INScore ... in action

An environment for the design of interactive, augmented, dynamic musical scores.

https://inscore.grame.fr

December 19, 2024

https://scsynth.org/u/Yann_Ics

INScore version 1.31
SuperCollider version 3.13.0

This short article aims to illustrate one way to use the open-source software INScore, like a tutorial introduction within the SuperCollider context. Furthermore, this could be the first step toward developing an interface as an extension of SuperCollider.

* * *

Here is the context

I am studying on a musical project named *Untitled* #2 developed in SuperCollider as an algorithmic and synthesis composition. In short, a third-party application generates a melody based on the dynamic Markov chain and an *ostinato* as an emergent phenomenon due to the structure of a resulting network. This said application is about an AI developed in Common Lisp called *Neuromuse3* based on a connectionism network and trained with the encoded voices of the first book of J. S. Bach's *Das Wohltemperirte Klavier* (BWV 846 – 869). All these data are sent via the OSC protocol. Then SuperCollider interprets them musically via different syntheses. A GUI mixer allows one to formalize the work in a real-time performance.

From this point, I imagined that one musician performer could interact by improvising according to some guidelines displayed through the INScore viewer.What should shown is the harmonic context (*in-time* with a before and an after to anticipate the oncoming) as note names plus optional transitional notes, and the rhythm related to this event. So, there is no direct interaction with INScore, only indications to serve the musician playing.

* * *

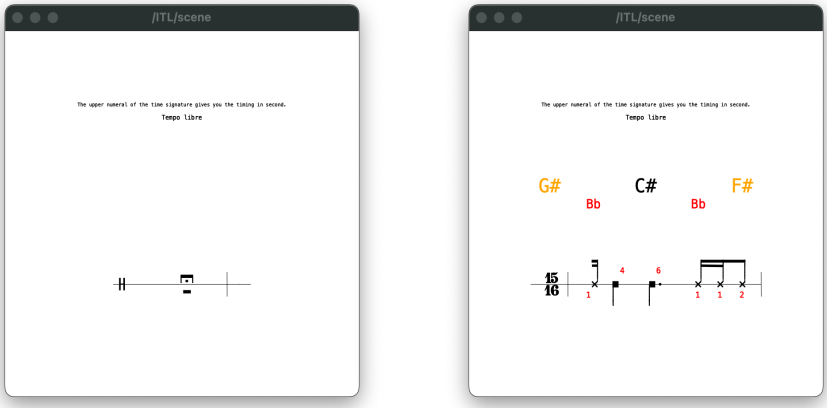
The coding part

Beforehand, to initialize the INScore display, some preliminary setup needs to be written on a file to be loaded remotely. These setups mainly consist of positioning each object related to each other. Although this can be done dynamically from a third-party application via OSC messages.

```
# clear the scene
/ITL/scene/* del;
/ITL/scene width 1;
# ---
/ITL/scene/timing set txt "The upper numeral of the time signature gives you
 the timing in second.";
/ITL/scene/timing y -0.6;
/ITL/scene/timing scale 0.8;
/ITL/scene/tempo set txt "Tempo libre";
/ITL/scene/tempo y -0.53;
# harmony
# --- previous
/ITL/scene/a1 set txt "";
/ITL/scene/a1 scale 3;
/ITL/scene/a1 color 'orange';
/ITL/scene/a1 y -0.15;
/ITL/scene/a1 x -0.55;
# --- transitional previous
/ITL/scene/a2 set txt "";
/ITL/scene/a2 scale 2;
/ITL/scene/a2 color 'red';
/ITL/scene/a2 y -0.05;
/ITL/scene/a2 x -0.3;
# --- current
/ITL/scene/a3 set txt "";
/ITL/scene/a3 scale 3;
/ITL/scene/a3 y -0.15;
/ITL/scene/a3 x 0;
# --- transitional next
/ITL/scene/a4 set txt "";
/ITL/scene/a4 scale 2;
/ITL/scene/a4 color 'red';
/ITL/scene/a4 y -0.05;
/ITL/scene/a4 x 0.3;
# --- next
/ITL/scene/a5 set txt "";
/ITL/scene/a5 scale 3;
/ITL/scene/a5 color 'orange';
/ITL/scene/a5 y -0.15;
/ITL/scene/a5 x 0.55;
# rhythm (pause)
/ITL/scene/rtm set gmn '[ \meter<""> \staffFormat<style="1-line">
 \clef<"perc"> \fermata<"long">(_/1) | empty*1/32 \staffOff ]';
/ITL/scene/rtm y 0.4;
/ITL/scene/rtm scale 0.8;
# set scene font
/ITL/scene/* fontFamily Courier;
```

