sequence beams

Sequence beams are the white studless beams that link blocks together onscreen. The order of blocks along the sequence beam controls the order in which they execute: The first block connected to the start symbol executes first; then (usually when the first block is completely finished) the next block along that sequence beam executes; then the third one, and so on. Sequence beams are just a visual way to represent the program flow, not unlike the arrows in a flowchart, as you can see in Figure 2-4.

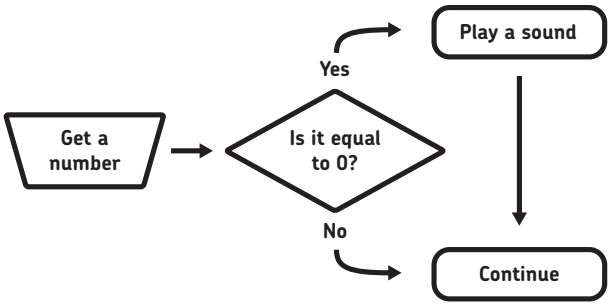


Figure 2-4: A simple flowchart for the program shown in Figure 2-3

branching and sequences

Like a flowchart, these "execution sequences" can branch, but they branch in two very different ways. The first is when a choice has to be made: Should the execution follow this path, or that one? In a text language this is done with something like an if-then or a switch-case command, while in NXT-G the Switch block takes over this role (more on that in a second).

And NXT-G offers another very different, very powerful way to make the execution proceed: You could have it go down two different branches simultaneously. This is *parallel programming*: using two or more parallel sequences to accomplish two things at once.

In NXT-G, parallel programming is as natural as breathing. Once you have laid down one sequence (or even a portion of that sequence), you can start a second sequence by dropping a block somewhere below it, *not* connected to the existing sequence beam (actually, it could be below it, above it, or anywhere else, as long as it doesn't link up to the sequence beam). Such an "orphan" block is normally ignored by NXT-G and not incorporated into your finished program.

Now, to branch a sequence beam and link in this first block of a parallel sequence, hover the arrow cursor over the existing sequence beam at the point you want it to branch, and press and hold the SHIFT key. You can now click and drag out a new branch of the sequence beam to the start of your orphan block, and that's it—you've started a new parallel sequence. Any blocks you drop behind this newly connected block will obediently join the rank and file of the second sequence. You can branch a new sequence from anywhere: the start symbol, midway along a sequence beam, or even within a Loop or Switch (although that's a bit trickier). Figure 2-5 shows a sample three-pane sequence.

