

A Tutorial on Algorithmic Music Generation in Reaktor



By Peter Dines

peterdines@gmail.com

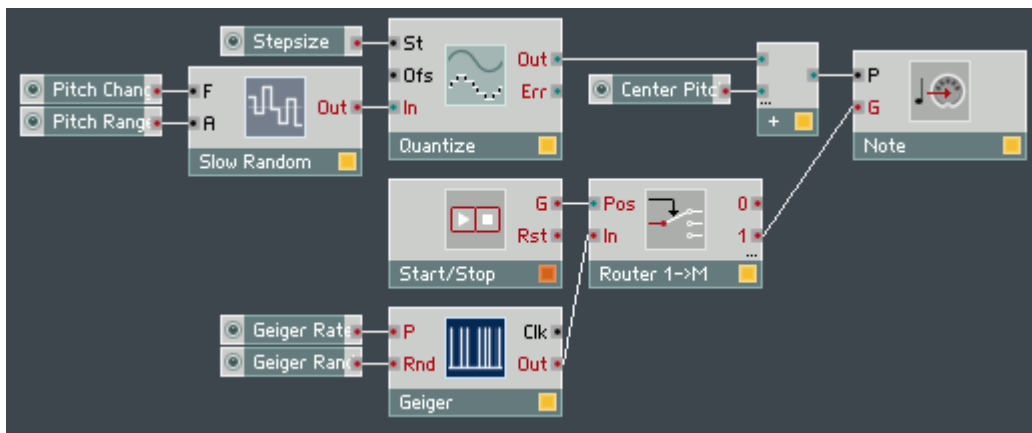
<http://reaktortips.com/>

Everyone loves randomly generated music, right? Heh heh, well, perhaps not. But if you're the kind of person who uses Reaktor, chances are it's a subject that interests you.

In this tutorial we're going to cover some of the basic nuts and bolts of the Reaktor toolkit that you'll need to create unpredictable music machines that play themselves.

First, let's load the random01.ens and see what it does and why. There's a Synchronizer instrument, cut out of the Gobox instrument from the factory library, a Perc. FM 4 synth driven by the SimpleAleatoric instrument, and a standard Mixer instrument to combine the audio output. Hit the space bar to run the clock and create sound. You should hear a basic beat and a melody. It's not terribly musical sounding, is it? As a matter of fact it's sort of bletcherous. There are reasons for that. The main one is that it's quite rudimentary. This is the single celled amoeba of Reaktor algorithmic music generation. It doesn't even have an eye spot yet. Stop the clock before it drives you up a wall and let's dissect it.

Go into the structure of the SimpleAleatoric_01 instrument and this is what you'll see:



Just as every cell in your body needs a source of energy, so every generative instrument in Reaktor needs a source of energy and motion. In this instrument, the Slow Random and Geiger modules are the driving force.

The Slow Random module is a source of random values. We're using that here to vary the pitch of the notes being output to the Perc. FM 4 synth. There are two controls here – one for the frequency in Hz, and one for the +/- range of the values. The Pitch Range is the range in semitones by which the pitch can vary. Since the Slow Random produces fractional values like 3.852, we feed it into a Quantize module, which we can use to force it to whole number values like 1, 2, 3, etc. These integer values can then be used as MIDI note pitches. A Stepsize control allows us to adjust the quantize value to various numbers to produce various melodic effects. They'll always sound a bit odd, though, because they'll be symmetric scales (with a constant interval between pitches) rather than the blend of whole tones and semitones we're used to in Western music. This is one of the areas that will improve as our musical amoeba evolves.

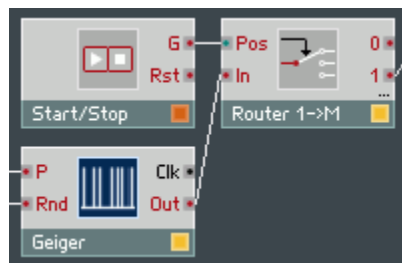
Tip: if you hover the mouse cursor over a wire in Reaktor, you will see a tooltip with the values carried over that wire. Do this with the input and output of the Quantize module to see how it affects the values passing through it.

