How to make a Random Number Generator

Computers cannot generate truly random numbers: instead, they create **pseudo-random numbers**. Each of these is generated from the previous one by an algorithm which makes numbers that look random enough, but aren't suitable for very secure applications like encrypting financial data, since it's possible that a sufficiently clever person could figure out the pattern and thus work out the randomly-generated password used to encrypt your data.

A better approach for secure data like this is to generate random numbers from physical processes such as flipping a coin (or, for example, measuring noise in the form of tiny fluctuations in voltage from a power supply). In theory these can't be guessed, but can be very slow to generate, and some care is needed to make sure these numbers are in fact "random enough". We'll demonstrate this by generating random numbers in Scratch from the volume level detected by a microphone.

Ingredients

A microphone! (connected to a computer running Scratch)

How we're making the random number

The simplest way to create a random number from the volume level would be to just use the volume level as the random number! However, this might not produce very evenly-distributed random numbers: for example, in a crowded room, the volume might always be between 20 and 30. This makes it easy to guess the random number: someone trying to guess would know that they don't need to bother guessing anything below 20 or anything above 30, which makes it ten times more likely that they might get the right answer!

Binary

What we'll do instead is make a random binary number out of several volume measurements. As an example, let's say we take eight volume measurements:

12	**14**	**15**	**8**	**6**	**8**	**12**	**13**

Safety: Please note that you use these resources at your own risk. Correct use of some components requires care.

THE UNIVERSITY OF WARWICK

Produced by Rob Sandford (robert.sandford@warwick.ac.uk)
Technology Volunteers: go.warwick.ac.uk/techvolunteers
Scratch Resources: go.warwick.ac.uk/scratchresources

Warwick Volunteers

Binary

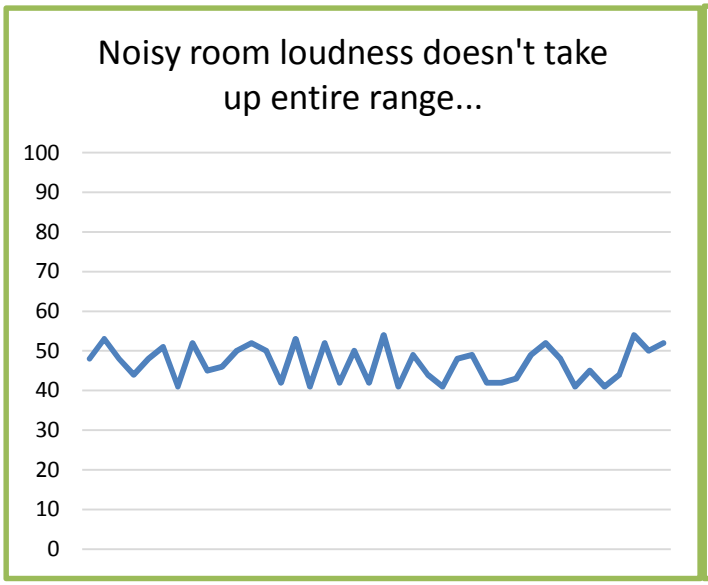
We would then use these numbers *modulo 2* (sometimes called *mod 2)* – that is, replace them with a 1 if the number is odd or a 0 if the number is even:

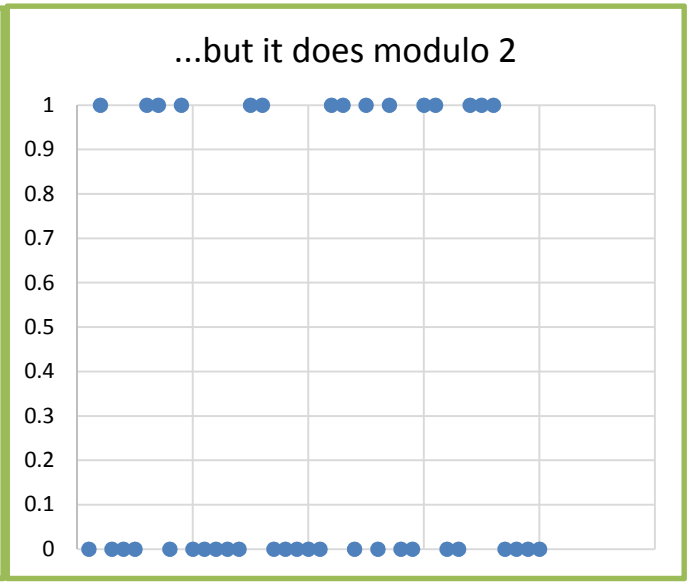
0	**0**	**1**	**0**	**0**	**0**	**0**	**1**

