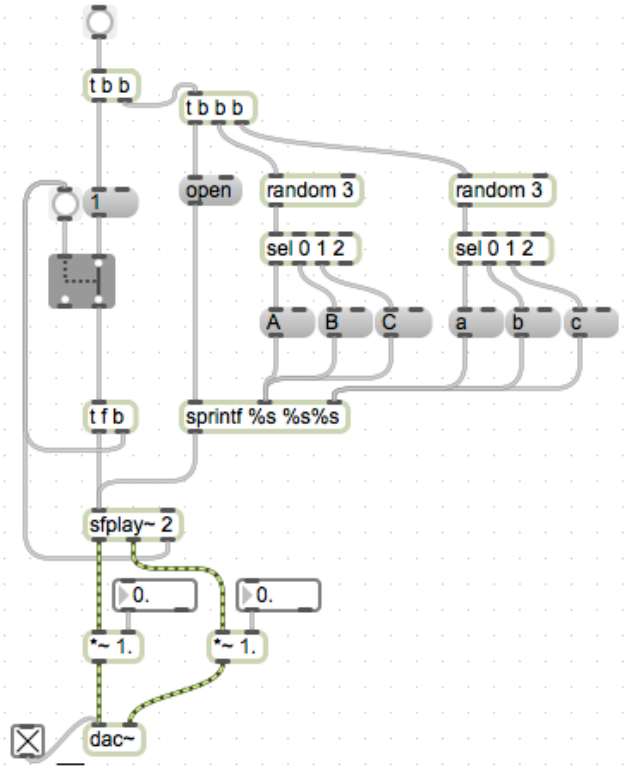
Expanding `sfplay~`

Electronic Music II

Spring 2013

1. Subpatches

a. We begin with one instance of our basic `sfplay~` setup, shown below.

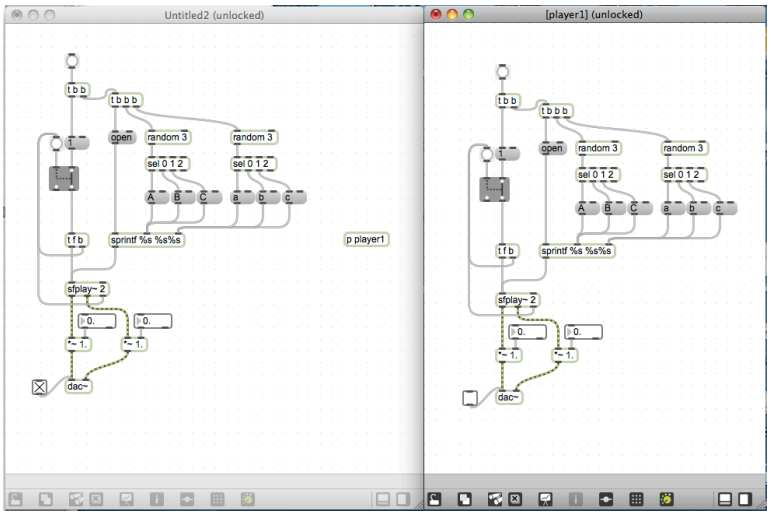


b. Copying and pasting this setup in one patcher quickly becomes visually confusing. This can slow down troubleshooting, and also makes it cumbersome (and more error-prone) to make any changes to one part of the setup later on.

c. We can mitigate this by using subpatches. Create a blank object, type the letter 'p', followed by a space, then the name for the subpatch.

d. Creating this object will open a new window with the name of the subpatch. This is effectively a 'compartment' of the patch; all subpatches are still considered part of the main patch.

e. Copy and paste our `sfplay~` setup into the subpatch window.