Up to this point we're producing pitches that vary between -36 and +36, and usually less than that, so we add a value to keep the pitches in an audible range. The Center Pitch control determines this value. 60 is a good default value, but you can change this to make the average pitch higher or lower. From

here the pitch goes to the P input on a MIDI Note out module, This is the way the generated note events get to another instrument. The Note module sends out a note whenever there's a trigger input at the G input, and uses whatever value happens to be at the P input at the time the trigger is received.

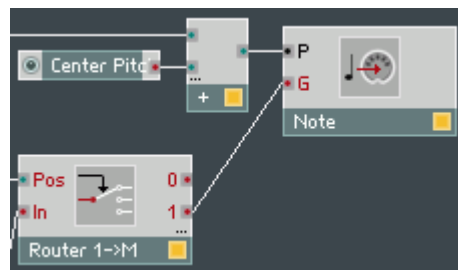
The Geiger module provides the trigger. There are two controls on the input of the Geiger. One controls the rate at which it fires events, and the other controls how random or steady the trigger rate is. Run the clock and play with these controls to get an idea of how they work. Since the Geiger module produces signals constantly, it's connected to a Router module to prevent it from running when the clock is stopped. The Router's Pos control input comes from a Start / Stop module. The value of the G output on the Start / Stop is zero when the clock is stopped and one when the clock is running. So if the clock is stopped, the Geiger's triggers go to the unconnected 0 output of the router, and if the clock is running, they go to the connected "1" output of the router.

Tip: if you right click on the input of a module and choose "create control", Reaktor will insert a control with a range of values typical for that type of input.

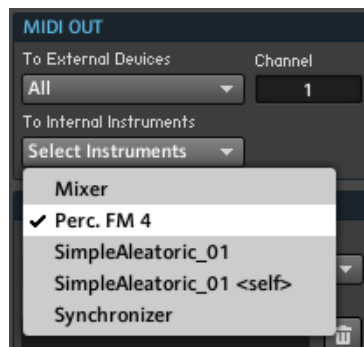


Tip: You can do almost the same thing by using a Multiply module instead of a Router, but I want you to get used to using Routers.

From the Router, the Geiger's trigger signals go to the G input on a MIDI Note module.



What happens next? If you look at the Out dropdown menu in the connect tab properties of the SimpleAleatoric instrument, you will see that its MIDI output is routed to the Perc. FM 4 instrument. It is also possible to route output to an externally connected hardware synth using the Ext. MIDI menu.

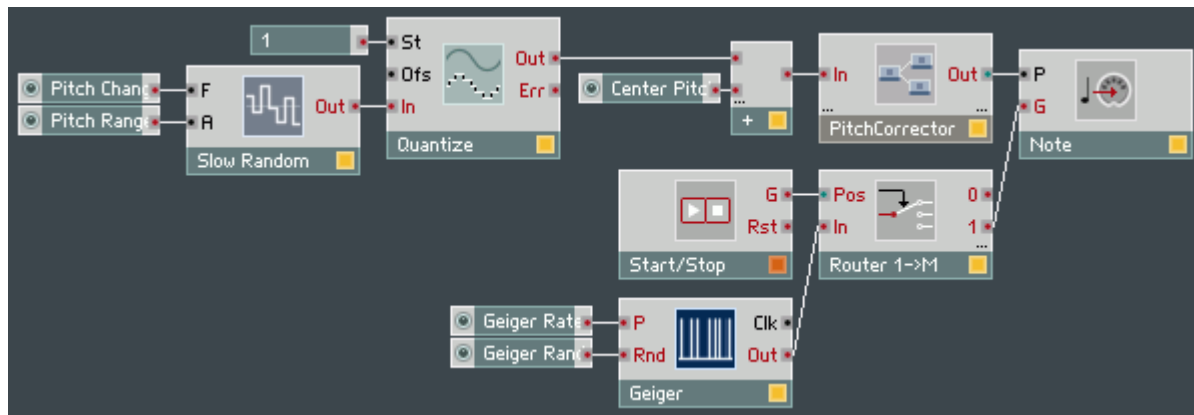


So to summarize, we have parts of the instrument that are sources of activity – the Slow Random and Geiger modules; we have parts that modify and channel the sources of activity – the Quantize, Add and Router modules, and we have an output for the instrument – the Note module. Now how do we improve the cacophony it creates? I don't know about you but the thing that bugs me the most about this instrument is the melodic quality. So let's see how we can improve it.