This looks a lot like a binary number! In other words, to get the random number, we'll multiply each column by the corresponding power of 2:

Value	$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Digit	0	0	1	0	0	0	0	1

So in this simple example, the random number would be 1×32 + 1×1 = 33. By just looking at whether the volume measurement is odd or even, we effectively "magnify" small fluctuations in the volume, which should make the random numbers more spread-out.

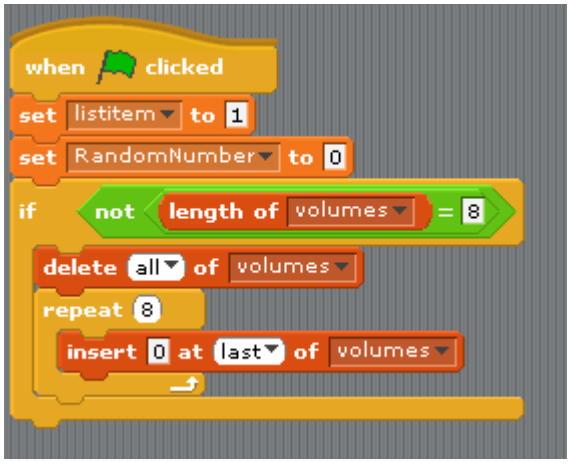




Step 1: A List

So, the first thing we need is a list of eight volume levels. We'll make a list called **volumes** and a variable called **listitem** to tell us which item of the list we're using. We'll also need a variable called **RandomNumber** which will hold the random number.

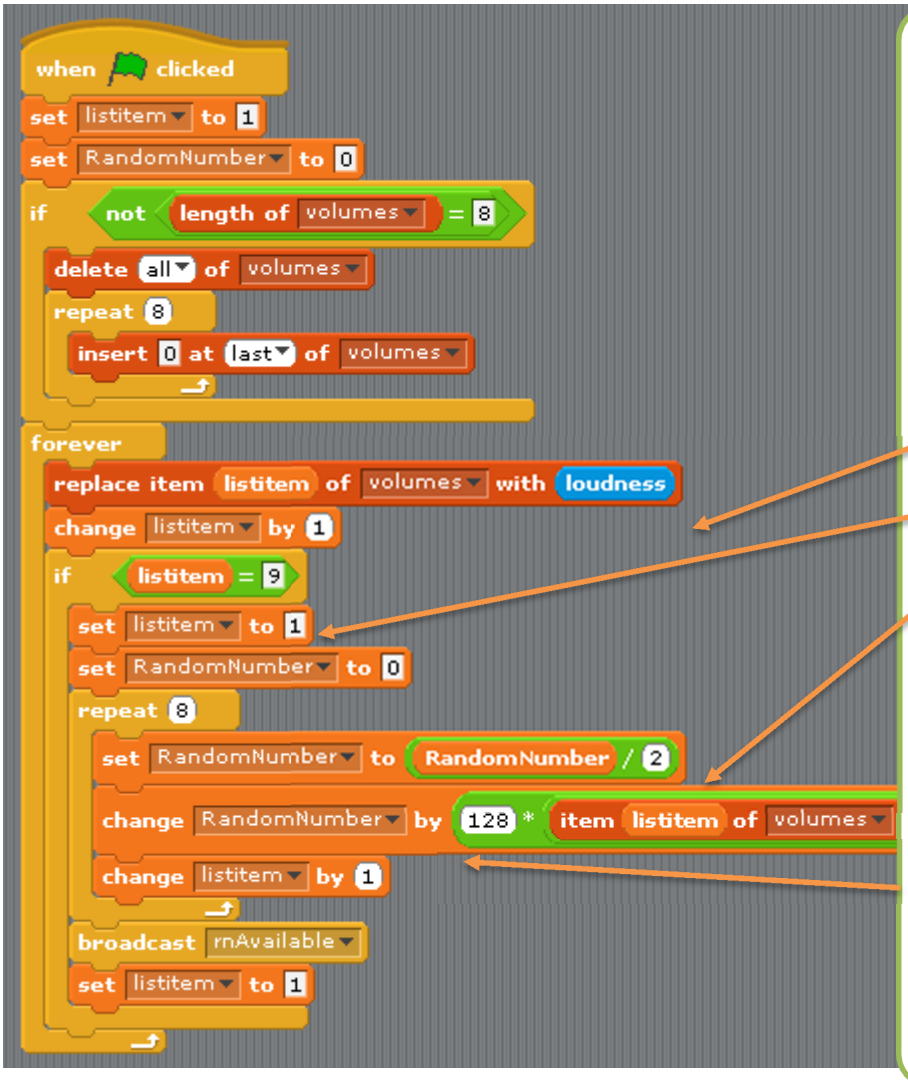
The first thing we want to happen when the program starts is to make sure the list of volumes has enough room for all eight binary digits. If it's the wrong length, we'll delete it and fill it with zeros.



