

Skini tutorial

Synchronous Reactive Programming
and Programming music

B. Petit-Hédelin July 2022

1 PART1: REACTIVE SYNCHRONOUS PROGRAMMING

A Skini program is reactive. It means that it is called by different *events* such as the command *start*, *stop*, the *pulses* emitted by the synchronization. When an event occurs, this causes a reaction. This means that our Skini program is asked to execute different instructions until a new reaction is needed to move the program forward. This way of programming is quite different from the one we practice with general programming languages. Before discussing musical programming, it is necessary to discuss some specificities of synchronous reactive programming.

We will discuss here the behavior of the Skini program according to the blocks available in the *Orchestration interface* which use the *Blockly* environment.

The programs of the tutorials can be loaded from the *./pieces* directory.

A Skini program starts with “Orch” block divided in 3 parts.

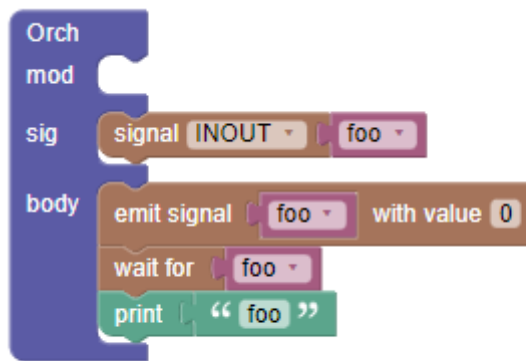
- 1) “mod” is the block in which we will put our Skini sub-modules.
- 2) “sig” is the block for declaring the signals.
- 3) “body” is the main program.



1.1 TUTORIAL 1: SIGNAL

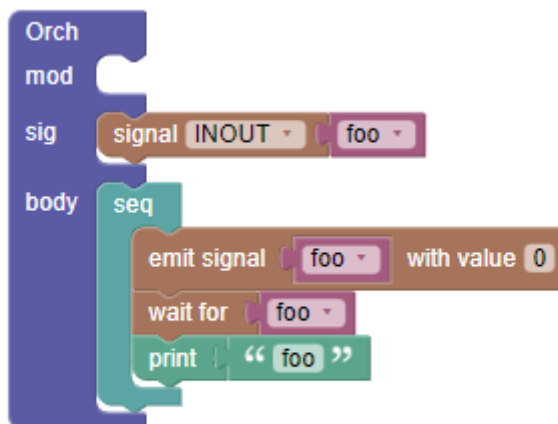
This simple program presents two basic concepts of Synchronous Programming.

- 1) The first one is the “signal”. Skini works using *signals* which are present or not at a reaction and which can have values. The *signal* can be IN, OUT or INOUT. For the tutorial is not very relevant, and we can put INOUT everywhere. The signals must be declared in the “sig” block, except for some of them which are declared by default such as the signal “tick” for example that we will see later. Signal names are created using the Blockly variables.
- 2) When the program is launched with “start”, it *reacts*. Here the reaction runs the first statement “emit” and, in the *same time* runs the “wait” statement, because the “emit” does not require a new reaction to work and, as the signal “foo” is present we print “foo”.

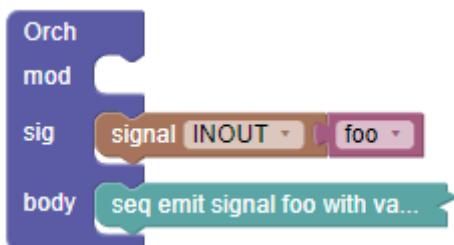


1.2 TUTORIAL 2: SEQUENCE

The result is equivalent to the Tutorial 1. The “seq” block has several usages. One is the ability to “collapse” the block in Blockly to simplify the view of the program. In a “seq” block the statements are executed one after the other in the reaction. Blockly offers interesting features for editing programs. Using the right click on a block show some of them.