Open the random02 ensemble and run the clock. It sounds maybe a little bit more musical now, doesn't it? Instead of sounding like a cat walking across a piano keyboard, it now sounds like a cat walking across only the white keys of a piano keyboard.

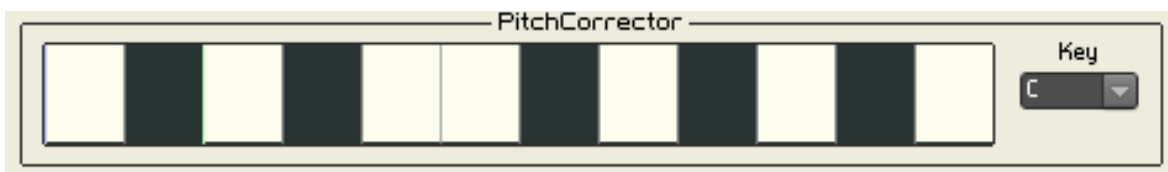
Tip: to keep a cat off your piano keyboard, close the lid.

Take a look at the structure of the SimpleAleatoric_02 instrument. Now there's an extra macro called PitchCorrector inserted in the structure before the Note module.



Notice that we have retained the Quantize module, but now its Stepsize is a fixed value of 1. This quantizes the values coming from the Slow Random to whole numbers, which can be used to represent the semitones in a chromatic scale.

This is what the PitchCorrector looks like on the instrument panel:

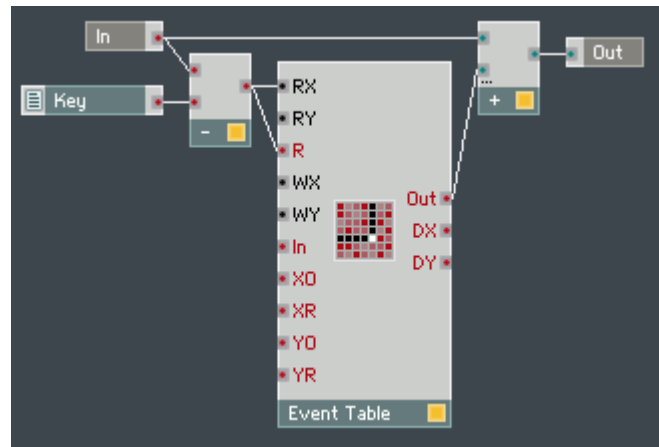


Does the pattern of light and dark bars look familiar? If you think it looks like piano keys, you're right. And there's a reason for that. The purpose of the PitchCorrector is to change the pitches coming from the Quantizer into the pitches of a Western [diatonic scale](#). The Key dropdown menu lets you select one of the twelve diatonic scales. If you want a minor scale, select its relative major – so to obtain E minor, you'd select G (major) in the dropdown menu. To get C Dorian, select Bb, and so on.

