
PRACTICAL 4 – KEYSTROKE SONIFICATION

Introduction

Your task this week is to work on a challenge: the development of a simple typing sonification system. The general idea of the system is that it should play sounds when keys are pressed on the keyboard. More specifically, the sounds played should be scales or arpeggios that start at the first letter of a word and end with the last letter. For example, if I type the words “hi bingo”, I should hear a sequence of two ascending notes, followed by a sequence of five ascending notes. Ascending refers to the pitch of the notes, and the notes should be played as the keys as pressed.

The choice of sequence of notes doesn't matter much for this exercise, you can choose whichever one you want. For example, you can choose a “major scale” such as C D E F G A B. Or you can choose one based on the frequency of the notes played. A simple one to produce is a harmonic sequence whereby the frequency of a note is a multiple of the frequency of the first note in the sequence. For example, if you start the sequence at 100Hz, the next frequency will be 200Hz, then 300Hz, etc.

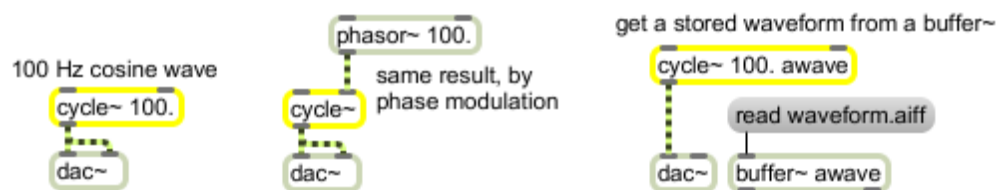
Tasks

The general task is described in the introduction, but you may find it challenging to complete it without clues. So here's how you can break down the problem into smaller problems.

Synthesizing a sound

In order to complete the task, you need to know how to play a sound, or in fact, a number of different sounds. There are different ways in which we could do this. For example, we could play samples which have been pre-recorded, such as the notes of a scale played by a piano. We can also use a synthesiser that can generate notes in real-time, this will be our chosen solution.

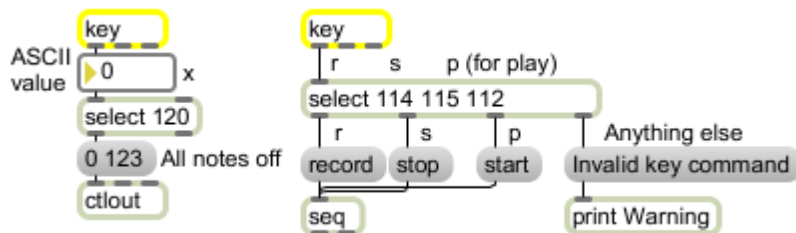
The simplest way to generate a sound is to use an oscillator object: **cycle~** (note the ~, which signifies that this is an MSP object).



Step 1: Synthesize a (pure tone) sound using `cycle~` and control its frequency using a number input box.

Play (synthesize) a sound when a key is pressed

Capturing keyboard events is easy using the **key** object:

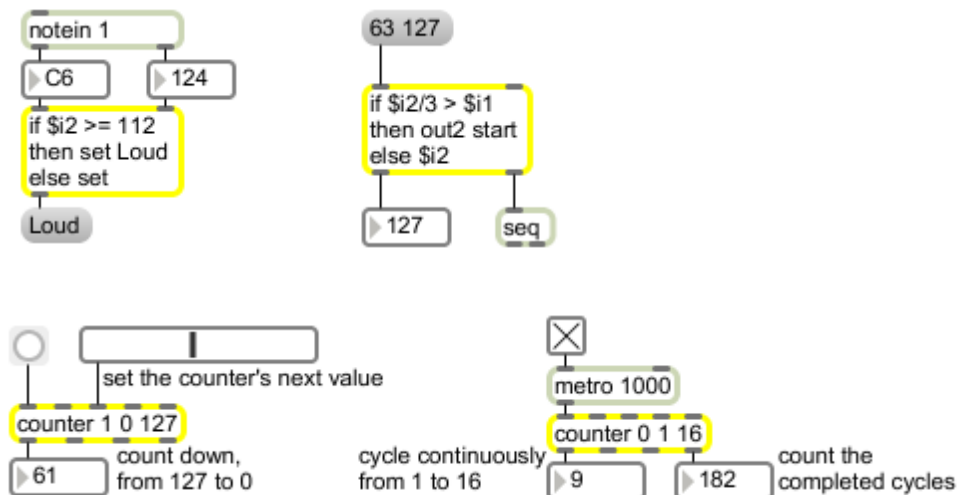


The next step involves synthesizing a sound when a key is played. The only sound synthesis parameter we need to worry about is the frequency of the sound we are generating. Since the **key** object is returning a numerical value when a key is pressed and the **cycle~** object takes a number as an input, linking the two together is not difficult.