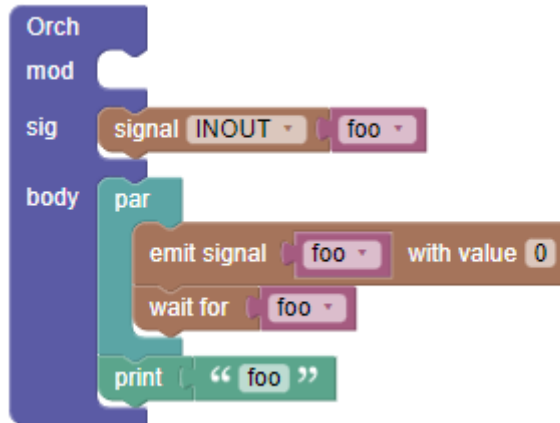


Here the block “seq” is collapsed:

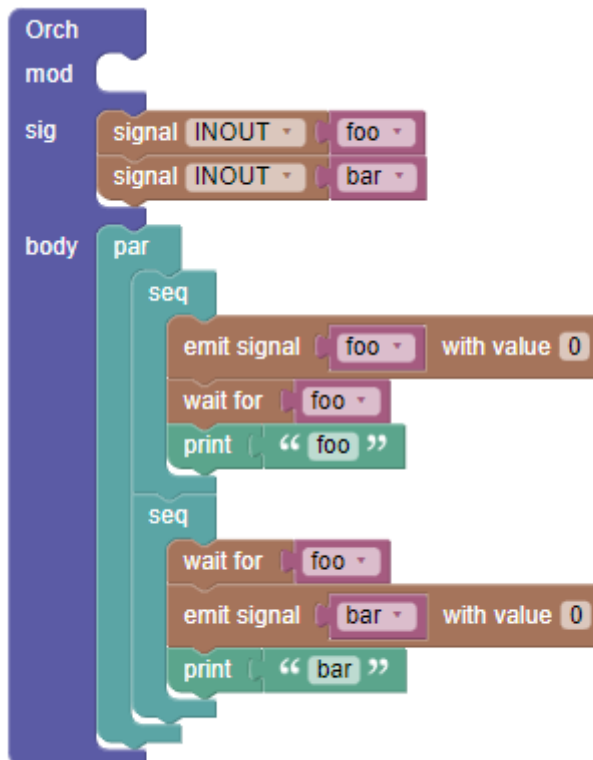


1.3 TUTORIAL 3: PARALLEL

In Skini parallel programming is natively supported. This very short program behaves exactly as the previous one. The print is executed when all the statement or block in parallel are finished.



Here is a more interesting way of using “par” and “seq” blocks. In this example we see that the “seq” blocks are necessary to put in parallel to sequences of statements.



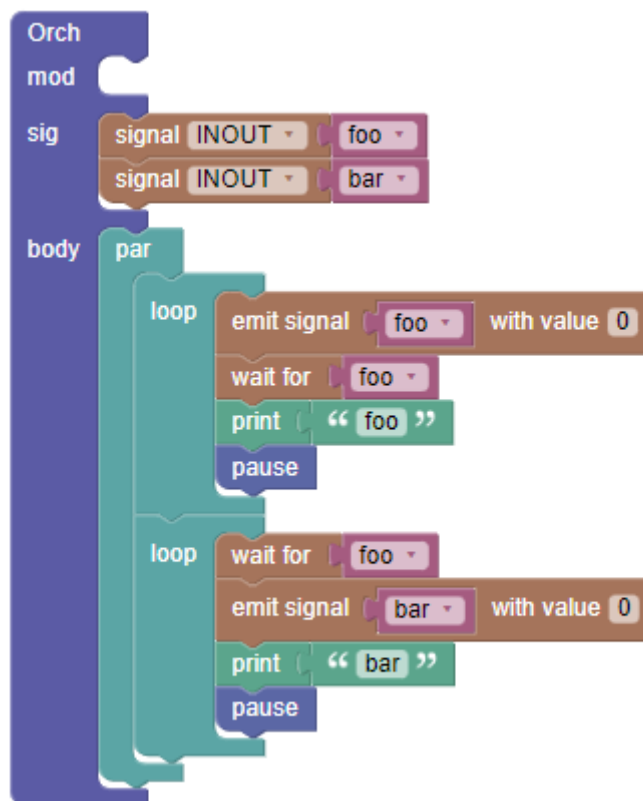
In one reaction we get “bar” and “foo” printed. We have to sequential blocks in parallel which exchange signals.

1.4 TUTORIAL 4: LOOP

The loop block plays its body and restart it. Notice here the presence of the “pause” block. Without this block you will get an error. As we have seen the “emit”, wait” and “print” are executed in the same reaction. Loop will try to restart the body of loop in this reaction which will cause another emission of the same signal “foo” in the reaction. Synchronous programming does not allow several emissions of the same signal in a reaction. “pause” is way to avoid an error due multiple emissions of the signal *foo* by stopping the sequence of statement until the next reaction.