2. Sends, receives, inlets

a. Since we now have **2** `dac~`'s in use, we encounter a new aspect of `dac~`'s behavior.

   i. With **one** `dac~`, checking the toggle object will turn on audio processing

   ii. With **two** `dac~`'s and **one** toggle object, the toggle object will turn on/off audio processing for **both** `dac~`'s

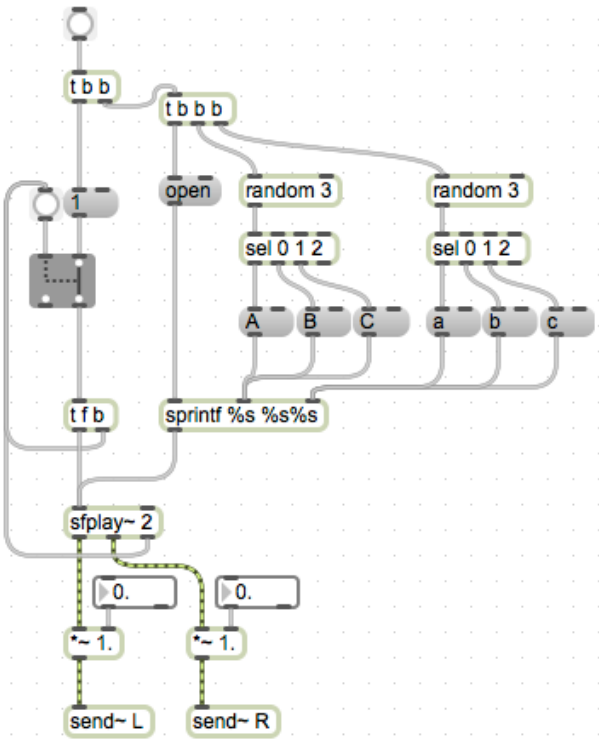
   iii. With **two** `dac~`'s and **two** toggle objects, **both** toggle objects must be on to turn on audio processing for **both** `dac~`'s. Having one toggle off and the other on will **not** turn on audio processing.

b. To avoid complications involving this behavior, we will only be using a `dac~` in the main patch, not in any subpatches.

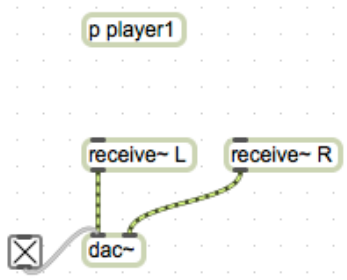
c. We will route audio from the subpatch to the main patch by using `send~` and `receive~` objects.

d. These objects create 'cordless' connections in a patch. Audio sent into a `send~` comes out from its matching `receive~`.

e. Create a `send~` and `receive~` for each channel of audio coming from `sfplay~`. Here I am labeling them L and R. Attach the outlets of the respective `*~` to its `send~`, as shown below.



f. Create a `receive~` for each `send~`, and connect their outlets to the inlets of `dac~`.



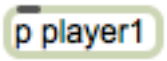
g. Now, audio created in our subpatch by `sfplay~` is running out to our single `dac~`, in the main patch.

h. We can route as many subpatches as we want this way. For demonstration, I will create nine new subpatches with identical contents.

i. As the number of subpatches in our main patch increases, it becomes less and less practical to control them all from within their own windows. We can control them from the main patch by creating inlets.

j. Create a blank object in a subpatch, and type 'inlet'. Instead of an ordinary text-box style object, it will create an object like the following:

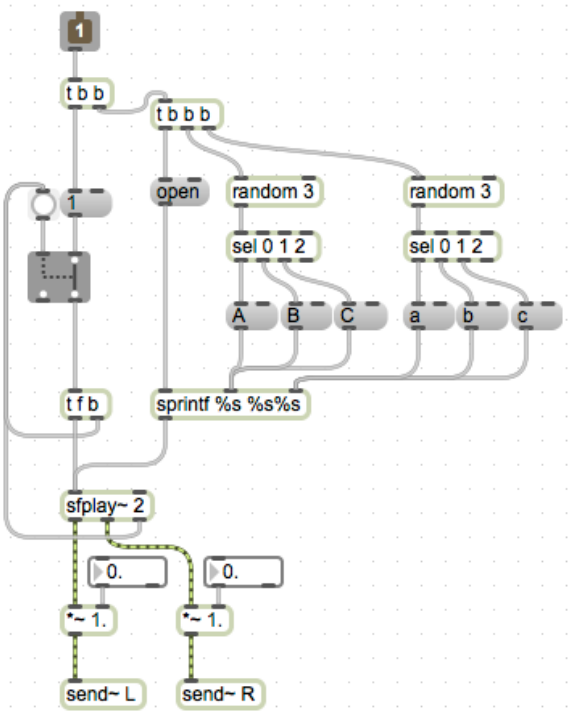
k. Notice that, in our main patch, the subpatch object now has an inlet:



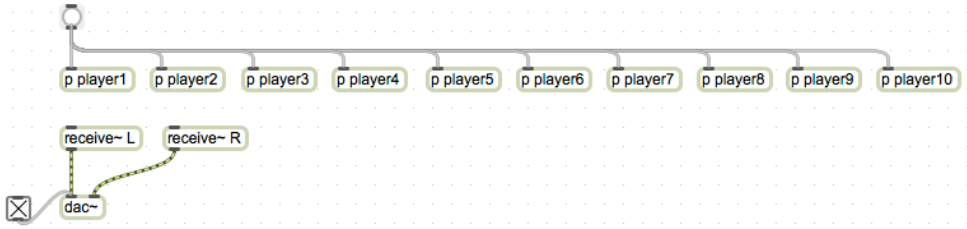
This will pass any input into the subpatch.

l. Create a bang button in the main patch, and connect it to the inlet of our subpatch.

m. Then, in the subpatch, connect the outlet of the `inlet` object to the `trigger`, which in our demo will select a sound file, open it, and play it.



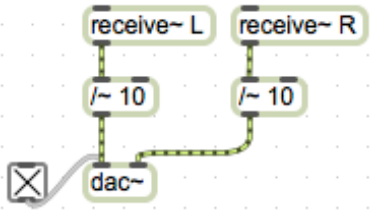
n. We can now trigger all of our subpatches from the main patch.



3. Mixing

a. Now that we have multiple objects sending sound to the `dac~` at the same time, we face the risk of the total amplitude being greater than 1, creating distortion.

b. Since we know how many objects are in play (for this demo, 10), we can use the `/~` object. Place one between the `dac~` and each `receive~`.

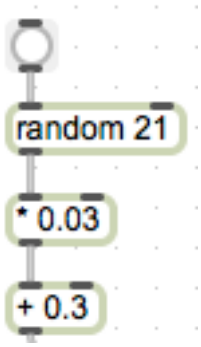


c. This will reduce all signals coming out of the `receive~`'s to a tenth of their value.

4. Random panning

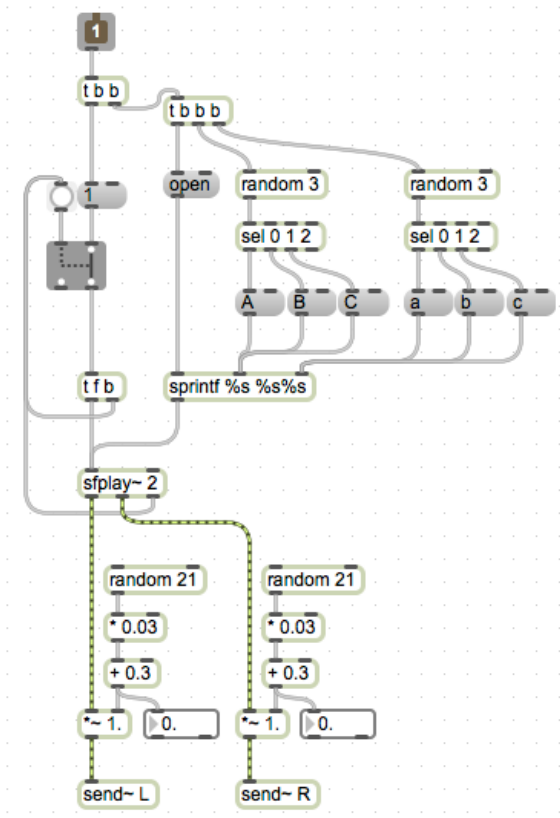
a. We can create a random panning of each sound we trigger, by controlling the amplitude of each output channel of `sfplay~`.

b. For this demonstration, I will use `random` and some math objects to create random numbers between 0.3 and 0.9, in 20 increments.

