HOMENAJE A CERVANTES: FOR VIOLIN, COMPUTER AND PROJECTIONS

Richard Hoadley

HOMENAJE A CERVANTES: FOR VIOLIN, COMPUTER AND PROJECTIONS

Richard Hoadley
rhoadley.net
music@rhoadley.net

ABSTRACT

This is a presentation of the dynamic score *Homenaje a Cervantes* (Homage to Cervantes) created for violin, computer and projections, originally commissioned for and first performed at the University of A Coruna, Spain in May 2017. The piece has been composed using the software packages SuperCollider and INSCORE; the violin part should be played live from a laptop screen or a projection. The texts used are the original Cervantes text, an English translation and a series of original poems created specially for this project by the poet Phil Terry.

1. INTRODUCTION

This paper seeks to demonstrate selected aspects of the technical and aesthetic structure of the composition *Homenaje a Cervantes*; in particular it exposes and examines the layers of code constructed to generate the notations for performance by the violinist. Algorithmic material manipulating audio, text and graphics is generated through scheduling of functions, audio analysis and in certain configurations physical computing or a combination of these elements. Functions and processes are constructed within SuperCollider's native language (**sclang**)[1]. Parts of the piece have been arranged to allow interaction with a dancer should one be available. These parts (primarily the sustained violin material towards the end of the piece) utilise the Microsoft Kinect for Xbox One sensor. SuperCollider algorithms generate time, frequency, amplitude and control values which are then sent either to the SC synth or via Open Sound Control (OSC), (using a custom SC class developed by the author) to the programme INSCORE [2]. INSCORE is able to generate and control a variety of notations, including common practice music notation. While, for both technical and musical reasons, I am currently most involved with the latter aspect, I am involved in other collaborative projects making use of generative graphics and text.

One way in which SuperCollider can be used is by selecting a line or larger region of code and 'evaluating' it. The relevant code is rendered immediately (i.e. just in time, or as soon as possible). If a section of code is enclosed within two parentheses — (*code here*) — inserting the cursor at any point between these and pressing <enter> or equivalent will evaluate that section of code. As the piece is to a significant extent performed through such live evaluation of code segments it can be said to exploit certain live coding practices. Due to the complex construction of much of the code, however, only a minimal quantity of actual typed coding is undertaken during any given performance. Deciding what is coded, when and how during a live coding performance is a fascinating issue worthy of significant future research. There is further discussion of live coding in my own work in [3] and more about the generic idea of live coding in [4].

To fully appreciate what follows some familiarity with SuperCollider's built in language **sclang** and Guido notation is desirable. The code is provided here for illustrative purposes only. It will not run successfully without a variety of dependencies.

At various points in the text I refer to passages from a demonstration video. This can be found at the following web address: http://rhoadley.net/video/homage.

2. TITLE

Figure 2 shows *Homenaje*'s title screen and **code listing 1** shows the relevant 'live' source code. The latter demonstrates an important aspect of my use of INSCORE in this case: the necessity to 'reset' particular groups of elements at strategic moments. In this case, if *Homenaje* has been played or rehearsed and we wish to return to the beginning, many objects will have been displaced and reformatted. The apparently redundant codes here, such as moving, scaling and (re-)setting the origin are necessary for this reason. The code also shows the way in which even in this digital environment, there is still a need for reference to physical aspects of the score (e.g. page height and width).

Lines of code beginning `~homage` reference the INSCORE class for SuperCollider prepared by the author. This is purely a convenience class designed to make the coding of Guido music notation within INSCORE easier and more straightforward. In the following case, the following line of code:

```
~homage.note("homageWin", 0, "a")
```

outputs to INSCORE the following OSC string:

```
/ITL/homageWin/score0 set gmn [a]
```

which INSCORE converts into the notation snippet shown in figure 1. The INSCORE class, once complete, including

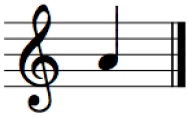


Figure 1: A simple INScore score.

properly prepared SC help files, will be made available for public download [1] . As can be seen in line 19 of **code listing 1**, the class makes use of INSCORE's ability to make use of fully formattable HTML code.

Figure 2: the title screen.

```
1
2  // the code listing below displays the title and formats the
       score
3  (
4  ~pageWidth="28cm";
5  ~pageHeight="36cm";
6  ~myNoteArray = " "; // empty score
7  ~homage.note("homageWin", 0, ~myNoteArray);
8  ~homage.scale("homageWin", "score", 0, 0.5); // size the score
9  ~homage.move("homageWin", "score", 0, -1.4, -0.8, 1); //
       position the score
10 ~homage.origin("homageWin", "score", 0, -1, -1); // position
       the score
11 ~homage.htmlFull("homageWin", "html", 0, "Helvetica", "50pt",
       "normal", "normal", 0.1, ~argbConvert.value([255, 100,
       100, 100]), " ");
12
13 Task({ 0.25.wait;
14    ~homage.move("homageWin", "text", 0, 3.0, 3.0);
15    ~homage.scale("homageWin", "html", 0, 1);
16    ~homage.rotate("homageWin", "html", 0, 0, 0); // due to
         later transformations, make sure the rotation of the text
         is reset
17    ~homage.origin("homageWin", "html", 0, 0, 0);
18    ~homage.move("homageWin", "html", 0, 0, 0);
19    ~homage.htmlFull("homageWin", "html", 0, "Helvetica",
         "50pt", "normal", "normal", 0.1, ~argbConvert.value([255,
         100, 100, 100]), "homenaje a cervantes"); // this is the
         title
20 }).play;
21 )
```

Code listing 1: the title screen listing

3. MELODY

Once the title has faded, sounds of horses walking and the wind blowing emerge, creating an atmospheric sonic

[1] The current version can be downloaded here: http://rhoadley.net/inscore/INScore.sc , but without documentation

Figure 3: Initial melody.

backdrop. Blurred coloured areas fade in and out. The colours used (green, blue, grey and brown) are algorithmically generated variants of four prominent colours present in the landscape of La Mancha[5] (see figures 4 and 5, the latter of which shows the background colours in use).