THE UNIVERSITY OF WARWICK

Produced by Rob Sandford (robert.sandford@warwick.ac.uk)
Technology Volunteers: go.warwick.ac.uk/techvolunteers
Scratch Resources: go.warwick.ac.uk/scratchresources

Warwick Volunteers



Step 2: Random Numbers Forever

We then want to generate random numbers forever. To do this, we repeatedly store the current loudness mod 2 (this simply means the remainder when divided by 2) in **volumes** at the position indicated by **listitem**, then increase **listitem** by 1. When **listitem** reaches 9, we'll reset it back to 1, and then turn the list of volumes into the random number by multiplying each binary digit by the correct power of two.

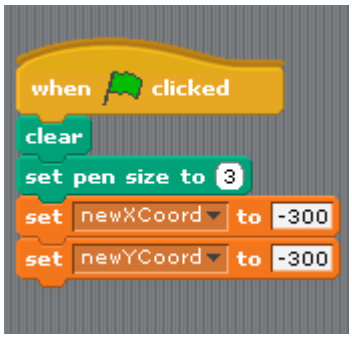
(Scratch doesn't let you use a "power of" symbol, so instead we need to multiply each digit by two the correct number of times – or, equivalently, multiply the digit's value by 128 and then shift the other binary digits to the right by dividing them all by two. It's a bit more complicated, but it works!)

Step 3: Displaying the random number

Of course, just making a random number isn't very interesting. It's useful to plot these random numbers so we can get an idea of how evenly distributed they are. There are of course more complicated ways of determining this accurately, but this one makes it easiest to see what's going on.

To start, we'll need a sprite. I've used an X shape, but any sprite will do – its job is just to move around and draw a dot at the right coordinates! We'll also need two new variables, **newXCoord** and **newYCoord** – these will both be set to random numbers and tell the sprite where to move next!

To begin, when the program starts, we'll want to clear any pen marks already on the screen. We'll also set the pen size to be large enough to see, and set **newXCoord** and **newYCoord** to -300 – this number will never appear as a random number, so it's a good placeholder to tell the sprite that the coordinates aren't ready yet.