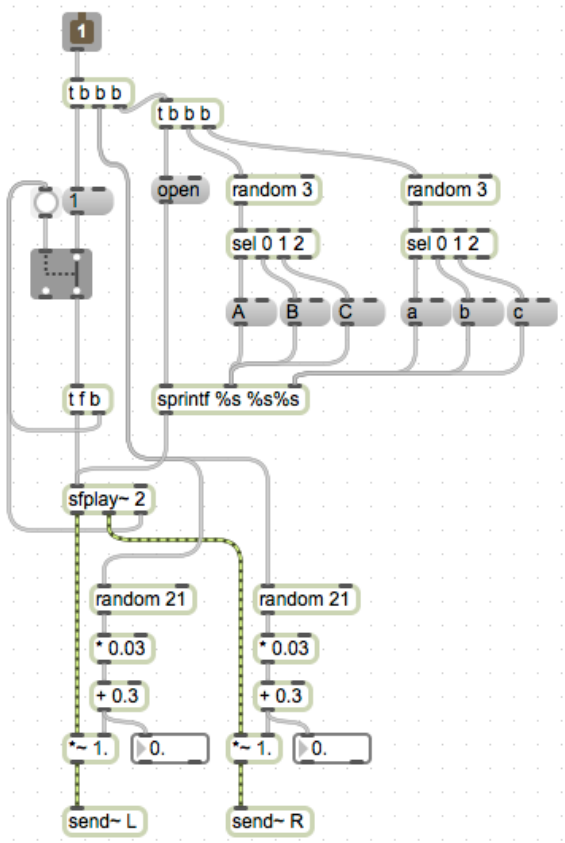


c. A brief explanation of the math involved:
   i. `random 21` creates values between 0 and 20. These represent the increments I want to create.
   ii. Because I want to scale these values between 0.3 and 0.9, I take the difference between them and divide by 20. [(0.9 – 0.3) / 20] = 0.03. Multiplying the output of `random` by this value gives us an output between 0 and 0.6.
   iii. Now to shift this range into what I want, I add 0.3. This makes the output between 0.3 and 0.9, in 20 increments.

d. Now, in each of our subpatches, add two copies of this setup. Connect the outlet of `+ 0.3` to the right inlet of each `*~`, as shown below.



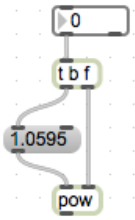
e. Add a bang to the first `trigger` object, such that the `random` objects are triggered before `sfplay~`.



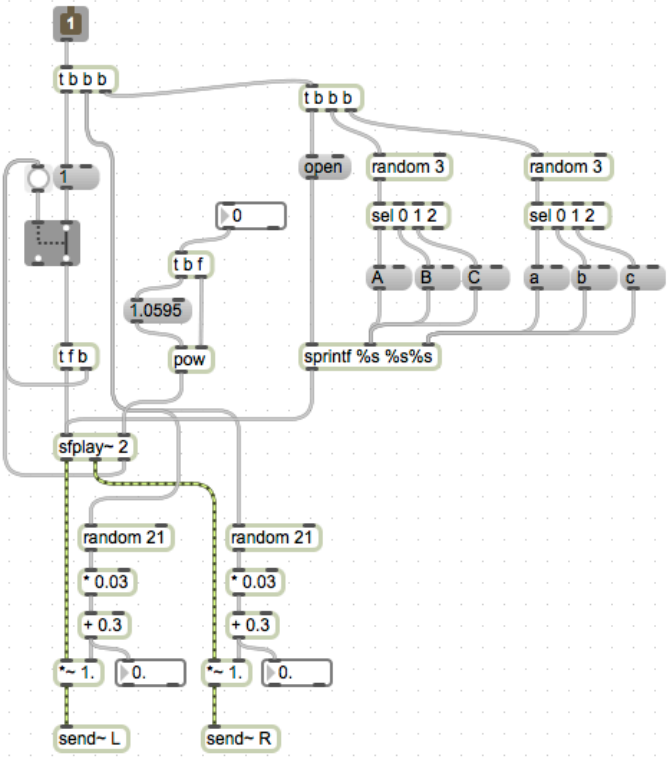
f. Now, each time we playback a sound file, the stereo field is randomly shifted.