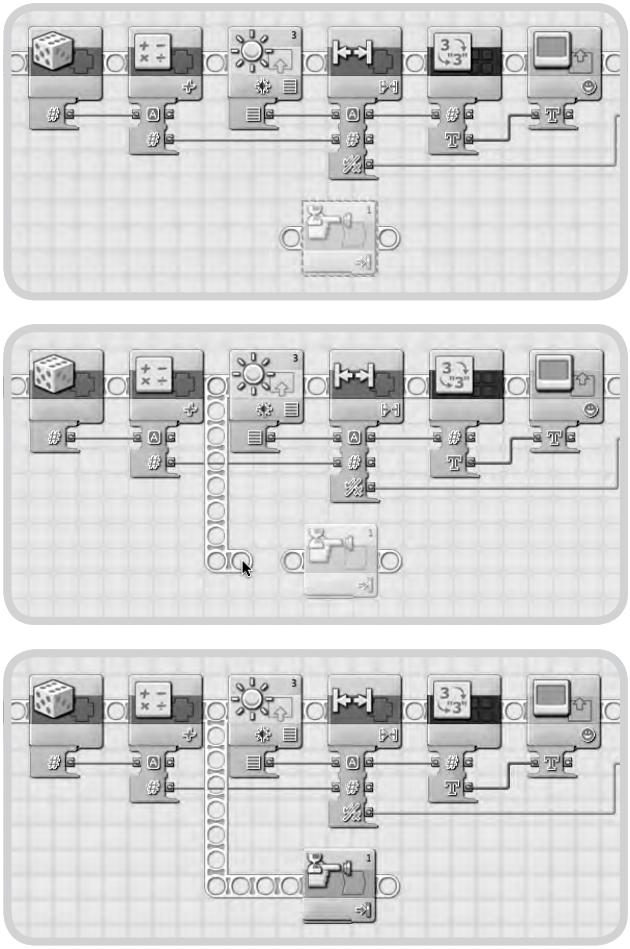


Figure 2-5: A three-pane sequence of creating a parallel sequence beam

MULTITASKING: NO SUCH THING AS A FREE LUNCH

Don't think that running two sequences in parallel makes things twice as fast. Just like you, the NXT can't really do multiple tasks at once (there's only one primary processor trying to execute your code). Behind the scenes it rapidly switches between all the currently executing sequences: It first executes a little bit of code from this sequence; it then skips down and executes a little bit of code from the next parallel sequence, and so on. In effect, running two sequences in parallel means that each sequence is running significantly slower than if the processor were running only one. There are still times when these parallel sequences are handy (for instance, watching a sensor for something to change while at the same time calculating some math or moving the robot), but it doesn't mean your NXT is working "twice as fast."

blocks and structures

Blocks and structures are the real workhorses of NXT-G. They are specialized chunks of code that do something specific, such as play a sound, control a motor, get the current reading from a sensor, and so on.

Almost any time you place a block or a structure, you need to set up its various limits and conditions from the configuration panel, such as what port a sensor is connected to, how long to run a motor, and so on—and there can be a lot of things to configure.

blocks

Of the two, *blocks* are the simplest; they're just one-square icons that accomplish something specific. Program execution usually halts while a block is executing; that is, the next block on the sequence doesn't start doing its thing until the previous "upstream" block has completely finished. But that's not always the case: Some blocks can start actions and then let the sequence continue executing, even while these actions are still taking place. A good example of such a block is the Sound block, which enables you to check or uncheck the box next to the words *Wait for completion*.

cloning blocks

Re-editing the configuration panel every time you drop a new block gets old fast. Every block has only one default state when dropped from the palette, but that's not much help if you want to repeat a block that is set with something other than the default state.

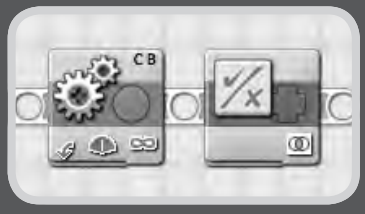
To repeat an existing block you can clone it. If you ALT-drag an existing block (OPTION-drag on the Mac), the environment makes a "clone" of the block you selected—copying every setting from its configuration pane into the new one, and even imitating the data plugs that are currently displayed. This is another wonderful feature inherited from NXT-G's "parent language" LabVIEW, and it's great that the developers left such things in to help the advanced user.

loop and switch structures

The Loop and Switch structures function almost like composite blocks, or blocks that contain other blocks. When they are first placed they do nothing, because there is nothing inside them to do; you must drop blocks into them to set up their sequences.

CONFIGURATION HINTS

Notice that the block icons give you hints about how the configurations are set up. For instance, a Move block set to *unlimited* displays a little infinity symbol in its lower-right corner, telling you how it is configured without your having to view the configuration pane. Another great example is the mini–Venn diagram on the Logic block, which changes to represent the sort of logical operation that it is configured to perform (in this case a logical AND).



connection [0]. To further complicate matters, a message going to an NXT can be placed in one of those 10 mailboxes to try to help keep things organized. Figure 7-11 shows a simple "map" of a BT network involving four NXTs.

An NXT is determined to be a master if it initiates a BT connection, and its identity as master cannot be changed while the program is running.

This hints at several important limitations under this system:

- There is no broadcast mode; if an NXT needs to send a message to every NXT in the network, it must do so to only one NXT at a time. (If the sending NXT is a slave, it must send a message "up" to the master, which then echoes the message "down" to the other NXT slaves, one at a time.)
- An NXT can never be both a master and a slave, because a slave NXT has only one connection: back to the BT master in the network.
- If you inspect the NXT-G blocks or the menus on the NXT brick, you'll see only four listed connections, [0] through [3]. This means that under NXT-G, a BT network will interconnect with at most four NXTs: the master with three slaves beneath it.