Figure 4: Windmills and landscape at La Mancha, Spain, showing the colours of the landscape used as backdrops for the notations.

Figure 5: Example of colours from figure 4 used as backdrop

Figure 3 shows a rendition of *Homenaje*'s opening melody, and **code listing 2** the equivalent code. Please refer to the inline comments for more explanatory detail about the code itself.

```
1
2  (
```

```
~playViolin = true; ~myType = 1; // '~playViolin' set to true
      will play an audio rendition of the melody.  If a real
      violinist is available, this should be set to false.
      '~myType' provides options in the style of the rendition.
      If '~playViolin' is true, then '~myStyle' set to 1 will
      play louder, more vigorous sounding violin samples

Task({
   // this is the Task that generates a version of the melody.
      Note that there are still a lot of formatting issues to be
      dealt with as music notation is so predominantly a
      graphic/semantic language.

   ~homage.colour("homageWin", "text", 0, [0,0,0,255]); // set
      the colour
   4.do({|i| ~homage.text("homageWin", 0, text: 4 - i);
      0.5.wait; }); // generate a count in

   // the three functions using ".stop" end any already running
      function.  We then clear the score and run the melody
      generator (~doViolinMelWithCursorFunc) again.
   ~colourFadeTask.stop; ~homageViolinMelTask.stop;
      ~homageCursorTask.stop;
   ~homageViolinMelody = ~scoreFormat ++ " \\intens<\"f\",
      dy=-10hs>";

   // ~doViolinMelWithCursorFunc is the main function, which
      calls on a further function to generate the melody, play
      it, display it and synchronise it with a cursor to aid
      real-time performance
   ~doViolinMelWithCursorFunc.value(rrand(6, 15), [55, 74], 2,
      [~minMaxAmp[0], ~minMaxAmp[1]], 120, ~myType, ~playViolin,
      durFactor: 8, countIn: 2);

   // the below displays the instruction "sempre tenuto e
      marcato".
   // both ~doViolinMelWithCursorFunc and ~colourFadeGlobal
      fade the relevant element.
   ~homage.text("homageWin", 0, text: "sempre tenuto e
      marcato"); ~homage.colour("homageWin", "text", 0,
      [0,0,0,255]);
   ~colourFadeGlobal.value("~homage", "homageWin", "text", 0,
      [255, 0], 4);
}).play; // end of melody generating task.
)
```

Code listing 2: Top level code of opening melody

Code listing 2 is top level code. In performance, evaluation of this complete section causes the main work to be done by the function `~doViolinMelWithCursorFunc` (see **code listing 3**). This function synchronises `~doHomageViolinMel` (line 4) with a moving cursor intended to help the performer.

```

(
~doViolinMelWithCursorFunc = ({ arg noteNum=4, range=[55, 62],
      octaves=2, amplitude=[0.6, 0.8], time=120, type = 1,
      play=true, durFactor=1.0, countIn = 2; // bpm
   var cursorPosX=0.1, cursorPosY = -0.5;

   // colour the score black
   ~homageGuido[0][10] = [0,0,0,255];
      ~homage.colour("homageWin", "score", 0, [0,0,0,255]);

   ~homageViolinMelTask.stop; ~homageCursorTask.stop; // stop
      any existing tasks
   ~homage.scale("homageWin", "score", 0, 0.45); // set the
      scale
   ~homage.tempo("homageWin", "cursor", 0, time); // set the
      cursor tempo

   // the function that generates the melody itself
   ~doHomageViolinMel.value(noteNum, range, octaves,
      amp:{amplitude.choose}, type: type, wait: true, play:
      play, sayDone: true, waitFactor: (60/time)*0.5, mm: time,
      scoreSize: ~scoreSize, durFactor: durFactor, countIn:
      countIn);

   ~homage.date("homageWin", "cursor", 0, 0); // the cursor

   // describe the movement of the cursor
   ~homageCursorTask = Task({
      var date, noteWait, totalWait=0;
      noteNum.do({|i|
         ~homage.position("homageWin", "cursor", 0, cursorPosX,
      cursorPosY);
         if ( cursorPosY == -0.5, { cursorPosY = -0.7 }, {
      cursorPosY = -0.5 } );
         noteWait = (60/time);
         totalWait = totalWait + noteWait;
         noteWait.wait;
      });

      (totalWait*0.75).wait;

      ~homageGuido[0][10][3] = 255; // set alpha channel to
      opaque
      ~colourFadeGlobal.value("~homage", "homageWin", "score",
      0, [~homageGuido[0][10][3], 0], 3); // fade the score
   }).play;
});
);
```

Code listing 3: ~doViolinMelWithCursorFunc

The violin melody itself is generated (and optionally either played, displayed or both) by `~doHomageViolinMel`, shown in **code listing 4**. Available arguments include:

- the number of notes;
- the range of the notes in terms of midi pitch;
- the range of the notes in terms of the number of octaves potentially covered;
- the variety of durations to be used and the weightings of those durations;
- the duration and amplitude of each audio rendered note;
- whether to display the score at once or in real time, whether to play it, whether to display it at all.