Step 2: using the key object and the result of step 1, synthesis a sound that has the frequency of the key ASCII value. Note that these ASCII codes are in the range of around 100-150, so you should expect to generate low pitch sounds with these frequency values.

Identify words and count their letters

First of all, let's simplify the problem by assuming that punctuation characters are part of words. When you have completed the exercise successfully, taking punctuation into account should not be too difficult. What we need is a variable that is incremented when any key (other than the space bar) is pressed, and reset to 0 when the space bar is pressed. You will need to use the **if** and **counter** objects to achieve this.



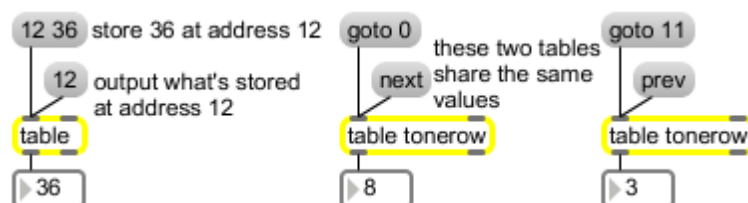
Step 3: Create a number box whose values is defined by the condition specified above (its default value is 0, it is incremented by 1 when a key other than space bar is pressed, and reset to 0 when the space bar is pressed).

Create a list of notes

Our goal is to play a different note for each letter of a word, but which notes should these be? You are welcome to choose any sequence of notes with ascending pitches (frequencies). If you want to use *musical* notes, you need to know what their frequencies are. You can easily find these values online, but to save you time, here are the frequencies of the notes of a chromatic scale starting on C4, using standard tuning:

Note	Frequency
C ₄	261.63
C [#] ₄ /D ^b ₄	277.18
D ₄	293.66
D [#] ₄ /E ^b ₄	311.13
E ₄	329.63
F ₄	349.23
F [#] ₄ /G ^b ₄	369.99
G ₄	392.00
G [#] ₄ /A ^b ₄	415.30
A ₄	440.00
A [#] ₄ /B ^b ₄	466.16
B ₄	493.88
C ₅	523.25
C [#] ₅ /D ^b ₅	554.37
D ₅	587.33
D [#] ₅ /E ^b ₅	622.25
E ₅	659.25
F ₅	698.46
F [#] ₅ /G ^b ₅	739.99
G ₅	783.99
G [#] ₅ /A ^b ₅	830.61
A ₅	880.00
A [#] ₅ /B ^b ₅	932.33
B ₅	987.77

Alternatively, you can use a harmonic scale whereby the frequency of a note in the sequence is equal to the frequency of the previous note of the sequence plus that of the first note of the sequence. For example: (100, 200, 300, 400, ...). Once you have decided what sequence of frequencies you would like to use, you need to store them. There are different ways of doing that but the suggested technique for this practical is to use the **table** object.



Step 4: Create a table that contains at least 10 frequency values

Put everything together

If you have completed the first 4 steps, you only need to connect the different parts together and you will have a patch that meets the requirements of challenge.

Taking things further

The patch you have developed has educational merits, but its usefulness is questionable. The point was not to create a useful patch, but one that will allow you to solve relevant problems in the future. However, if one tries to relate this simple system to the general concept of *Responsive environments*, one could argue that the keys pressed by a typist are significant events taking place in his/her environment. Intervention possibilities (some of them useful!) are boundless. We can currently capture typing data, which seems limited, but consider what information this can be extended into: words, with known spelling by referring to a dictionary, syntax, even possibly meaning and style, though these latter pieces of information would not be trivial to establish. If there is more than one typist in a room, what, or how fast they are collectively typing could generate *appropriate* ambient music in real time. From a simple spellchecker that notifies the typist with a sound to a responsive ambient music system, the only limit to what can be done is our imagination.