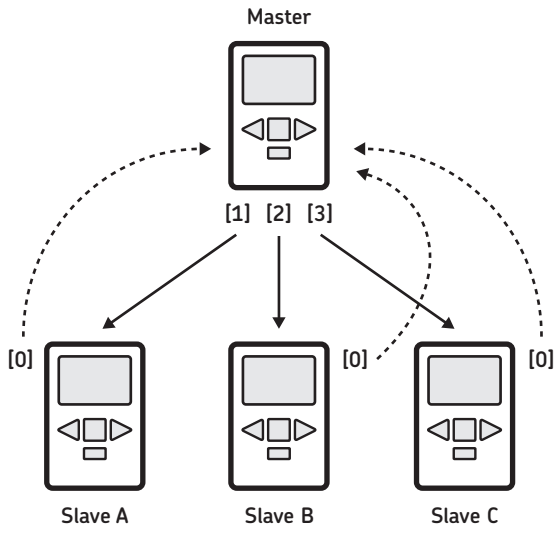


Figure 7-11: A diagram of the specific connections in a typical four-NXT network

setting up an inter-NXT connection

Setting up an inter-NXT connection is simple. To do so, all you need are (at least) two NXTs with BT turned on and visible. To set up a connection for the first time on the NXT that will ultimately be the BT master in this mini-network, do the following:

1. Select **Search** under the BT menu and wait while it tries to discover BT-capable devices in the area. After a while, it should ask you to select the device you want to connect to from a list.
2. Select the NXT you want to network with and then assign it to connection [1], [2], or [3]. (Connection [0] is not available because this is a downlink from master to slave, and connection [0] is reserved for a slave uplink connection.) For this example, use connection **[1]**.
3. If this is the first time these two NXTs have been connected, you have to go through the passkey windows on each to complete the connection. As with the computer, once you have walked through the passkey window to establish a connection for the first time, the NXT will remember it for you. For any future connections you should be able to open the Contacts menu, select the NXT you want to connect to, and then assign a connection to it. It's that easy.

NOTE **If you play with BT connections a lot (especially if you are using a NXT-to-PC BT connection to download and test NXT-to-NXT BT programs—as discussed in Chapter 8), you might find that the computer reports that it can't establish a connection. One reason for this is that only one connection is allowed for any BT slave (the one back to the master). Therefore, if you tell the computer to connect to an NXT that is currently configured in an existing network, chances are something will act funny. Either the computer will not be able to establish a connection, or the previous network will be torn down by the NXT being ripped out of it when the computer demands a connection. To avoid confusion, make sure that the device in question is not already locked into some other BT network before trying to establish a BT connection. If it is, disconnect it first. No single NXT can be part of two networks simultaneously.**

WAITING FOR DISCOVERY

Note that the wait while the NXT tries to discover other BT devices depends in part on just how many BT-visible devices are in range. During a scan, the NXT requests and gathers information from any BT device it can find, which can be a lot of information if there are a lot of devices. I tried this once at a large conference, and was surprised when several *minutes* later, the NXT returned with a list of something like 20 "unknown devices," presumably BT-equipped cell phones and so on. It seems a lot of tech folks leave their BT devices on and visible!

communicating between NXTs

After your NXTs are connected, you can send information from one NXT to the other, as long as you remember which connection (address) to send (mail) the information to. For example, to give a friend a new sound or program file that is on your NXT, you could set up a BT connection to your friend's NXT, select the file you want to transfer, choose *Send* (denoted by the flying letter icon), and then select the connection over which you want to send the file. The two NXTs should then obediently transfer the file, so your friend ends up with a copy as well. Now you have a way to transfer custom sound files or program files without any need to haul a computer around!

BT messaging under program control

After two or more NXTs are connected, programs running on them can exchange information over the network via NXT-G Send and Receive Message blocks. While the network connection must be set up manually from the NXT menu system, all the message sending and receiving can be handled within a running program.