You can try to compile the program without pause, you will get an error on the screen. This error is called a “causality error”.

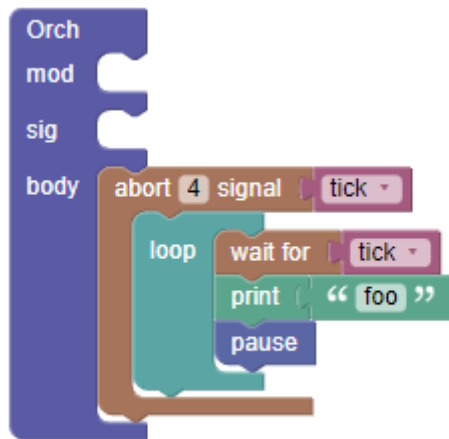
This program will never end until you “stop” it. The reaction is regularly generated by the synchronization coming from Ableton Link or MIDI Synchro.



1.5 TUTORIAL 5: ABORT

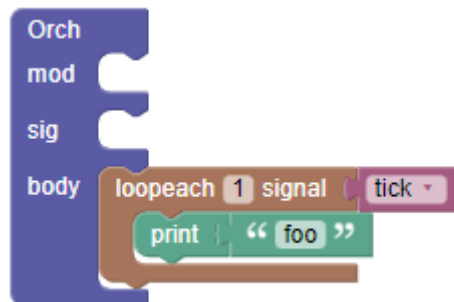
Let see a simple way of stopping a loop using the “abort” block. The “abort” block kills it body according to the presence of a signal. We introduce here the *tick* signal. This signal is created by default, you don’t need to declare it in the “sig” block. The *ticks* are emitted according to the synchronization by Ableton Link, MIDI Synchronization, or Skini local synchronization. We will use this signal very often.

Abort will act after 4 activations of the *tick* signal. Run the program you will get foo printed 3 times as the block is killed the 4th time, it is not printed this last time.



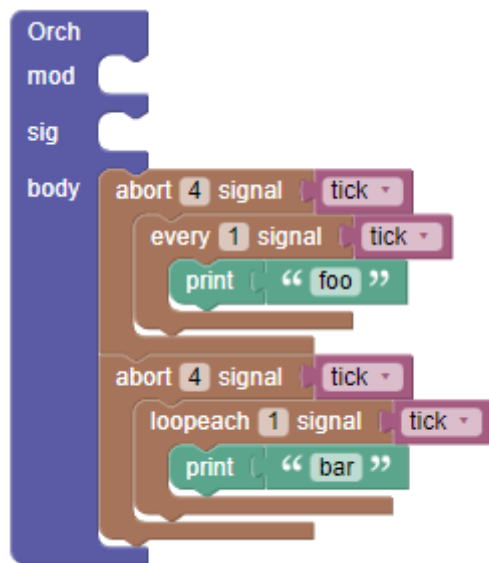
1.6 TUTORIAL 6: LOOP EACH

The behavior of this block is equivalent of the previous “loop” block with its “wait” and “pause”. We could kill the “loopeach” block in the same way as the previous block “loop”.



1.7 TUTORIAL 7: EVERY

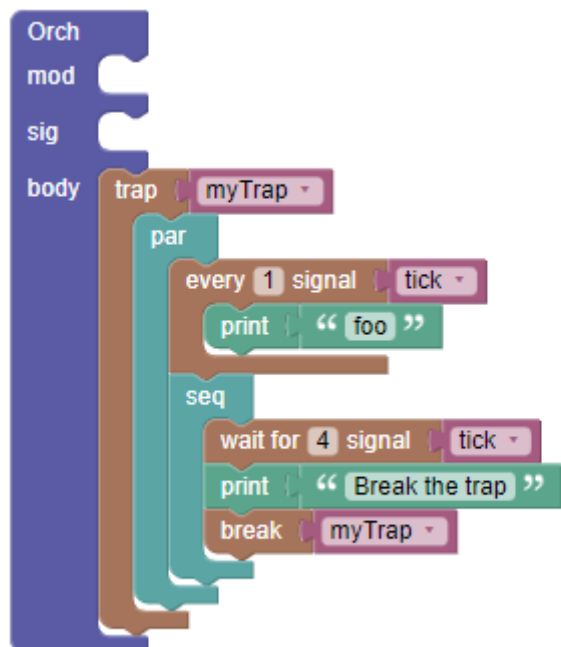
The “every” block is very close the “loopeach”. To see the difference, load the following block:



You will print foo 3 times, and bar 4 times. The only difference is that “loopeach” runs its body and processes the signal, “every” processes first the signal and then runs its body.