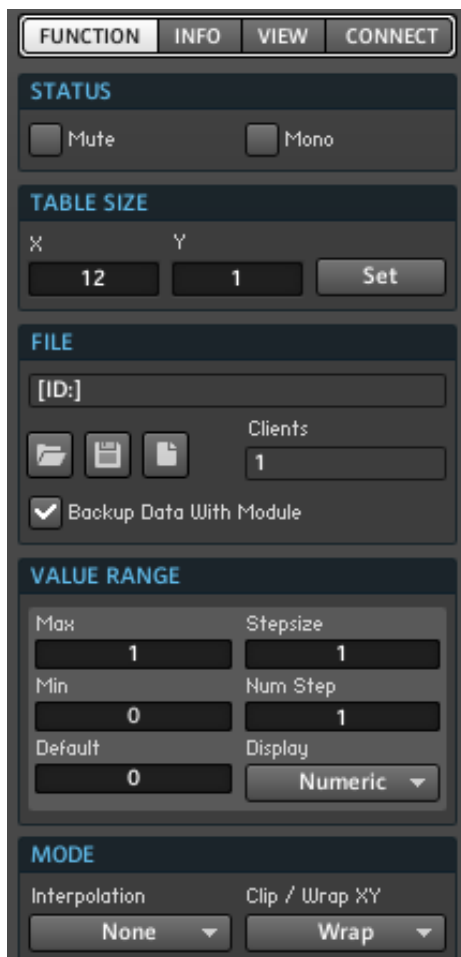
Tip: Plato thought that Dorian and Phrygian modes would toughen up a soldier and that the Lydian and Ionian modes would weaken one. Don't be a wimp. Choose a tough mode. (Then again, Plato also believed in Atlantis.)

The heart of the PitchCorrector macro is a Reaktor Event Table. Drill down into the PitchCorrector.

Incoming pitches are sent to the RX and R inputs of the Event Table. (ignore the Subtract module for a moment.) The value at RX selects the read position in the table, and any event at R sends the value it finds there to the Out port of the Event Table. From there it is added to the pitch, making it a semitone sharp if the table contains a 1 at that position.



The table is set up so it's 12 units long. It's also set to Wrap, which is the default. You can check this out in the table properties.



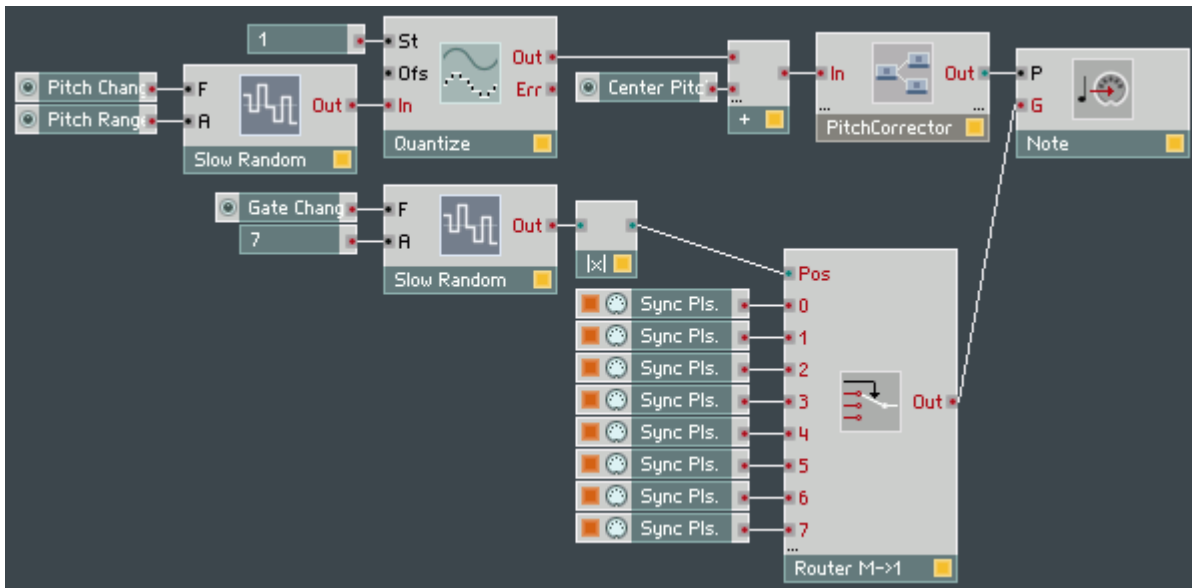
This macro works by exploiting two features in Reaktor: **1)** that every C pitch is divisible by 12 (except for C0, which is zero), and **2)** the Wrap feature in the Event Table's properties. The Wrap works like this: a value of zero selects position zero; a value of one selects position one, and so on. But when you get higher than the length of the table, the read position wraps around to the start. So a value of 12 at the RX input selects the first position again. So will 24, 36, 48, 60, etc. – anything divisible by 12.