The process of 'composing' these algorithms is itself an essential part of the creative act: each compositional gesture will require different musical options. Arguments are added (or, more rarely, taken away) as the aesthetic need arises. If a function is used in a subsequent composition it is likely that these arguments will be tidied up in order to promote clarity and ease of use.

```
(
~doHomageViolinMel = ({ arg num=10, range=[67, 74], numOct=2,
      durRange=[0,1,2,3,4,5],
      durWeight=[0.15,0.24,0.21,0.21,0.12,0.08], durFactor=1.0,
      amp=1.5, wait=true, play=true, display=true,
      waitFactor=0.25, type=1, halo=[4.0, 10.0], sceneNum=0,
      sayDone=false, mm=120, scoreSize =
      "\\pageFormat<w=20cm,h=24cm>", countIn = 2;
   var durDictionary= Dictionary.newFrom(List[0, " ", 1,
      "*1/8", 2, "*1/4", 3, "*1/4.", 4, "*1/2", 5, "*5/8", 6,
      "*1/2.", 7, "*7/8", 8, "*1/1", 9, "*9/8"]), score =
      scoreSize + "\\clef<\"treble\"> \\meter<\"2/4\">
      \\tempo<\"[1/4]=" + mm + "\" ,dx=-5, dy=4> _ _", dur = 1,
      note = rrand(range[0], range[1]), prevNote = 0, octave = [
      0, 1 ].choose, chordInt = 0, chordDurString, chordDurNum;

   ~homageViolinMelTask.stop; // stop any existing running tasks

   if ( ~homageViolinMelody != "", { score =
      ~homageViolinMelody } );

   ~homageViolinMelTask = Task({
      num.do({
         if ( durWeight == "choose", { dur = durRange.choose},
      { dur = durRange.wchoose(durWeight) } ); // dur is the
      number (1-8)

         octave = numOct.rand; // choose octave

         // if there is a repeated note, get a new one
         while ( { note == prevNote },
            {
               note = rrand(range[0], range[1]); // pick a
      pitch within the range
               note = note + (octave*12); // transpose the
      pitch to the octave chosen earlier
            } );

```

```
22          prevNote = note; // keep a record of the chosen note
23
24          // if there's a chord, you have to notate it
        differently (a chord is indicated by setting 'dur' to 0)
25
26          if ( dur == 0, {
27              chordInt = [8, 9, 10].choose; // choose the chord
        interval.  The interval will be a minor 6th, a major 6th
        or a minor 7th as these are relatively straightforward
        intervals for a violinist to play.  If a chord is needed,
        'dur' is set to zero, so we have to pick another duration.
         We pick from the original list, but without the zero
        which we don't want to pick again.
28
29              if ( durWeight == "choose", { chordDurNum =
        durRange.choose}, { chordDurNum =
        durRange.wchoose(durWeight); } );
30
31  // if zero has been chosen, pick something else from the list
        until it isn't zero (the chord still needs a duration)
              while ( {chordDurNum == 0}, {
32                if ( durWeight == "choose", { chordDurNum =
        durRange.choose}, { chordDurNum =
        durRange.wchoose(durWeight); } );
33              } );
34
35              chordDurString = durDictionary[chordDurNum]; //
        string of the chord duration (i.e. "*1/4")
36
37  // add to the score string, if we are using a chord
38              score = score + "{" ++ ~guidoNoteMap.value(note) ++
        chordDurString ++ "," ++
        ~guidoNoteMap.value(note+chordInt) ++ "}";
39
40          }, { // add to the score string if there *isn't* a
        chord
41              score = score + ~guidoNoteMap.value(note) ++
        durDictionary[dur];
42          } );
43
44          // send to INScore if display is true
45          if ( display == true, { ~homage.note("homageWin",
        sceneNum, score ) } );
46
47          ~homageViolinMelody = score; // store the score data
        in an environment variable
48
49          if ( play == true, { // do we want to hear it?
50              if ( dur == 0, { // if it's a chord
51
52                  // for aggressive playback, type = 1
53                  if (type == 1 , {
54  ~playViolinArcoSforzando.value(((chordDurNum)*waitFactor)*durFactor,
        0.01, 0.5, 0.49, (note), 1.0, amp);
55  ~playViolinArcoSforzando.value(((chordDurNum)*waitFactor)*durFactor,
        0.01, 0.5, 0.49, (note + chordInt), 1.0, amp); // second
        note
56                  },{
57  ~playViolinArcoGentle.value(((chordDurNum)*waitFactor)*durFactor,
        0.01, (note + chordInt), 1.0, amp)
58                  } );
59
60  // the 'halo' effect generates a longer (usually quieter)
        sustained note emanating from each `formally' played note,
        creating a `halo' of sound.  To switch off use halo = [0,0]
61                  ~playViolinArcoGentle.value(rrand(halo[0],
        halo[1]), 0.01, (note + chordInt), 1.0, amp);
62              });
63
64              if (type == 1 , { // aggressive or gentle?
65
66          ~playViolinArcoSforzando.value((dur*waitFactor)*durFactor,
        0.01, 0.5, 0.49, note, 1.0, amp)
67                  }, {
68           ~playViolinArcoGentle.value((dur*waitFactor)*durFactor,
        0.01, note, 1.0, amp);
69              } );
70
71          ~playViolinArcoGentle.value(rrand(halo[0], halo[1]),
        0.01, note, 1.0, amp);
72          }); // if play is false, we skip the above
73
74
75          if ( wait == true, { // do we want the whole score at
        once (false, we don't want to wait)?
76              if ( dur == 0, { ((chordDurNum)*waitFactor).wait;
        }, {(dur*waitFactor).wait; } );
77          } );
78      }); // end do
79
80      // tell us if you've finished playing/printing the melody
81      if ( sayDone == true, {
82          "homageViolinMelody done".postln;
83          ~homage.colour("homageWin", "score", 0, [0,0,100,255]);
84          ~homageGuido[0][10] = [0,0,100,255];
85      });
86  }).play;
87  });
```