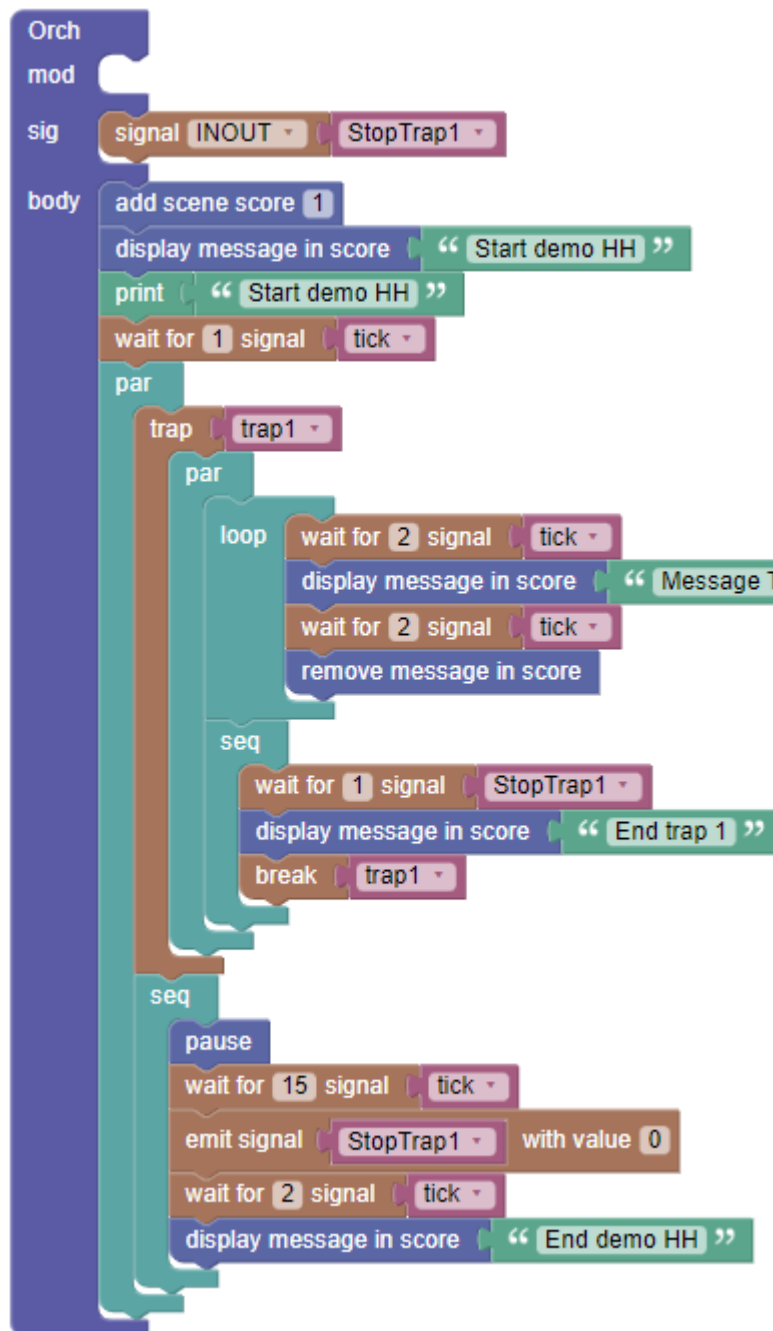
1.8 TUTORIAL 8: TRAP

Here is an example of using trap to have almost the same behavior of an “abort”. “Trap” comes with “break”. The trap is declared using a Blockly variable. Here “foo” is printed 4 times. The trap is broken after receiving a 4th tick.



1.9 TUTORIAL 10: MORE ON TRAPS

In this example we will create and use a signal: StopTrap1. (demoHHwait.xml).