5. Random pitch shift

a. Recall the setup introduced previously, to change the playback speed such that the sound file is transposed by a number of semitones:

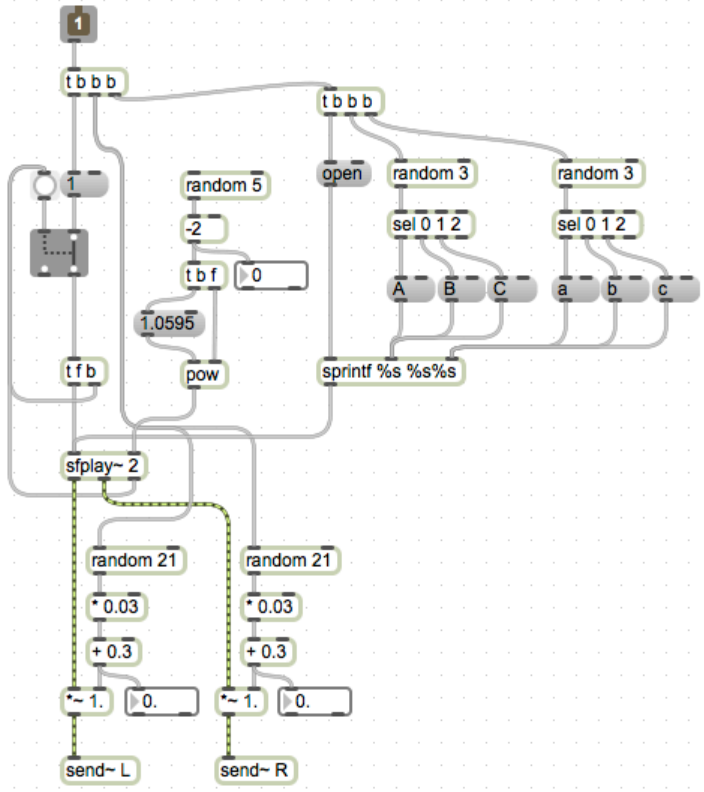


b. To incorporate this into our existing setup, connect the outlet of `pow` to the right inlet of `sfplay~`.

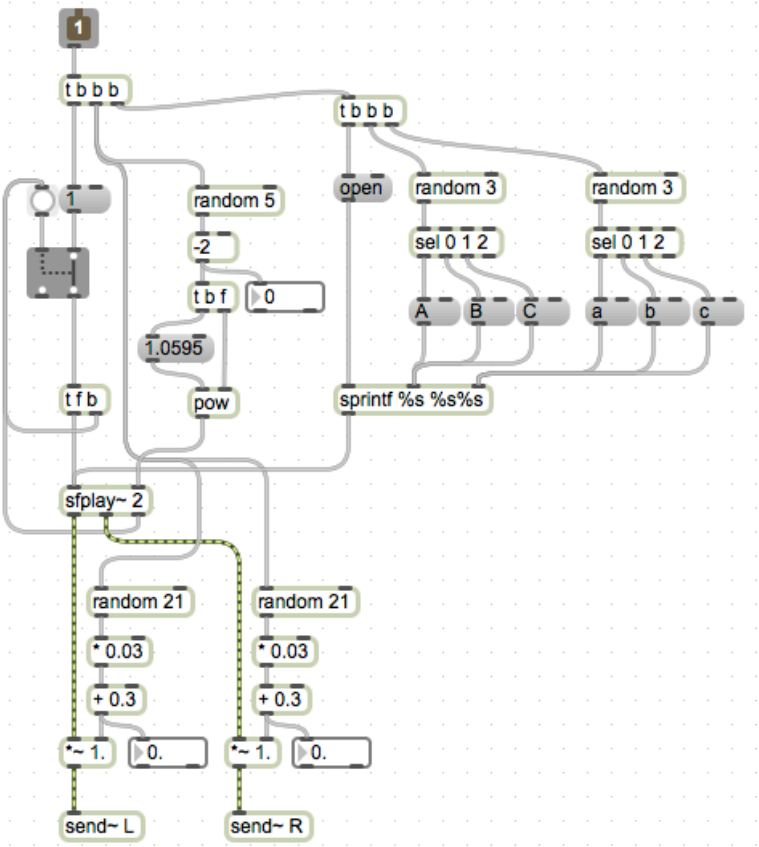


c. Now create a `random` object. This will create values for the amount of pitch shift. For this demonstration, I will create objects such that the sound file will be shifted between -2 and 2 semitones.

d. Connect the outlet of this setup to the inlet of the pitch-shift `trigger` as shown.



e. We want to set this value before triggering `sfplay~`, just like with the value for panning. We can connect the same bang that sets those values to also set this value.