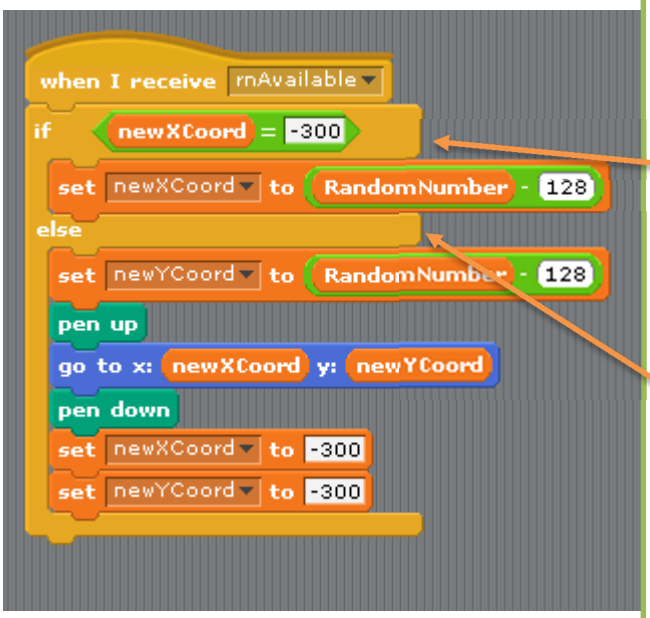


THE UNIVERSITY OF WARWICK

Produced by Rob Sandford (robert.sandford@warwick.ac.uk)
Technology Volunteers: go.warwick.ac.uk/techvolunteers
Scratch Resources: go.warwick.ac.uk/scratchresources

Warwick Volunteers

Step 4: Drawing a point



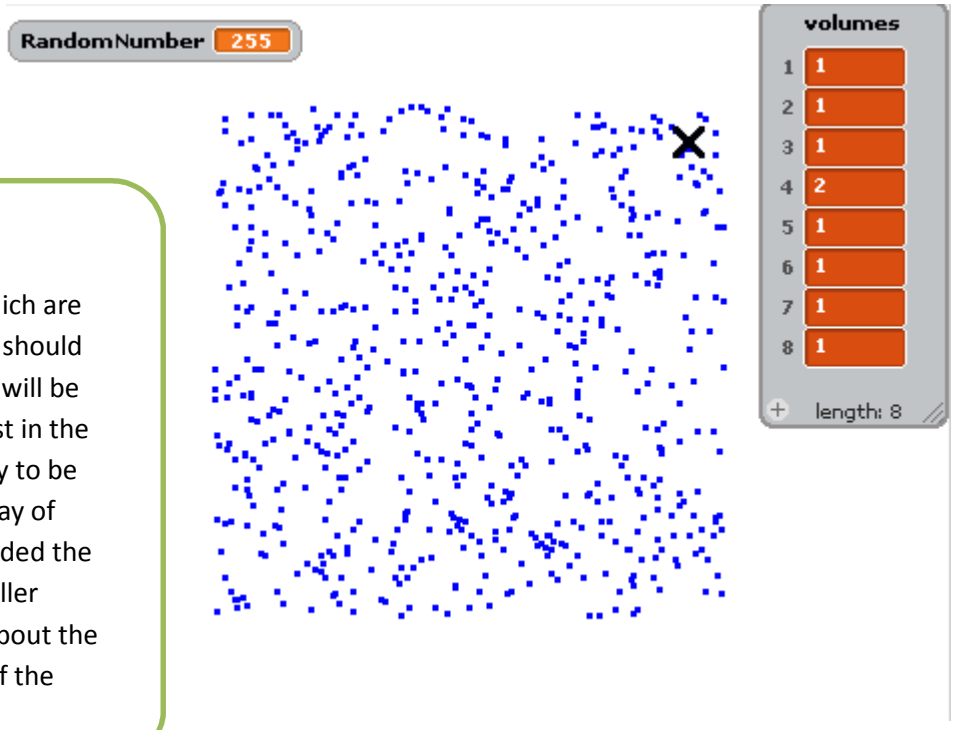
We'll then add a signal to the random number generator. This signal, **rnAvailable**, will tell the sprite when a random number is available. (See Step 2 to see when to send this signal)

When the sprite receives **rnAvailable**, it'll check if its new x-coordinate is -300, meaning that it doesn't yet know the next x-coordinate. If so, it'll set **newXCoord** to the random number, and then wait to receive the signal again.

On the other hand, if it does already know the x-coordinate that it wants to move to next, it should set **newYCoord** to the random number. It then lifts the pen, moves to the new coordinates, and drops the pen down again to leave behind a dot at these new random coordinates. It then resets both coordinates to -300, and the cycle begins again!