To send a BT message, drop a Send Message block onto the program sequence beam and configure it. There are three parts to the configuration pane:

- The connection used to send the message out on ([0] for a slave sending a message back to the master NXT; or [1], [2], or [3] for a master NXT sending to one of the three slaves it might be connected to)
- What the message is (you need to specify both its type—Text, Number, or Logic—and its value)
- Which mailbox to place the message in on the receiving NXT

All these parts can be set in the configuration panel, or wired in from some other part of the program through the block's data hub. You can send logic values (*true* or *false*); numbers between –2,147,483,648 and 2,147,483,647 (for those of you with two fingers, that's 2^{32} different numbers, as opposed to the RCX limit of 2^8); or strings up to 58 symbols (like this sentence)—a great step up from the old IR messaging system.

receiving messages

Sending a message isn't very useful unless you have a way to retrieve and read incoming messages. To do this, you'll use the Receive Message block.

RBT VS. RXE

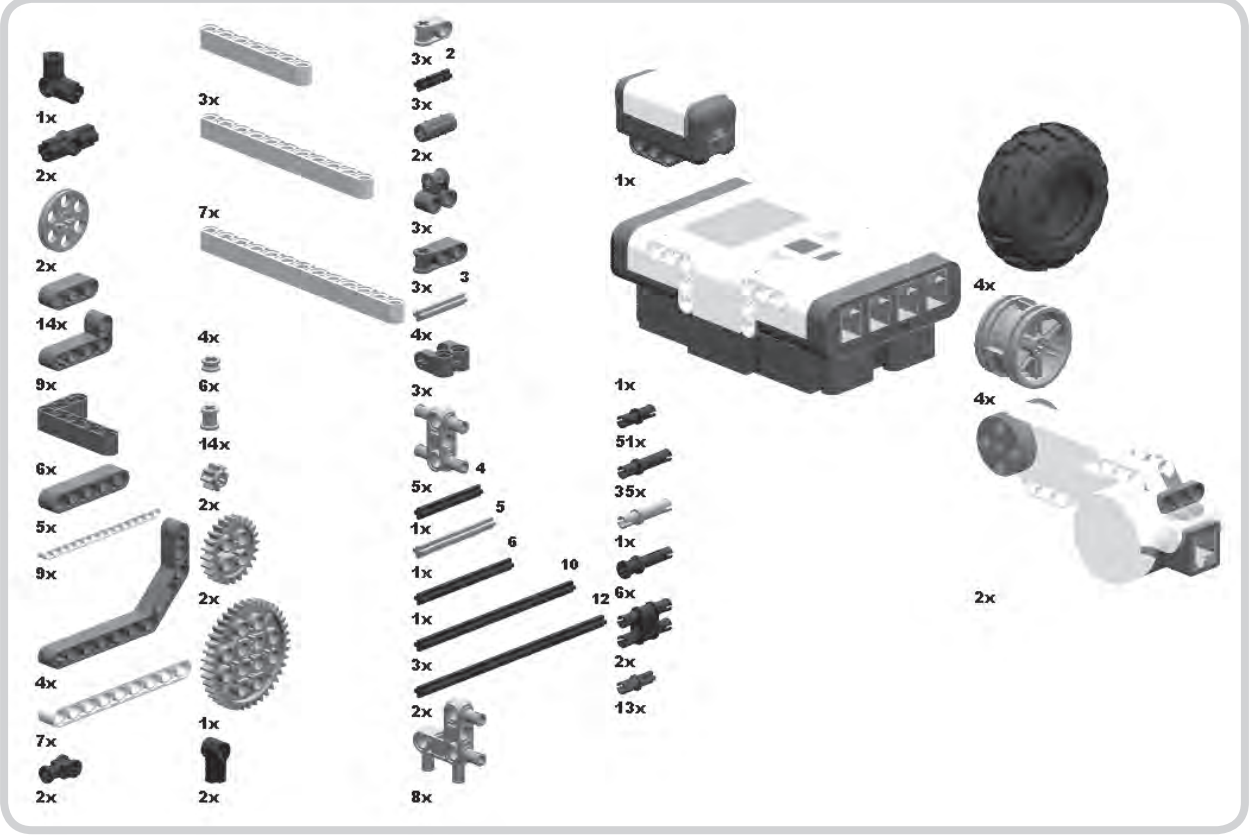
There is a difference between a NXT-G RBT file on the computer and the executable (RXE) file on the NXT brick. The RBT file (with the .rbt extension) is what you can open and edit in the NXT-G environment, while the RXE file (with the .rxe extension) is the actual machine code that the NXT can use. The RXE file is the file that is mailed to your friend's NXT, but there is no way to translate an RXE file into an editable RBT file. So although BT mailing enables you to share *working* programs, it does not give your friends a copy of the code for the program that they can open, learn from, or understand in detail. Still, it's better than nothing.