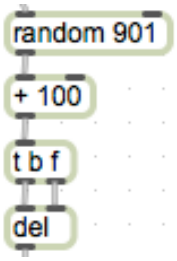
f. Now, each sound file will not only be panned randomly, but will randomly be pitch-shifted, either up or down, by up to 2 semitones.

6. Random delays

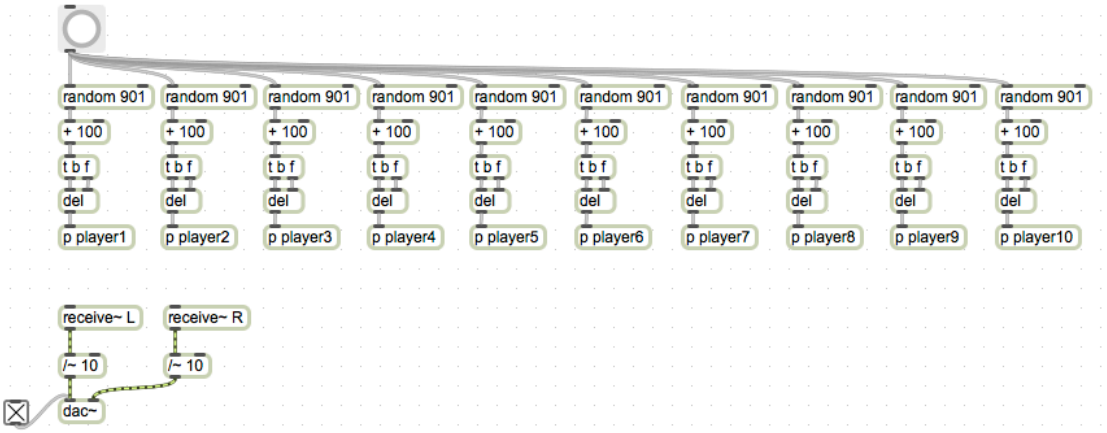
a. To add unpredictability to each total event created by this setup, we can use `delay` objects in combination with values generated by `random`.

b. As mentioned earlier, `delay` takes a bang as input, and outputs a bang after a number of milliseconds.

c. For this demonstration I will create values for `delay` that are between 100 and 1000 (ie, 1/10th of a second and 1 second).



d. Because this modification concerns more the overall output of the patch, and less each individual sound file's literal playback, I will connect the output of this setup to the inlet of each subpatch.



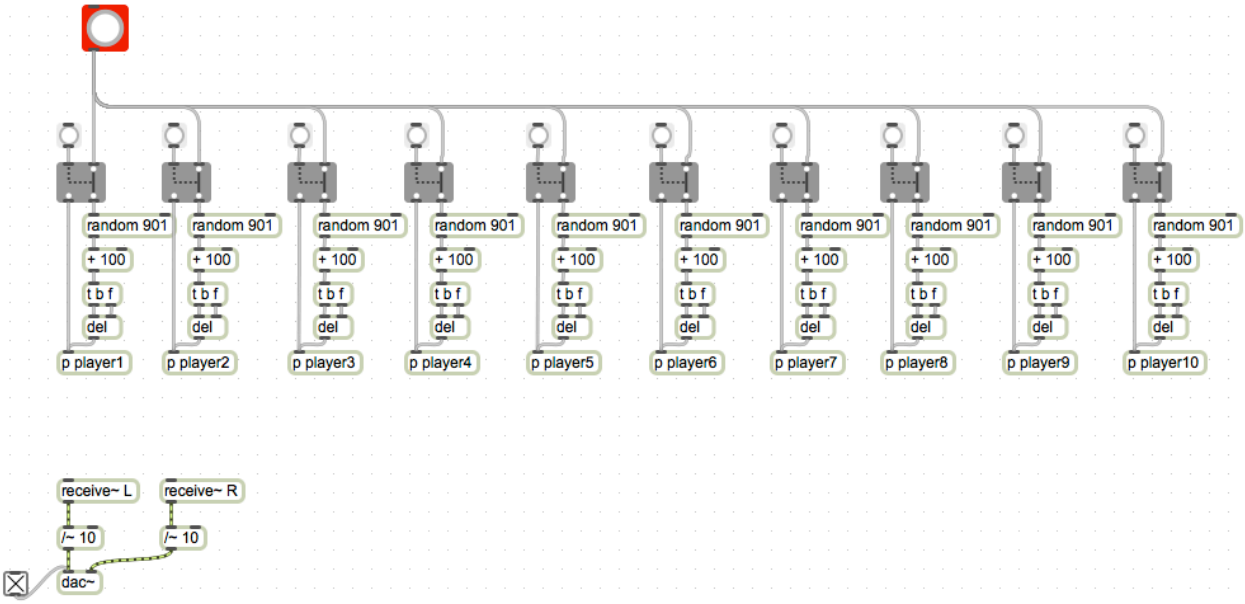
e. Now, each subpatch will be triggered after a random delay between 0.1 and 1 second.