We have first to declare the signal in the “sig” block. The signal is INOUT. It can be used for emitting and receiving. Most of the time we can use this option set to INOUT.

“add scence score” will allow us to see the result of our program on the “score” window. It is set to 1 according to what we put in the group description where the “scene” of the groups is set to 1.

“print” is a way to get information on the console.

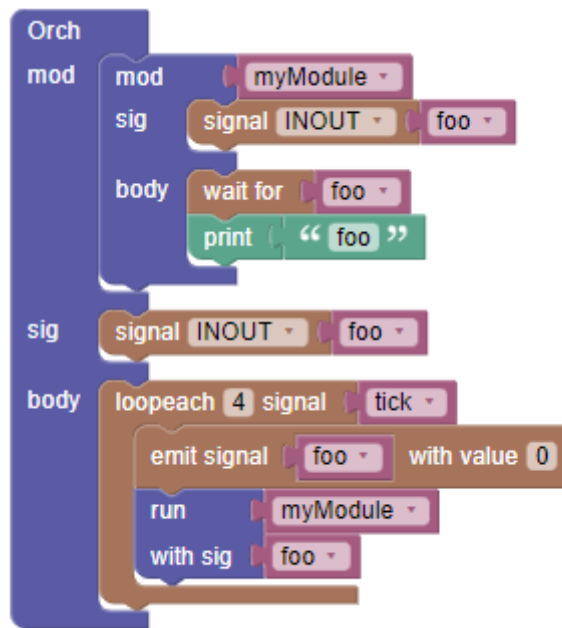
We have several “wait for” blocks in this example. They allow to stop the program until a specific signal arrives. Signals must be first created as variables to be used.

You already know about “seq” and “par”.

The goal here is to stop the “loop” when a “StopTap1” signal is emitted. For that we have the trap in parallel with a simple way of emitting the “StopTrack1” signal using some “ticks”. The “break” block kill the trap.

1.10 TUTORIAL 9: MODULES

You can structure your program using modules. They have the same structure as the “Orch” block. The signals used in the modules must be declared. There are no default signals in the modules. “tick” would have to be declared for example.



1.11 CONCLUSION

Reaching this point, you got the basement of the Skini programming. There are other subtleties that will be covered when dealing with more musical examples, but you already have the programming basics for creating music.

2 PART 2: EXAMPLES OF PROGRAMMING FOR MUSIC

In this part we will review the basic concepts of part 1 applied here to make music.

To run the program and the music we must prepare the musical environment. Here are the patterns descriptors and the groups we will use in our tutorials.

	Note	Note stop	Flag	Text	Sound file	Instrument	Slot	Type	Free	Group	Duration
1	10	510	0	Beat1	Beat1	0	0	4	0	0	8
2	11	510	0	Beat2	Beat2	0	0	4	0	0	8
3	12	510	0	Beat3	Beat3	0	0	4	0	0	8
4	13	510	0	Beat4	Beat4	0	0	4	0	0	8
5	14	510	0	Beat5	Beat5	0	0	4	0	0	8
6	20	511	0	Ambiance1	Ambiance1	1	0	4	0	1	4
7	21	511	0	Ambiance2	Ambiance2	1	0	4	0	1	8
8	22	511	0	Ambiance3	Ambiance3	1	0	4	0	1	4
9	30	511	0	Synthe1	Synthe1	2	0	4	0	2	2
10	31	511	0	Synthe2	Synthe2	2	0	4	0	2	2
11	32	511	0	Synthe3	Synthe3	2	0	4	0	2	2
12	33	511	0	Synthe4	Synthe4	2	0	4	0	2	2
13	34	511	0	Synthe5	Synthe5	2	0	4	0	2	2
14	35	511	0	Synthe6	Synthe6	2	0	4	0	2	2
15	36	511	0	Synthe7	Synthe7	2	0	4	0	2	2
16	40	511	0	Conga1	Conga1	3	0	4	0	3	4
17	41	511	0	Conga2	Conga2	3	0	4	0	3	4
18	42	511	0	Conga3	Conga3	3	0	4	0	3	4
19	43	511	0	Loop1	Loop1	4	0	4	0	4	8
20	44	511	0	Loop2	Loop2	4	0	4	0	4	8
21	45	511	0	Loop3	Loop3	4	0	4	0	4	16