Tip: remember that the first position in a Reaktor table is counted as a zero, not a one. Reaktor tables are like arrays in programming. This table counts from 0 to 11, not from 1 to 12.

Let's examine what would happen in the key of C major. A middle C has a value of 60. With wrap turned on, that's going to select the value at the first position of the table. Since the value at that position is zero, nothing will be added to the pitch. C into the macro, C out of the macro. But a C# will read the second position, which contains a value of one. The value of one will be added to the incoming C#, "correcting" it to a D. C# into the macro, D out. The way the table colors are set up in this instrument, values of zero look white in the table and values of one look black. That's what causes the piano key appearance.

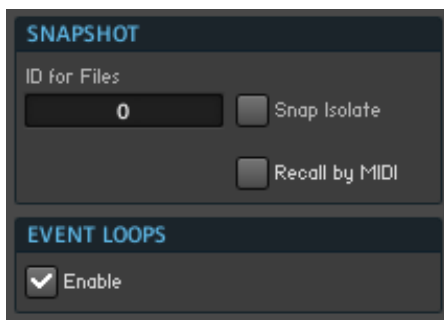
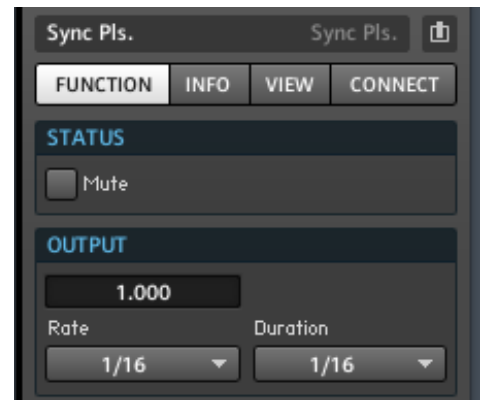
Now, what about the Key menu and Subtract module? If you want to have the SimpleAleatoric_02 instrument play in the key of C# major, you have to subtract 1 from the read position value going to the table's RX input so that the relationship between the notes – the pattern of tones and semitones – remains the same. With the Key menu set to C#, an incoming C# note will trigger a read operation at the first position in the table rather than the second position. The C# will not be corrected to D, as it was when we wanted to play in C major. However, an incoming D will trigger a read operation at the second position, adding a semitone to the D and correcting it to D# - which is one of the notes in a C# major scale. Each entry in the Key menu will subtract an appropriate number of notes from the RX read position. You can see this for yourself in the properties for the Key menu.

Now that we've ratcheted up the melodic intelligence of the instrument a bit, it's time to work on the rhythm. Open up the random03 ensemble and dig into the SimpleAleatoric_03 instrument.



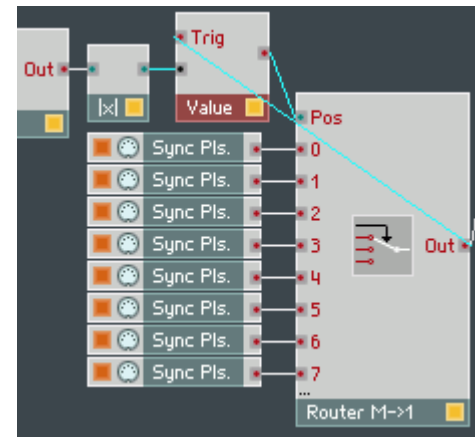
As you can see, the Geiger module is gone, replaced by a few other modules. Run the clock and give it a listen. The notes have a closer relationship to the steady drumbeat now, don't they? There are now moments where the melody sounds almost deliberate, or even playful. That's because we're using Sync Pls. modules, which trigger at a steady rate relative to Reaktor's master clock. It's still not going to win a Grammy for songwriting, but we're getting closer to something listenable.

There are seven Sync Pls. modules connected to a Router module. Each Sync Pls. has a different firing rate which is set in the properties. The Pos input on the Router determines which Sync Pls is selected at any given moment. And the Pos input is being fed by another Slow Random module. Why is there a |X| module in between the Slow Random and that Pos input? It gives the absolute value of the events generated by the Slow Random. So if the Slow Random produces a 5, the |X| module puts out a 5. But if the Slow Random produces a -2, the |X| puts out a 2. Thus the value is always in the range of zero to seven.