Code listing 4: ~doHomageViolinMel

3.1 The Aesthetics of Melody Generation

Straightfoward algorithms govern the details of the generation of this initial melody. It terms of compositional process, the ideas are developed in ways that mirror (my own) 'standard', non-digital compositional methods, focusing on traditional musical elements such as atmosphere, tempo, dynamics, articulation and tessitura. The development of the construction of the algorithm relies very much on trial and error, although previous experience gained through the previous development of algorithms has a substantial influence.

In **code listing 4**, one of the first 'decisions' involves the choice of duration for a new event. This involves a variety of durations (line 2):

```
durRange=[0,1,2,3,4,5]
```

and corresponding probability weightings:

```
durWeight=[0.15,0.24,0.21,0.20,0.12,0.08]
```

Because the creative intention was to create short, intensive, active and rhythmically metrical phrases, `durRange` only consists of whole notes, where '1' represents one quaver's duration. As the intended mood of these melodies is vigorously active, the weightings favour the shorter durations with the most weighted duration being one crotchet (one quarter note) in length. Ultimately, the final values for these are achieved through repeated generation and regeneration until the aesthetically required balance is achieved.

Subsequent algorithms choose the octave of the pitch, and the pitch itself, in this case a randomised value from the provided range. A previous duration value of zero indicates that a chord should be created, and as the violinist must navigate the dynamic part at sight, only the most straightforward violin diads are allowed: a minor or major sixth or a minor seventh (**code listing 4**, line 26+).

The algorithm also includes the possibility of specifying one of two different 'styles' of playback (should playback be required in the absence of a human violinist). These styles are either using louder samples with a sharp attack or quieter samples with a more gentle attack.

4. TEXTS, CHORDS AND LINES

The next section of *Homenaje* (from about 2:50 of the demonstration video) introduces sections of text from Cervante's original version of *Don Quixote* [2] as well as an English translation [3]. Text excerpts are accompanied by a variety of musical figures, described as either *lines* or *chords*. Each of these types of figures has originally appeared in earlier compositions (e.g. *Calder's Violin*[6] and *How To Play the Piano*[7]).

4.1 Texts

The digital texts were converted into UTF-8 format text files allowing for the straightforward inclusion of Spanish accents. During the set-up of the piece these files are loaded into environment variables:

[2] Original available here
[3] English translation available here

```
g = File(~path ++
    "text/quijote_sp_text_only_utf-8.txt","r");
~homSpWords = g.readAllString;
g.close;
```

The entire text is then converted to lowercase so avoiding difficulties of sentence construction using upper-case lettering:

```
~stanzaTextInputSp = ~homSpWords.toLower;
```

Only small pieces of the text are chosen for display. Due to the size of the entire text and the time it can take to choose a range within it, a smaller section is chosen in advance, and the smaller chunk to be used is taken from that and stored in another environment variable. In the below, a starting point is chosen from within the entire text, and then a chunk of 1000 characters is chosen from that starting point:

```
// used in ~generateFullStanza
~randPlaceInStream =
    ((~stanzaTextInputSp.size)-1200).rand;
~stanzaTextInputSpSmall =
    ~stanzaTextInputSp[~randPlaceInStream..
(~randPlaceInStream+1000)];
```

Below is the function ~homageTextFunc which automates this process. The arguments mean that the main function will be run just once, and will produce a 'stanza' of one word and one line. After it has appeared, it will fade out in between 0.2 and 6 seconds. The language chosen will be English (a zero value is Spanish and a value in between is the probability that one language or the other will be chosen — a value of 0.5 will mean each language is equally likely. At the same time a fragment is chosen from audio files of readings of each version. This will have an amplitude of between 0.1 and 0.35 (maximum amplitude is nominally 1.0), and the function will pause between 1 and 4 seconds between lines and 2 and 7 seconds between 'stanzas':

```
~homageTextFunc.value(1, [1, 1], [0.2,
    6.0], 1.0, [0.1, 0.35], [1.0, 4.0],
    [2.0, 7.0]);
```