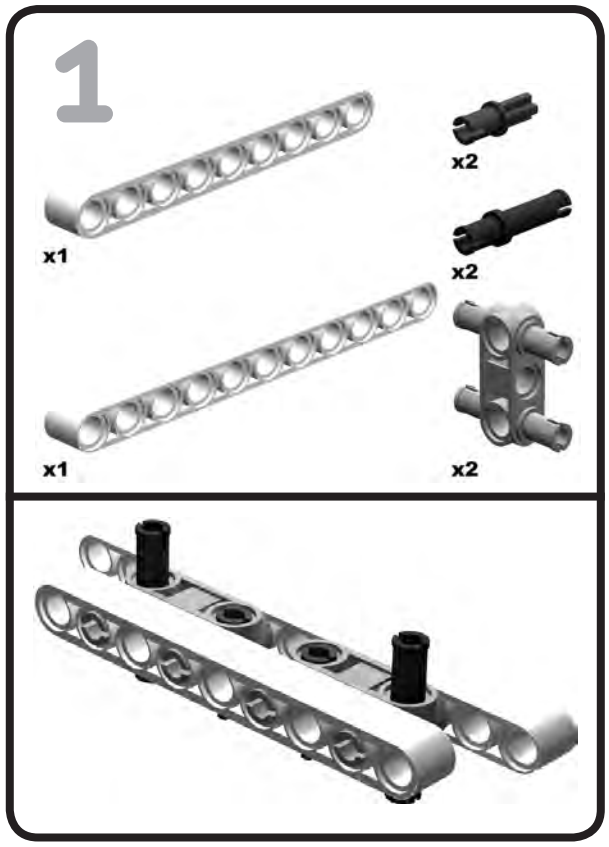
building ScanBot

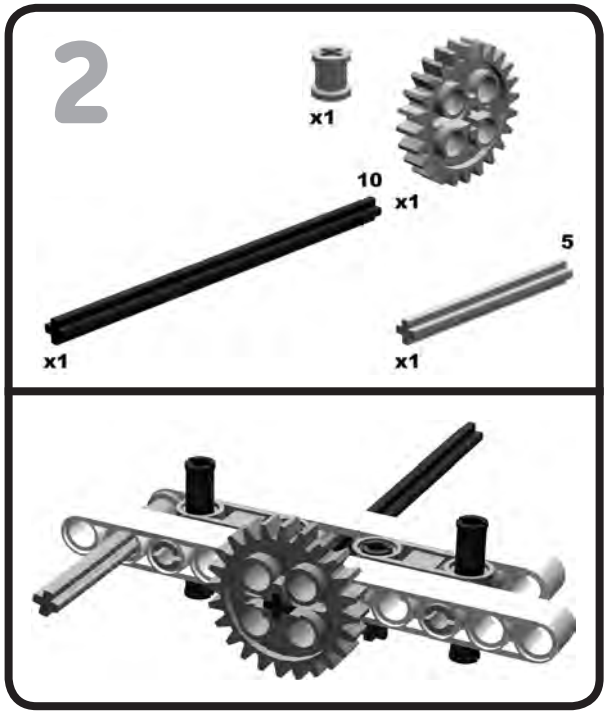
ScanBot has a simple design and is quite easy to build. It works by moving a Light Sensor across every part of an image to be scanned, taking hundreds of measurements as it goes. It compares each measurement with a variable threshold and determines whether to represent the tiny area from which the measurement was taken with a black or white dot on the NXT's LCD. A unique and fun aspect of ScanBot's design is that the entire robot moves over the image to be scanned, instead of feeding the image through the robot or moving the Light Sensor over the image within the robot.