What you should see

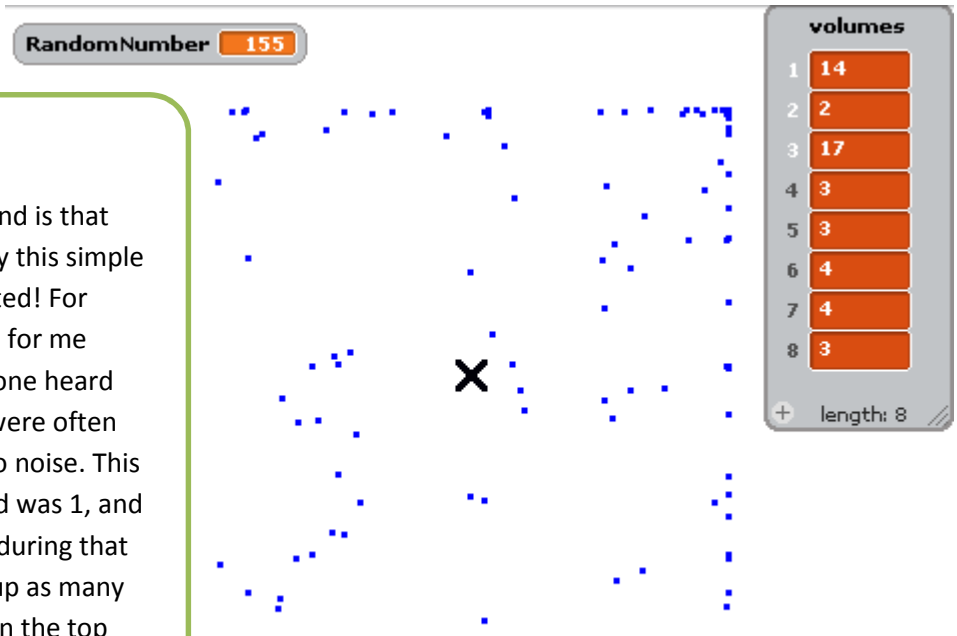
In theory, random numbers which are perfectly uniformly distributed should look something like this. There will be clumps and dots that are almost in the same place, but they're unlikely to be very large – a more accurate way of looking at this is that if you divided the square up into equal-sized smaller squares, you'd expect to find about the same number of dots in each of the smaller squares.



THE UNIVERSITY OF WARWICK

Produced by Rob Sandford (robert.sandford@warwick.ac.uk)
Technology Volunteers: go.warwick.ac.uk/techvolunteers
Scratch Resources: go.warwick.ac.uk/scratchresources

Warwick Volunteers



What you'll probably see

In practice, what you'll probably find is that the random numbers generated by this simple program aren't uniformly distributed! For example, this is what it looked like for me when the sound that the microphone heard was typing on a keyboard. There were often periods of time when there was no noise. This meant that the loudness measured was 1, and so the resulting random numbers during that time were all equal. This showed up as many random numbers being gathered in the top right corner.

Depending on what you're using the random numbers for, this might be okay, or this might mean they would need more processing or to be generated in a different way in order to be useful. These are the same questions that cryptographers and other people who want to generate random numbers have to answer.

Some ideas for other things you could try

- Try exposing the microphone to different sounds. What produces the most uniform-looking random numbers – talking? music? traffic sounds?
- Try adding a delay to the volume measurement loop – not necessarily a big one, maybe between 0.1 and 0.5 seconds. Does this seem to make a difference to the random number plot? If so, why do you think this might be?
- How about comparing the results you get to the pseudorandom numbers that Scratch produces? The simplest way to do this is to change the blocks where we set newXCoord and newYCoord – instead of RandomNumber - 128, you'll want to use the "pick random" block.

THE UNIVERSITY OF WARWICK

Produced by Rob Sandford (robert.sandford@warwick.ac.uk)
Technology Volunteers: go.warwick.ac.uk/techvolunteers
Scratch Resources: go.warwick.ac.uk/scratchresources

Warwick Volunteers