☐ Simulator in a sepearte Group

Tempo Max

Tempo Min

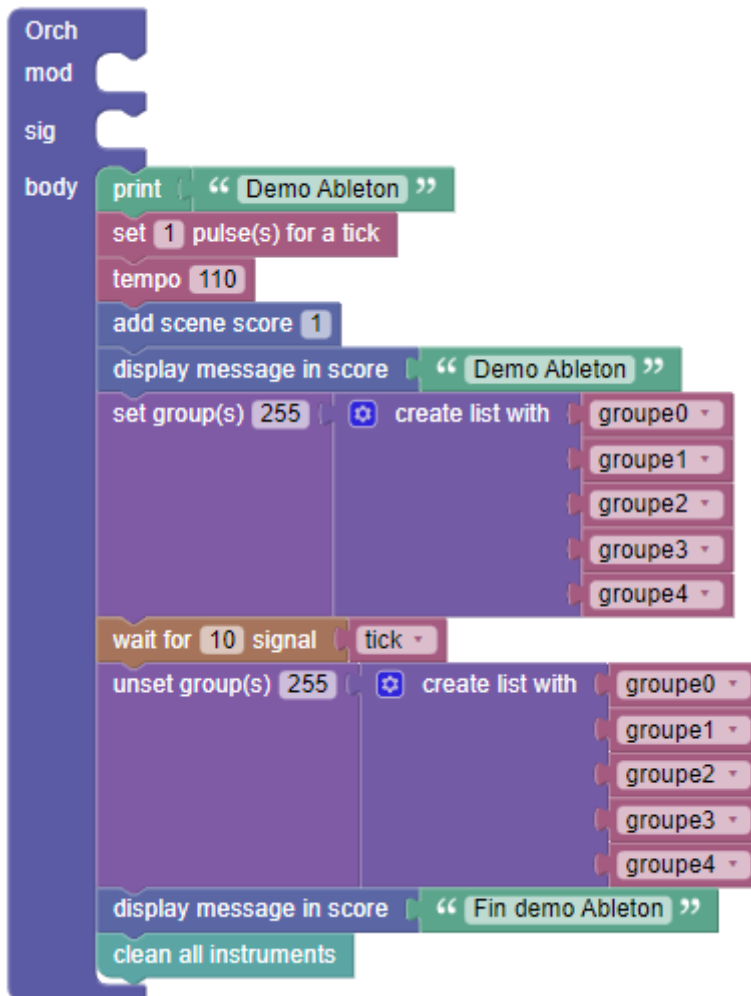
Limit Waiting Time (in pulse)

	Groupe	Index	Type	X	Y	Nb of E1. or Tank nb	Color	Previous	Scene
1	groupe0	0	group	170	100	20	#CF1919		1
2	groupe1	1	group	20	240	20	#008CBA		1
3	groupe2	2	group	170	580	20	#4CAF50		1
4	groupe3	3	group	350	100	20	#5F6262		1
5	groupe4	4	group	20	380	20	#797bbf		1

2.1 A FIRST METHOD FOR SETTING GROUPS

(tutoGroups1.xml, demoAbleton.csv)

There are several ways of presenting (or activating) groups of patterns to the simulator or the audience. This example shows how to present groups during a period. Skini receive “pulses” according to the synchronization mechanism chosen. We could decide to set the “tick” according to a multiple of these pulses. The impact of the choice will be explained later. We chose to have a tick per pulse. “set group(s)” can be used for a single group or for a list of groups.



“add scene score” will allow us to see the result of our program on the “score” window. It is set to 1 according to what we put in the group description where the “scene” of the groups is set to 1.