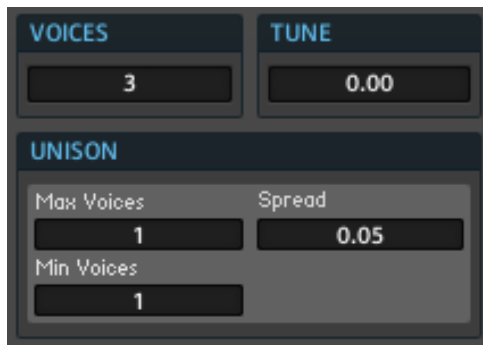
One more quick and easy improvement we can make to the rhythm is to tighten up the timing of the router changing position to let through a different Sync Pls. First thing we have to do is right click on the background of the SimpleAleatoric_03 instrument and choose properties. At the bottom of the Properties dialog, tick the "event loops enable" checkbox. Normally, Reaktor warns you if there are feedback loops in your event structures. However, they're safe to add if you know what you're doing. If you don't, and you do something stupid like create an infinite loop, Reaktor can and will shut itself down completely to avoid locking up your computer. So always save your work before you start tinkering with event loops. Event loops should be enabled globally in Reaktor's system preferences. Then they can be enabled selectively per instrument or per macro.

Once you have enabled event loops in the instrument, insert a Value module, connect the output of the router to its Trig port, and connect the output of the |X| module to its input port. Then connect the output of the Value module to the Pos input on the Router module. This module latches the value coming from |X| and prevents it from going through until it receives a signal at the Trig input. When it receives a trigger, it sends through whatever value is currently at the input. In other words, it doesn't remember the “queue” of signals that it received up to that point.



The rhythm of the melody should be tighter now, with fewer jerky little trills when a different Sync Pls. is selected by the router. In effect, it's listening to itself. Controlling feedback loops like this is one of the secrets to making a machine that not only plays itself but sounds like it knows what it's doing.

If you had trouble turning on event loops and hooking up the Value module properly, no worries – open up the random04 ensemble, which includes the changes we just made plus one more: it's polyphonic.



Look in the properties of the SimpleAleatoric_04 instrument. This one has three voices, not just one like the earlier versions. Remember that each module in the instrument has a yellow lamp in its corner – that means it's capable of polyphonic processing. So each connection and each module in this instrument is carrying three parallel data channels. Imagine there

Tip: a module or macro can be set to monophonic or polyphonic in its properties or its right click context menu.

were three parallel layers of the structure stacked one atop the other – that's what's happening here. The Slow Random modules act like three separate Slow Randoms, producing a different value for each voice. Thinking of Reaktor polyphony in this way – as parallel data channels, rather than as merely how many voices a synth can play – will help you go on to build interesting and intelligent machines.

Hit the space bar to start the clock and you should hear a more complex melody, this time with harmonies. You can select a different number of voices, but three seems to be a nice middle ground for this instrument. If you route the midi to another synth, which you can easily do with the routing menus in the instrument header, you may want only one voice, or four.

Here are some ideas for ways to modify and expand the SimpleAleatoric instrument:

- Drive the pitch or rhythm with a sine LFO whose speed and amplitude are controlled by other sine LFOs. Try other LFO shapes too.
- Use event tables to draw curves and use a clock to iterate through them to produce values.
- Use an event randomizer and an absolute value module to vary the velocity of the notes produced.
- Use a delay module to create “MIDI echo” in the notes you send to a synth.
- Extend the range of the Event Table in the PitchCorrector macro so you can program altered scales like harmonic minor
- Use Counter and Compare modules to change a value or fire an event after X number of F# notes (for example) have been produced by the instrument

You should now have a solid foundation to explore these and many more ideas. Happy building!

Peter Dines
peterdines@gmail.com

Visit the <http://reaktortips.com/> blog for news and updates