7. Controlling randomness

a. Ironically, with so many randomly determined parameters, a useful way to vary the output of our patch is to *not* randomize some parameters.

b. We will do this using `ggate` objects, both in our main patch and subpatches.

c. To switch our random delays, add a `ggate` before the `random` object. One outlet of `ggate` will go to `random`, creating a random delay, and the other will go to the subpatch, triggering a sound with no delay.

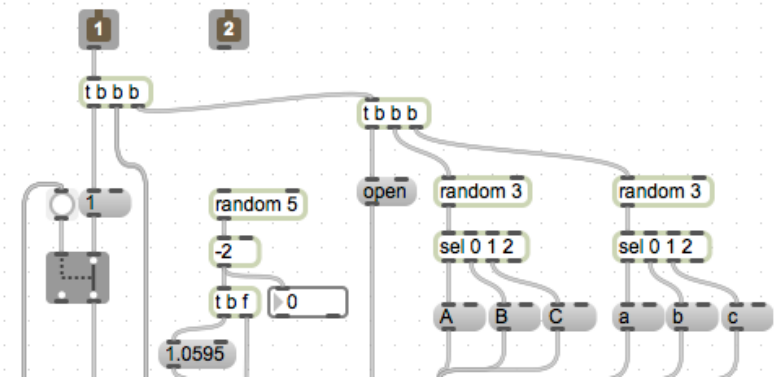


Notice that I have changed the color of the bang button that is triggering sounds, to visually distinguish it from the buttons to switch on or off delay for a subpatch.

d. To add switches inside of a subpatch, but control them from our main patch, we need to add inlets.

e. This process will involve restructuring our subpatches slightly, so that we can control which values are determined randomly entirely from the main patch. I will begin with controlling the pitch shift parameter.

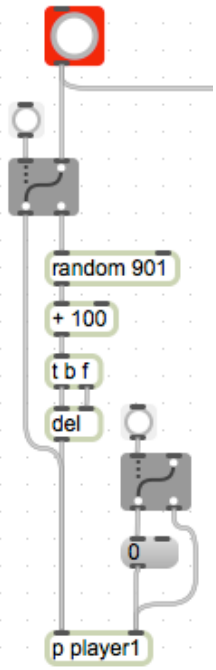
f. First, I will create an `inlet` within a subpatch.



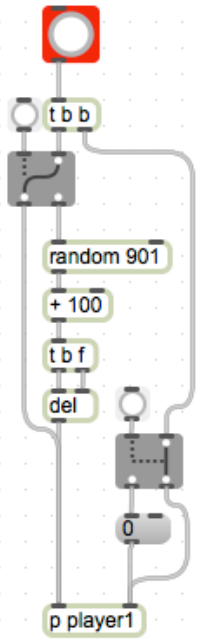
(patch image truncated for space)

g. In the main patch, create a `ggate` and a message box containing the number zero. Connect one outlet of `ggate` to the inlet of the message box.

h. Connect the outlet of the message box **and** the other outlet of `ggate` to the new inlet on the subpatch. Shown below.