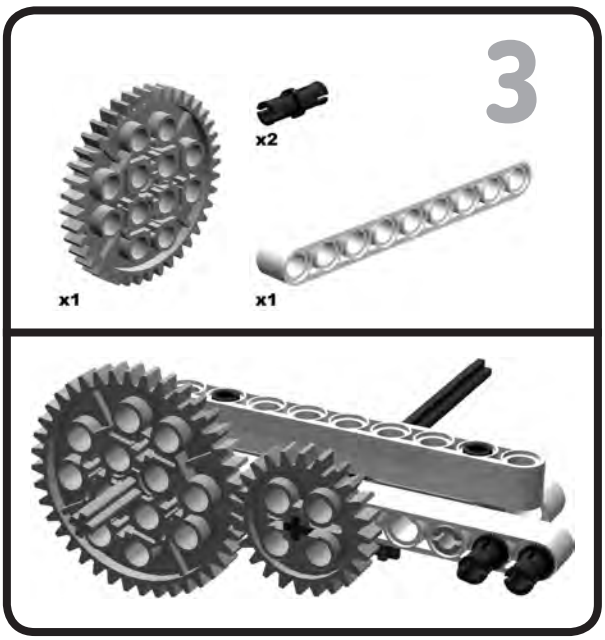
ScanBot has four main components, which are labeled in Figure 15-1. In the middle of ScanBot, two long beam constructions form the bridge, which supports the Light Sensor carriage, a motorized carriage that holds a Light Sensor and travels back and forth across the bridge to scan successive lines of the image. Each end of the bridge is supported by a wheel: one under the NXT module, and one (which isn't visible in Figure 15-1) under the motor module. The two wheels are connected by a drive shaft, which the motor module rotates to move the entire ScanBot across the image. To scan, the Light Sensor carriage travels over the bridge, scanning one line of the image. Then the motor module moves ScanBot down a tiny bit, and the Light Sensor carriage travels across the bridge again, scanning another line of the image right below the previous line. This process is repeated many times, until as much of the image that can fit on the LCD has been scanned.

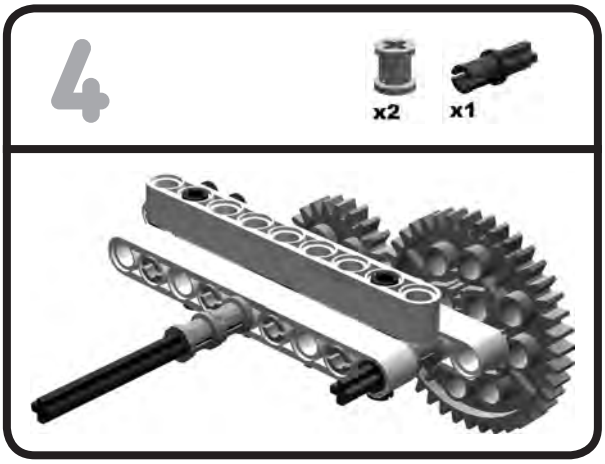
First, build the motor module, which moves ScanBot down the image being scanned. The motor module includes the wheel beneath it and part of the drive shaft that extends toward the other side of the bridge. This will later be connected to the wheel, which sits under the NXT module on the other side of the bridge. Both wheels will then turn the same amount, giving a preciseness to the downward movements that is necessary to get a good scan.