Code listing 5 contains a full listing of the function.

```
(
~homageTextFunc = ({ arg doNum=40, stanzaDoNum=[1, 6],
    stanzaFadeTime=[0.2, 6.0], langVar=0.5, ampArray=[0.1,
    0.35], lineWait=[1.0, 4.0], stanzaWait=[2.0, 7.0];
  ~stanzaRepeatTask.stop; ~stanzaTask.stop; // stop any
    running tasks
  ~stanzaRepeatTask = Task({
    var lang = ~voiceBufEn;
    doNum.do({ |i|
      ("~homageTextFunc stanza:" + i ++ "/" ++
    doNum).postln; // keep us informed
      (
        ~stanzaWordNum = [3, 6]; ~stanzaLineNum = [1, 3];
        ~stanzaTask = Task({
          var stanzaTaskNum = rrand(stanzaDoNum[0],
    stanzaDoNum[1]);
          stanzaTaskNum.do({ |i|
            ("stanzaTaskNum:" + i ++ "/" ++
    stanzaTaskNum).postln;

    ~generateFullStanzaOrig.value(rrand(stanzaFadeTime[0],
    stanzaFadeTime[1]));
            if ( langVar.coin, { lang = ~voiceBufSp }, {
    lang = ~voiceBufEn });
            // here is the audio taken from readings of
    the book in Spanish and English.  Currently the audio is
    chosen at random from within the files
            if ( 0.99.coin, { ~fragmentSynthEnv =
    Synth.new("fragmentSynthEnv", [\dur, rrand(3.1, 8.0),
    \amp, rrand(ampArray[0], ampArray[1]), \bufnum,
    lang.bufnum, \startPos, rrand(0, lang.numFrames), \rate,
    1, \attack, 0.3, \sustain, 0.4, \release, 0.3, \effectBus,
    ~effect]); } );
            rrand(lineWait[0], lineWait[1]).wait;
          });
        }).play;
      );
      rrand(stanzaWait[0], stanzaWait[1]).wait;
    });
  }).play;
});
);
```

Code listing 5: ~homageTextFunc

```

(
// initialise variables
if ( ~stanzaLayerNum == nil, { ~stanzaLayerNum = 0 });
if ( ~stanzaSceneArray == nil, { ~stanzaSceneArray = [
    nil,nil,nil,nil,nil,nil,nil,nil,nil,nil,nil ] }); // this
    array is to keep track of which object layers contain what
    information and whether they are fading, rotating, growing
    or shrinking, etc.
~fullStanzaFontSize=10;

// the function
~generateFullStanzaOrig=({ arg fadeTime=((4.0.rand)+0.5);
  var colourRand=155.rand, alphaRand=(155.rand)+100,
    hexColour, textStreamPortion="", randPlaceInStream,
    fadeInAndOutRoutine;

  // choose a layer that is not already involved (e.g. fading)
  while ( { (~homageHTML[~stanzaLayerNum][12][1] == true) ||
    ~stanzaSceneArray.includes(~stanzaLayerNum) == true }, {
    ~stanzaLayerNum = 12.rand } );

  // set the font size, face, colour and alpha values
  ~homageHTML[~stanzaLayerNum][15] =
    ((~fullStanzaFontSize.rand)+5).asString ++ "pt";
  ~homageHTML[~stanzaLayerNum][14] = ~fontArray.choose;
    hexColour = ~argbConvert.value([alphaRand, colourRand,
    colourRand, colourRand]);
  ~homageHTML[~stanzaLayerNum][10] = [colourRand, colourRand,
    colourRand, alphaRand];
  ~homageHTML[~stanzaLayerNum][11] = alphaRand;

  // choose the larger section
  randPlaceInStream =
    ((~stanzaTextInputSpSmall.size)-120).rand;

  // choose the smaller text from within this
  textStreamPortion =
    ~stanzaTextInputSpSmall[randPlaceInStream..(randPlaceInStream+120)];

  ~stanzaText = ~createTextStreams.value(textStreamPortion,
    ~stanzaWordNum, ~stanzaLineNum);

  ~phonemeSigns[~stanzaLayerNum] = ~stanzaText;

  // this actually sets the text on screen
  ~homage.htmlFull("homageWin", "html", ~stanzaLayerNum,
    ~homageHTML[~stanzaLayerNum][14],
    ~homageHTML[~stanzaLayerNum][15],
    ~homageHTML[~stanzaLayerNum][2],
    ~homageHTML[~stanzaLayerNum][3], 0.1, hexColour,
    ~phonemeSigns[~stanzaLayerNum]);

  // move the text
  ~homageHTML[~stanzaLayerNum][4] = (rrand(~winSize[0].neg,
    ~winSize[0]))*0.6;
  ~homageHTML[~stanzaLayerNum][5] = (rrand(~winSize[1].neg,
    ~winSize[1]))*0.6;
  // set to (slightly smaller than) the window bounds
  ~homage.move("homageWin", "html", ~stanzaLayerNum,
    ~homageHTML[~stanzaLayerNum][4],
    ~homageHTML[~stanzaLayerNum][5], ~stanzaLayerNum);

  // scale the text
  ~homageHTML[~stanzaLayerNum][9] = rrand(4.0, 8.0);
  ~homage.scale("homageWin", "html", ~stanzaLayerNum,
    ~homageHTML[~stanzaLayerNum][9]);

  // now start to fade the text
  ~htmlFadeFull.value("~homage","homageWin", "html",
    ~stanzaLayerNum, ~homageHTML[~stanzaLayerNum][14],
    ~homageHTML[~stanzaLayerNum][15],
    ~homageHTML[~stanzaLayerNum][2],
    ~homageHTML[~stanzaLayerNum][3], 0.1,
    ~phonemeSigns[~stanzaLayerNum],
    [~homageHTML[~stanzaLayerNum][11], 0],
```

```
        ~homageHTML[~stanzaLayerNum][10], fadeTime);

    // collect more than one to avoid repetition when the
       actions overlap...
    ~stanzaSceneArray = ~stanzaSceneArray.add(~stanzaLayerNum);
    if ( ~stanzaSceneArray.size > 10, { ~stanzaSceneArray =
       ~stanzaSceneArray.drop(1) } );
});
)
```

Code listing 6: ~generateFullStanzaOrig

4.2 Lines and Chords

Lines and chords are two types of generative musical figures both of which have appeared in earlier compositions. More detail about lines and chords, their construction, and the manner of their representation can be found in [7].