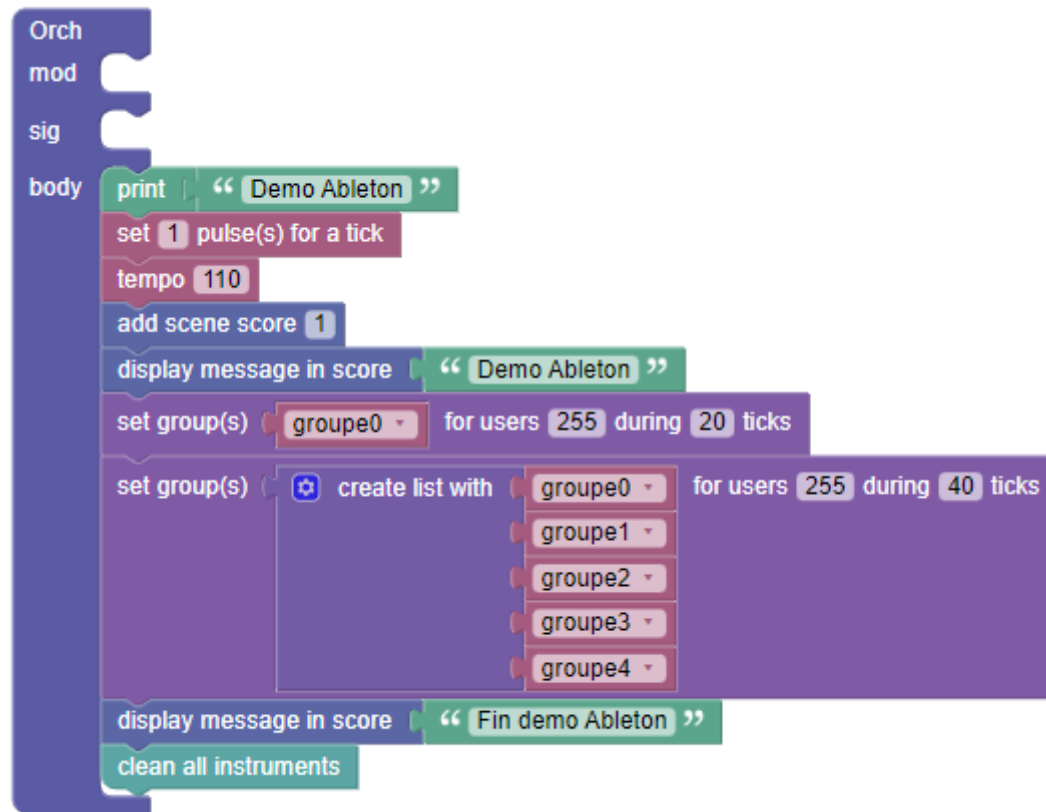
The groups are set for a *range* of users. Skini is designed for interactive or generative music using web clients. Each client can be assigned to a number. When activating a group, we can choose to tell for which clients we want to activate the group. 255 means everybody.

“clean all instruments” is a way to empty the FIFO of the instruments which have been filled by the simulator.

2.2 ANOTHER METHOD FOR SETTING GROUPS

(tutoGroups2.xml, demoAbleton.csv)

There are blocks dedicated to activating groups during a period. We first set the group 0 with a time limit of 20 ticks. Then we set several groups together for 40 ticks.