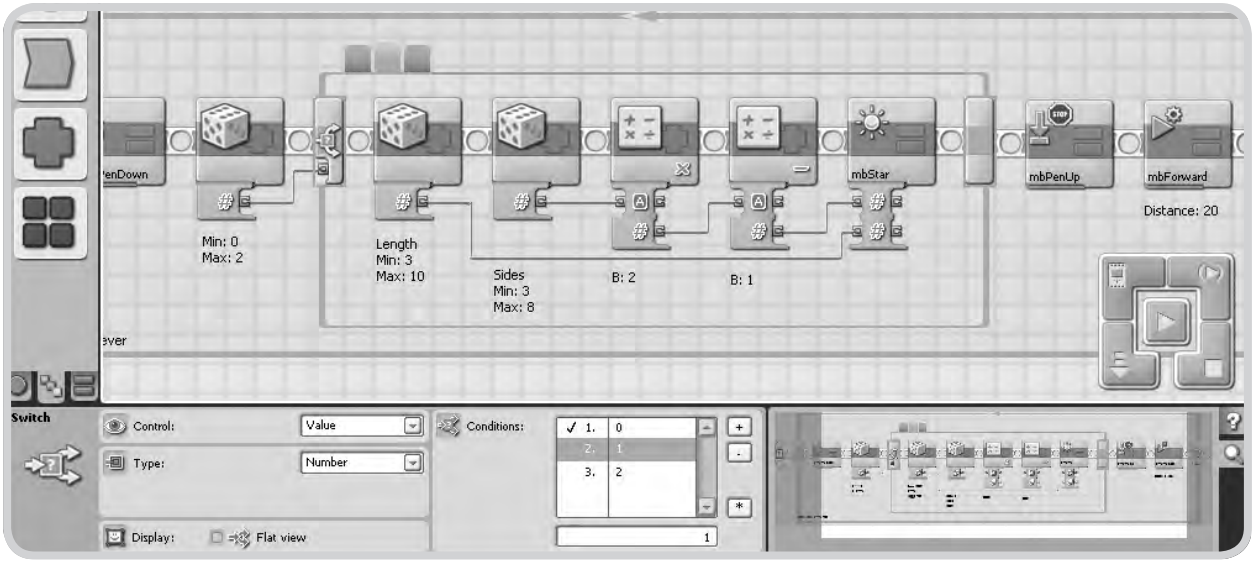


Figure 16-37: Inside the Switch block, option 2: Star

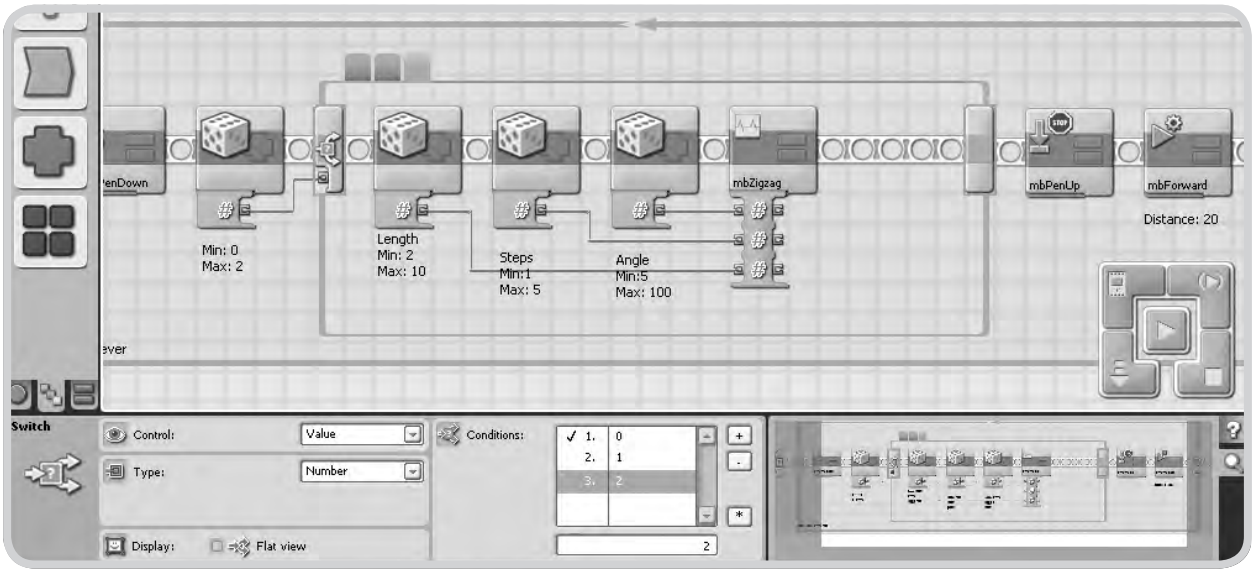


Figure 16-38: Inside the Switch block, option 3: Zigzag

Figure 16-39 shows the results when Marty runs MegaRandom.