5. WINDMILLS

One of the main focuses of *Homenaje* must be, of course, La Mancha's infamous windmills (see figure 4). I had discussed with Phil Terry whether he might have any ideas regarding the text of *Don Quixote* and he composed a series of twelve 'quixotes', each of which took a famous scene from the book and created a 'concrete' poem from each in the shape of a windmill. Below is the text of the first one:

Wicked breed
Unimaginable adventures
Monstrous giants
Happy memory

"What giants?"
Said Sancho Panza.
"Those giants that you
can see over there"

replied his master
"with long arms".

Great service, raw novice, arduous combat,
cowardly creatures [4]

We made audio recordings of Phil reading out each of the poems and these recordings accompany the concrete visualisations during performance.

Figure 9 shows its implementation visually in INSCORE. This demonstrates more of the text-based features of the programme. In the demonstration video the passage begins at about 4:40.

[4] The scene to which this refers can be found in Book 1, Chapter 8, available here

Figure 9: Quixotes: concrete poetry by Phil Terry.

Code listing 7 is rather lengthy, combining as it does complex graphical placements of textual elements.

```

~quixotes = ({ arg windmillNum = 1, num,
      wmSpeedOffsetMinMax=[0.35, 0.45], dilapidation=10,
      scale=1.0, xOffset=0.5, yOffset=0.5;

      // font size needs to be mapped onto the number of
      characters in each word or line so that the cosmetic
      appearance of the windmill graphics can be maintained.
   var font = "Garamond", text, fontSize, fontSizePt,
      fontSizeSpec = ControlSpec(26, 10, 'lin', 1, 20),
      wmSpeedOffset = [ rrand(wmSpeedOffsetMinMax[0],
      wmSpeedOffsetMinMax[1]), rrand(wmSpeedOffsetMinMax[0],
      wmSpeedOffsetMinMax[1]), rrand(wmSpeedOffsetMinMax[0],
      wmSpeedOffsetMinMax[1]), rrand(wmSpeedOffsetMinMax[0],
      wmSpeedOffsetMinMax[1]) ], windmillVarNum =
      ((windmillNum-1) * 14), var windmillTask;

   ~homageHTML.size.do({ |i|
      ~homageHTML[i][9] = scale; // define scale (html 9)
      ~homage.scale("homageWin", "html", i, ~homageHTML[i][9]);
   });

   // windmill blades (lines 0 - 3) text and size
   4.do({|i|
      ~homageHTML[i+windmillVarNum][13] =
      ~quixotesArray[num][i];
      fontSize =
fontSizeSpec.map((~homageHTML[i+windmillVarNum][13].size/12.0)-1.0);
      ~homageHTML[i+windmillVarNum][17] = fontSize.asString ++
      "pt";
      fontSizePt = ~homageHTML[i+windmillVarNum][17];
      ~homage.htmlFull("homageWin", "html", i+windmillVarNum,
      font, fontSizePt, text: ~homageHTML[i+windmillVarNum][13]);
      // move them
      ~homage.move("homageWin", "html", i+windmillVarNum,
      ~homageHTML[i+windmillVarNum][4] + yOffset * scale;,
      ~homageHTML[i+windmillVarNum][5] + xOffset * scale);
   });

   // middle lines text size
   7.do({ |i|
      ~homageHTML[i+4+windmillVarNum][13] =
      ~quixotesArray[num][i+4];
      ~homageHTML[i+4+windmillVarNum][17] = "18pt";
      ~homage.htmlFull("homageWin", "html", i+4+windmillVarNum,
      font, ~homageHTML[i+4+windmillVarNum][17], text:
      ~homageHTML[i+4+windmillVarNum][13]);
   });

   // last three lines text size
   3.do({|i|
      ~homageHTML[i+11+windmillVarNum][13] =
      ~quixotesArray[num][i+11];
      fontSize =
fontSizeSpec.map((~homageHTML[i+11+windmillVarNum][13].size/12.0)-1.0);
      ~homageHTML[i+11+windmillVarNum][17] = fontSize.asString
      ++ "pt";
      fontSizePt = ~homageHTML[i+11+windmillVarNum][17];

      ~homage.htmlFull("homageWin", "html",
      i+11+windmillVarNum, font, fontSizePt, text:
      ~homageHTML[i+11+windmillVarNum][13]);
   });

   // position of body
   if ( dilapidation == 0, { 7.do({|i|
      ~homageHTML[i+4+windmillVarNum][7][2] = 0; }) });

   // note the role of dilapidation here.  Each time the
      function is run during performance, the value of
```

```
        dilapidation increases and so the windmills gradually
        become more and more uneven.
    7.do({ |i|
        ~homage.move("homageWin", "html", i+4+windmillVarNum,
        ~homageHTML[i+4+windmillVarNum][4] + yOffset +
        rrand(dilapidation.neg*0.005, dilapidation*0.005) * scale,
        ~homageHTML[i+4+windmillVarNum][5] + xOffset +
        rrand(dilapidation.neg*0.005, dilapidation*0.005) * scale);
    });

    // create and rotate the last three lines...
    ~homage.rotate("homageWin", "html", 11+windmillVarNum, zPos:
        0);
    ~homage.rotate("homageWin", "html", 12+windmillVarNum, zPos:
        110);
    ~homage.rotate("homageWin", "html", 13+windmillVarNum, zPos:
        250);

    // and move them
    3.do({|i|
    ~homageHTML[i+11+windmillVarNum][4] =
        ~homageHTML[i+11+windmillVarNum][4];
    ~homage.move("homageWin", "html", i+11+windmillVarNum,
        ~homageHTML[i+11+windmillVarNum][4] + yOffset * scale,
        ~homageHTML[i+11+windmillVarNum][5] + xOffset * scale);
    });

    // spin the blades
    windmillTask = Task({
        var zPos1 = (rrand(1.75, 2.25)*wmSpeedOffset[0]);

        while ( { ~homageHTML[0+windmillVarNum][12][3] == true },
        {
            ~homage.drotate("homageWin", "html", 0+windmillVarNum,
        zPos: zPos1);
            ~homage.drotate("homageWin", "html", 1+windmillVarNum,
        zPos: (rrand(1.75, 2.25)*wmSpeedOffset[1]));
            ~homage.drotate("homageWin", "html", 2+windmillVarNum,
        zPos: (rrand(1.75, 2.25)*wmSpeedOffset[2]));
            ~homage.drotate("homageWin", "html", 3+windmillVarNum,
        zPos: (rrand(1.75, 2.25)*wmSpeedOffset[3]));
            0.04.wait;
        } );
    }).play;
});
```