Each line is defined by the discriminant or tag `/ITL/`, the name of the scene, i.e. the window viewer, plus an identifier, followed by a keyword and its parameter(s) if required, which are well documented with examples in the INScore application folder as PDF and INScore files. Here, only the lines concerning the text of harmony and the score `gmn` (acronym of guido musical notation) will be dynamically set according to the context. The code is saved (in a file named `ics.inscore` currently) and must be loaded in a remote server (from the INScore API) in order to be accessible from any device.

The score consists of displaying in the INScore viewer the previous harmony, the current harmony, and the next harmony plus the passing notes as a set of note names, and the rhythm partition.



(a) initialization and waiting state ... (b) ... in action ...

Figure 1: This is what should be displayed on the musician's device.

On the SuperCollider side, I receive data from a third-party application – namely *Neuromuse3*.

```
// opening incoming port
thisProcess.openUDPPort(7772); // ostinato
```

Note: In this case, the data are collected through OSC but can be obviously set or generated within SuperCollider. Also, I had to use SC as a router because as long as there is no standard in Common Lisp (it exists an interface within INScore but requires Lispworks which is not free and even less open source) I have taken side to code in SC. The Common Lisp interface remains hence to be done or at least extended to SBCL and CCL among others.

Then, SC needs to initiate the network address to communicate with INScore, and incidentally, the INScore file previously discussed.

```
// set sending address
~inscore = NetAddr.new("10.25.172.200", 7000);
// load init setup
~inscore.sendMsg("/ITL/scene", "load",
                 "http://remote.server/ics.inscore");
```

Note: The initialization file needs to be accessed from a remote server for convenient purposes, mainly because like so it is always possible to load the file directly from any device. Anyhow the IOS INScore version does not allow to loading of files in the API.

From this point, each specific data has to be sent in real-time with the component of the `/ITL/` concerned scene according to the following syntax:

```
<net-address>.sendMsg("/ITL/scene/identifier", <remaining-message>)
```

The remaining message is a list of parameters as strings or numbers separated by commas.

Note: I coded some methods to format notes, chords, and musical notations, according to the Guido syntax (see lines 286 to 391 of `gsa.sc`) intended to the 'remaining message':

- `midiguido` – Integer/Array *arg* [dur]
- `degguido` – Intger *args* [dur, range, to, asChord]
- `tag` – String *args* [tag-name, rest-arg(s)]

For obvious convenience, I made functions to well-format the messages to be sent according to raw data.

```
~harmonyINScore = {
    | prevOst, degOst, intOst, nextOst, netaddr |
    var nn=["C","C#","D","Eb","E","F","F#","G","G#","A","Bb","B"];
    var cn = degOst.midiguido(noteNames: nn);
    var res = Array.with(
        [if(prevOst.asArray[0].isNil)
            {""} {prevOst[0].midiguido(noteNames: nn)}],
        [if(prevOst.asArray[1].isNil)
            {""}
            {(prevOst[0]+prevOst[1]-11)
                 .mod(12)
                 .midiguido(noteNames: nn)}],
        [cn],
        [(degOst+intOst-11).mod(12).midiguido(noteNames: nn)],
        [nextOst.midiguido(noteNames: nn)]);
    res.do{
        |it,i| netaddr.sendMsg("/ITL/scene/a%".format(i+1), "set",
            "txt", it.join(" "))
    }
}
```