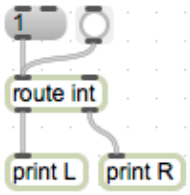
i. Create a `trigger` object, and connect the **main** bang button to its inlet. We will use this trigger to set all values for the subpatch, before sending the bang to play a sound (delayed or not).



j. At this point, the second inlet of our subpatch is receiving either a 0 or a bang. We want 0 to signify **no** pitch shift, and the bang to **cause** pitch shift. To tease these apart, we will use the `route` object.

k. `route` sends input to different outlets, depending on whether it matches an argument given when the object is created. In this regard it resembles `select`, but because it outputs the actual input, compared to *select* which outputs bangs, we will use this (to reduce the number of message boxes we need to create).

l. A brief demo of `route`:

i. 

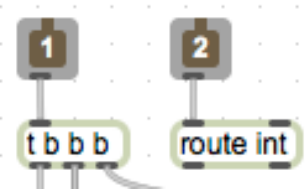
(setup)

ii.

Object	Message
L	1
R	bang

(output)

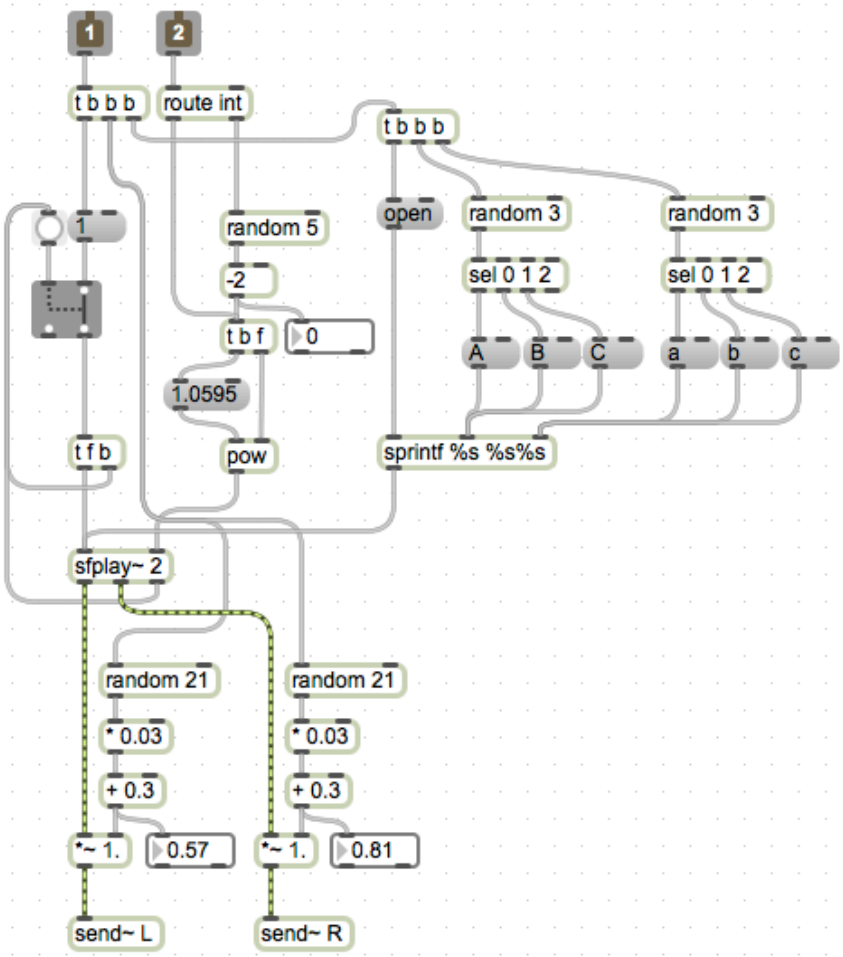
m. In the subpatch, create a `route int` object. Connect the 2nd inlet to its inlet.



n. As demonstrated above, the left outlet of `route` will pass along our 0 message. The right inlet will pass along a bang.

o. Connect the left outlet to the `t b f` trigger in the pitch-shift chain. Connect the right outlet to the `random 5` object in the same chain.

p. **Disconnect** `random 5` from the bang coming from inlet 1.



q. Now, from the main patch, we can control whether the sound is pitch shifted randomly, or pitch shifted by a fixed amount (in this demo zero, or no pitch shift). Follow a similar procedure to add control over the other playback parameters (L amplitude, R amplitude, file selection [A and a]).

r. At this point, it would be wise to mark in the subpatch what each inlet is effecting. We can do this with comment boxes. These technically are `comment` objects, but appear in the patch simply as boxes of text.