Code listing 7: ~quixotes

Figure 10: ~quixotes showing increasing dilapidation.

Figure 11: ~quixotes flying off.

5.1 Dilapidation

One of the more expressive arguments of the function `~quixotes` (**code listing 7**) is `dilapidation`. This value determines the 'tidiness' of the windmills generated and, reflecting Don Quixote's deteriorating hold on reality, each time the function is called the value is increased and as a consequence the windmills' movements become increasingly uneven and unpredictable. Figure 10 shows two windmills demonstrating this. After a certain level of dilapidation is reached in performance, the windmills disintegrate completely (see Figure 11) and 'fly off' the 'page'.

As dilapidation increases, so too does the disintegration Phil Terry's reading of his Cervantes-inspired poems. With each verse the audio is rendered using decreasing bit and sample rates, making it increasingly incomprehensible. The effects of dilapidation are clearly visible and audible from about 6:00 in the demonstration video.

6. PICASSO

The final section of *Homenaje* features Picasso's 1955 black on white sketch of Don Quixote, his horse Rocinanate, his sidekick Sancho Panza as well as a number of windmills. The drawing was made on August 10, 1955 for the August 18-24 issue (No. 581) of *Les LETTRES françaises*, a weekly French journal directed by Aragon, in celebration of the 350th anniversary of the publication of Don Quixote, Part I [8].

The image has been cut into 23 pieces, some of which can be seen in figure 12. These pieces appear and fade along with the musical algorithms used. In the case of this scene, the musical ideas are initially based around guitar samples (along with occasional harp samples), reflecting the importance of the guitar in Picasso's output: in 2011 there was an exhibition at MOMA — *Picasso: Guitars 1912-1914* (here: https://www.moma.org/calendar/exhibitions/1088). The music consists of a variety of idiomatic guitar ideas and gestures: mordents and turns, strumming, and plucked melody lines (see **code listing 8**), which combine and build into a rhythmic texture (see about 7:40 in the demonstration video and **code listing 9**).

```

// a single guitar note
```