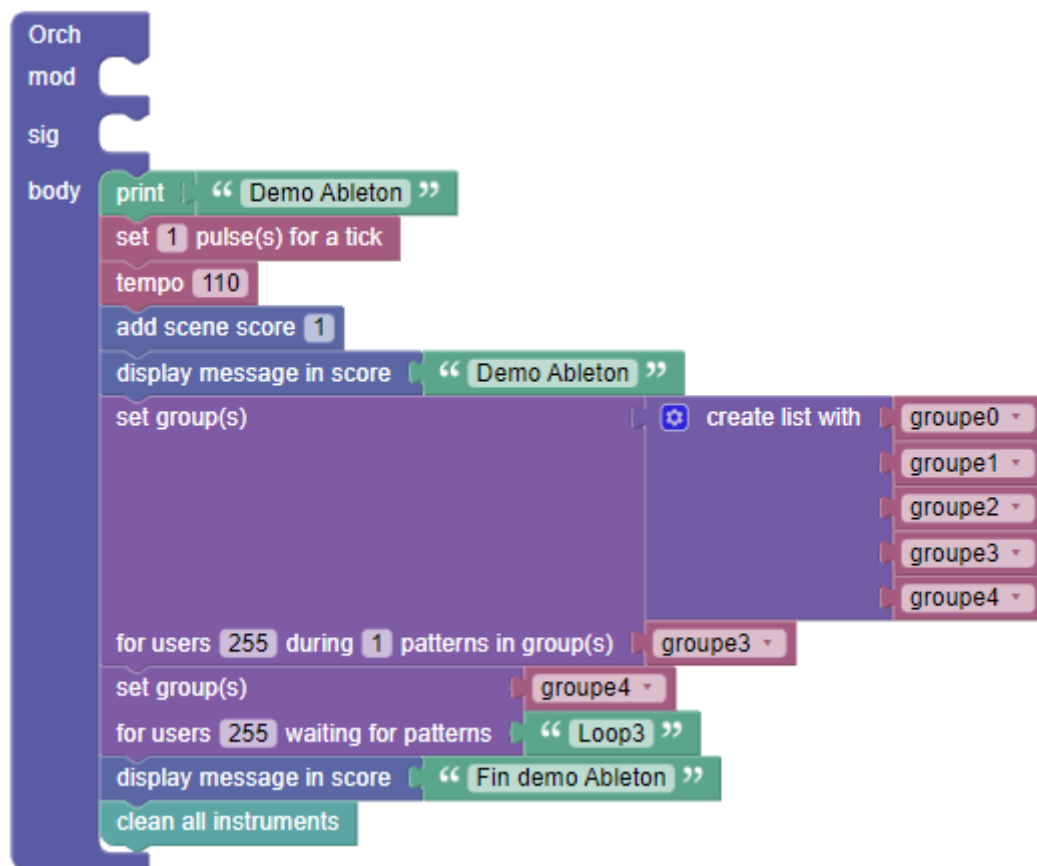


2.3 SETTING GROUPS WAITING FOR PATTERNS

(tutoGroups3.xml, demoAbleton.csv)

Skini is designed to deal with random processes. Radom could mean that an audience is producing events or a random engine like the simulator.

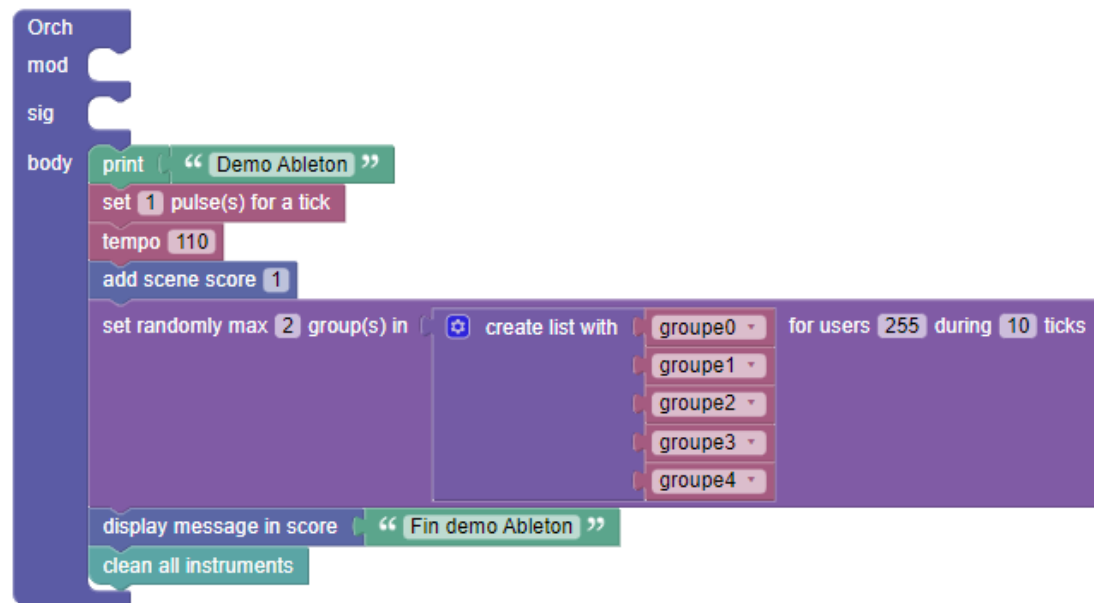
In the following program we have two ways to manage the activation of groups according to random events. In the first “set group” we are waiting for a pattern appearing in a specific group before deactivating the groups. In the second “set group” we wait for a specific pattern. In both cases we can use a single group or a list of groups. Notice that the patterns in the second “set group” must be a string and not a variable.



2.4 SETTING GROUPS RANDOMLY

(tutoGroups4.xml, demoAbleton.csv)

You can use random process in the compilation of your program. Here the program produced will activate a maximum of two groups in the list during 10 ticks. You can use the *controller* or *score* to visualize different compilations.



3 CONCLUSION

This tutorial is an introduction to the basic Skini programming. You are now ready to read the chapter 8 of the Skini documentation which details all the blocks.