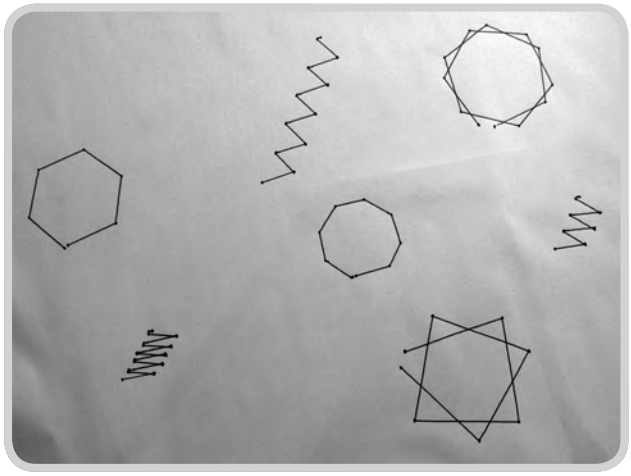


Figure 16-39: Marty, the performance artist!

SpiralStraight

Finally, here are two examples of how the counter of a Loop might be used to create a shape that develops according to the current value of the Loop counter. The first of these, SpiralStraight (Figure 16-40), is essentially a modified version of the RandomStar program, with a twist that each side is just a little longer than the previous one. The turn angle is chosen at random at the start of the program, but then remains the same for the rest of the drawing. Some examples from running this program repeatedly are shown in Figure 16-41.

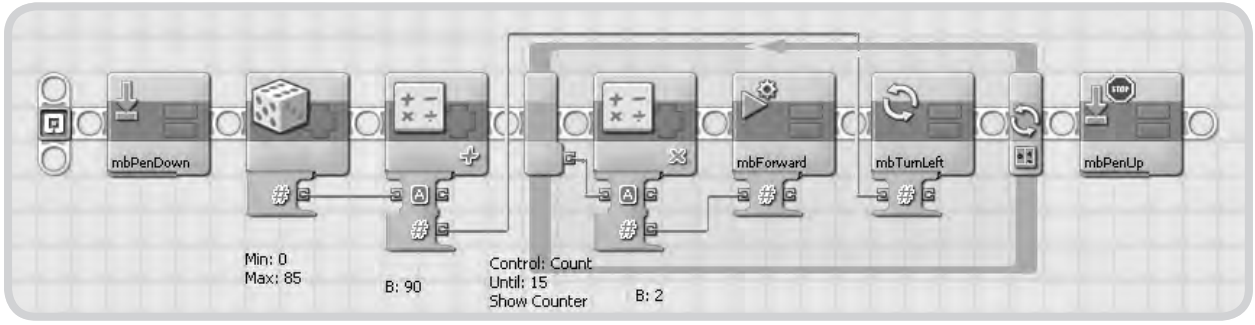


Figure 16-40: SpiralStraight

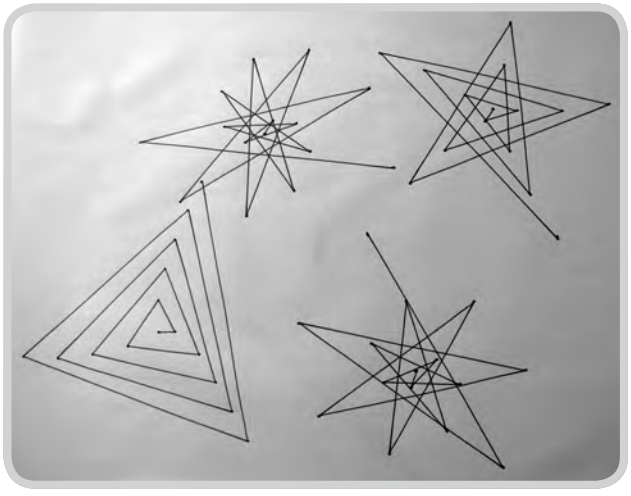


Figure 16-41: Examples of SpiralStraight