* * *

```
~gmnINScore = {
    | rtm, denom=16, netaddr|
    var midinote=67, dy=8 /*stem size*/, tmp="", res="", score;
    rtm.do{ |dur|
        if (dur/(denom/4) >= 1)
        {
            if (tmp.isEmpty.not)
               { res = res+tmp.tag(\bm, [\dy, dy]); tmp="" };
            res = res+format("\\stemsDown \\noteFormat<'square'>
                \\text<'%', font='Courier', fsize=8pt, color='red',
                 dy=6, dx=4> ", dur)
                    ++ midinote.midiguido(dur/denom)
        }
        {
            tmp = tmp + format("\\stemsUp \\noteFormat<'x'> \\text
               <'%', font='Courier', fsize=8pt, color='red', dy=-2,
                    dx=2> ", dur) ++ midinote.midiguido(dur/denom)
        }
    };
```

```
    // the gmn score
    score = "[ \\meter<'" ++ format("%/%", rtm.sum, denom)
            ++ "'> \\staffFormat<style='1-line'> \\clef<'none'>
                \\bar \\dotFormat<dy=0,dx=0,size=1.2> "
            ++ if (tmp.isEmpty) {res} {res+tmp.tag(\bm, [\dy, dy])}
            ++ " \\bar ]";
    // send to INScore
    netaddr.sendMsg("/ITL/scene/rtm", "set", "gmn", score)
}
```

* * *

Structure of the received OSC messages in SC:

```
#   tag  |  seq   | reldur |  deg   |  int   |  nextcli  |     opt_rtm       | realdur
[ msg.[0], msg.[1], msg.[2], msg.[3], msg.[4],  msg[5..7], msg[8..msg.size-2], msg.last ]
```

* * *

Receiving OSC messages from N3 and sending OSC messages to INScore through SC's functions previously defined (and incidentally where data are interpreted for synthesis and analysis computation):

```
OSCdef(\ostinato,
    {
        arg msg, time, addr, recvport;
        //msg.postln;
        if (msg[1].asInteger != 0)
        {
            ~gmnINScore.(
                msg[8..msg.size-2].asInteger,
                netaddr: ~inscore
            );
            ~harmonyINScore.(
                ~prevOst,
                msg[3].asInteger,
                msg[4].asInteger,
                msg[5..7].asInteger[1],
                ~inscore);
            ~prevOst = [ msg[3].asInteger, msg[4].asInteger ]
        }
        {
            ~inscore.sendMsg("/ITL/scene/rtm", "set", "gmn",
                "[ \\meter<\"\"> \\staffFormat<style=\"1-line\">
                    \\clef<\"perc\"> \\fermata<\"long\">(_/1) |
                    empty*1/32 \\staffOff ]");
            5.do{ |i| ~inscore.sendMsg("/ITL/scene/a%".format(i+1),
                 "set", "txt", "")};
            ~prevOst = nil
        }
    }, '/N3', recvPort:7772
)
```

* * *

Discussion

In fact, the principle of INScore is rather simple as long as you are ready to learn the syntax, which is relatively intuitive. Moreover, JavaScript language for advanced or specific use offers the possibility to solve any requirement. On the Guido engine side, things seem to be limited especially for advanced notation, which can be worked around with MusicXML files or images generated by any third-party application. Anyway, we have to take into account that *it inevitably ends up being a sight reading challenge*. It is certainly not a panacea when it comes to animated notation, but I was seduced by its simplicity and also by the fact that the application met my needs.

Beyond what I have described, INScore allows wide interactivity not only as I did in the sense of third-party application toward INScore, but also to share any kind of data using the same OSC protocol toward any third-party application. All of these are listed in chapters 16 *Sensors* page 63 and 17 *Events and Interaction* page 68 of the document named *INScore OSC Messages Reference v.1.31* (referenced in Resources).

Now, what can be done in the SuperCollider context is a class to well-format the message to send, which can be seen as an extension (or an interface according to the INScore terminology), plus some convenient methods as I did for my work. Despite my modest contribution to what INScore can do, for sure I will use it more and more in that direction. Anyway, I plan to extend the proposition to the technical aspects of the instrument involved like the ocarina and the clarinet, toward *Untitled* `#2` ...

* * *

Resources

- INScore message syntax :
  https://inscoredoc.grame.fr/rsrc/INScoreMessages.pdf
- Guido syntax online : https://guidoeditor.grame.fr/
- Others :
  - `// install SuperCollider package gsa.quark`
    `Quarks.install("https://github.com/yannics/GSA/gsa")`
  - *Neuromuse3* & *Untitled* `#2` as study:
    https://github.com/yannics/Neuromuse3/
  - BWV corpus : http://www.titanmusic.com/data.php