu Square', 2015, (https://en.m.wikipedia.org/wiki/Lo_Shu_Square), Viewed July 24, 2015.
'Wikipedia: Magic Square', 2015 (https://en.m.wikipedia.org/wiki/Magic_square), Viewed July 24, 2015.

Judith Opdebeeck
Youtube: https://www.youtube.com/c/Nerdycopia
Twitter: @Nerdycopia
Goodreads:
https://www.goodreads.com/user/show/82637062-judith-opdebeeck
TikTok: https://www.tiktok.com/@nerdycopia?