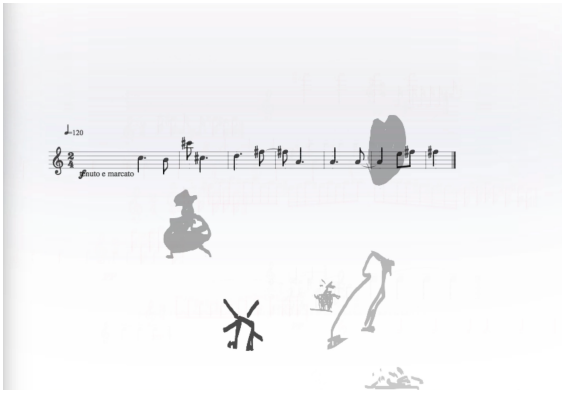
```
// throughout this section the amplitude of the guitar is governed by the environment variable ~guitarAmp which makes it easier to balance the sound in performance environments with different acoustics.

1.do({ ~guitarNote = ([ 50, 57 ].choose)-12; ~playGuitar.value(2.1, 0.01, ~guitarNote, 1.0, ~guitarAmp); }); ~picassoPartAppearFunc.value(rrand(0.001, 0.01), 1, false, false);

1.do({ ~guitarNote = ([ 50, 57, 62, 64, 67, 69, 71, 73 ].choose)-12; ~playGuitar.value(2.1, 0.01, ~guitarNote, 1.0, ~guitarAmp); });
~picassoPartAppearFunc.value(rrand(0.001, 0.01), 1, false, false);

// a few notes
// included are functions ~picassoPartAppearFunc and ~picassoPartAppearNoFadeFunc which manage the appearance and fading of the divided Picasso sketch
// the guitar notes are taken from a central array of values
(
Task({
   var numNotes = rrand(4, 8);
   var waitTime = rrand(0.05, 0.13);

   numNotes.do({ |i|
      1.do({ ~guitarNote = ([ 50, 57, 62, 64, 67, 69, 71, 73 ].choose)-12; ~playGuitar.value(2.1, 0.01, ~guitarNote, 1.0, ~guitarAmp); });
      if ( i == (numNotes-1), {
         ~picassoPartAppearFunc.value(rrand(0.001, 0.01), 1, false, false);
      }, {
         ~picassoPartAppearNoFadeFunc.value(2.0, 1, false, false, rrand(100, 255), 0);
      } );
      (waitTime*(rrand(0.95, 1.05))).wait;
   });
}).play;
);


// low strum
4.do({ ~guitarNote = ([ 50, 57, 62, 64, 67, 69, 71, 73 ].choose)-24; ~playGuitar.value(2.1, 0.01, ~guitarNote, 1.0, ~guitarAmp); });
~picassoPartAppearFunc.value(rrand(0.001, 0.01), 1, false, false);

// arg octave = 0, amp = 3.0;
~guitarStrumFunc.value(2, ~guitarAmp);

// a variety of twists and turns
~guitarTwistFunc.value(2, ~guitarAmp*0.25, true);
~picassoPartAppearFunc.value(rrand(0.005, 0.05), 1, false, false);
~guitarTwistFunc.value(6, ~guitarAmp*0.25, false);
~picassoPartAppearFunc.value(rrand(0.005, 0.05), 1, false, false);
~guitarTwistFunc.value(12, ~guitarAmp*0.25, false);
~picassoPartAppearFunc.value(rrand(0.005, 0.05), 1, false, false);

// many turns
Task({ rrand(2, 8).do({ ~guitarTwistFunc.value(12, ~guitarAmp*0.25, false); rrand(0.1, 0.4).wait;
~picassoPartAppearFunc.value(rrand(0.001, 0.01), 1, false, false); }); }).play;

Task({ 4.do({ ~picassoPartAppearFunc.value(rrand(0.001, 0.01), 1, false, false); ~guitarTwistFunc.value(12, ~guitarAmp*0.3, false); rrand(0.1, 0.4).wait; }); }).play;

Task({ rrand(4, 12).do({ ~guitarTwistFunc.value(12, ~guitarAmp*0.25, false); rrand(0.1, 0.4).wait; }); }).play;

~playGuitar.value(0.1, 0.01, 55, 1.0, ~guitarAmp);
~picassoPartAppearFunc.value(rrand(0.001, 0.01), 1, false, false);

~playTinyGuitarStream.value(rrand(4, 24), true, {rrand(~guitarAmp*0.01,~guitarAmp)} );
~picassoPartAppearFunc.value(rrand(0.001, 0.01), 2, false, false);

~playTinyGuitarStream.value(rrand(4, 24), true, {rrand(~guitarAmp*0.01,~guitarAmp)});

// guitar 'strum' function in funcs
~guitarStrumFunc.value(14, ~guitarAmp);

// a small, quiet, relatively metrical guitar melody:
~playTinyGuitarStream.value(rrand(4, 24), true);
~guitarStrumFunc.value(2, amp: ~guitarAmp);

// play short burst of guitar melody
(
Task({

   rrand(4, 12).do({
      ~guitarNote = [ 50, 57, 62, 64, 67, 69, 71, 73 ].choose;
      ~playGuitar.value(2.1, 0.01, ~guitarNote, 1.0, ~guitarAmp);
      ~picassoPartAppearFunc.value(rrand(0.001, 0.01), 2, false, false);
   });

   rrand(4, 18).do({
      ~playGuitar.value(10.5, 0.01, ([ 50, 57, 62, 64, 67, 69, 71, 73 ].choose)-12, 1.0, ~guitarAmp);
      rrand(0.125, 0.13).wait;
      ~picassoPartAppearFunc.value(rrand(0.001, 0.01), 2, false, false);
   });
}).play;
);

// harp
(
Task({

   rrand(4, 12).do({
      ~guitarNote = [ 50, 57, 62, 64, 67, 69, 71, 73 ].choose;
      ~playHarp.value(2.1, 0.01, ~guitarNote, 1.0, 0.4);
            ~picassoPartAppearFunc.value(rrand(0.001, 0.01), 1, false, false);
   });

   rrand(4, 18).do({
      ~playHarp.value(10.5, 0.01, ([ 50, 57, 62, 64, 67, 69, 71, 73 ].choose)-12, 1.0, 0.4);
      rrand(0.125, 0.13).wait;
            ~picassoPartAppearFunc.value(rrand(0.001, 0.01), 1, false, false);

   });
}).play;
)

// guitar and harp together
(
Task({

   rrand(8, 18).do({
      ~guitarNote = [ 50, 57, 62, 64, 67, 69, 71, 73 ].choose;
      ~playGuitar.value(2.1, 0.01, ~guitarNote, 1.0, ~guitarAmp);
   });

   rrand(8, 18).do({
      if ( 0.25.coin, {
         ~playGuitar.value(10.5, 0.01, ([ 50, 57, 62, 64, 67, 69, 71, 73 ].choose)-12, 1.0, ~guitarAmp);
      }, {
         ~playHarp.value(10.5, 0.01, ([ 50, 57, 62, 64, 67, 69, 71, 73 ].choose)-12, 1.0, 0.5);
      } );
      rrand(0.125, 0.13).wait;
   });
}).play;
)
```

Code listing 8: Picasso and guitars 'live coding'

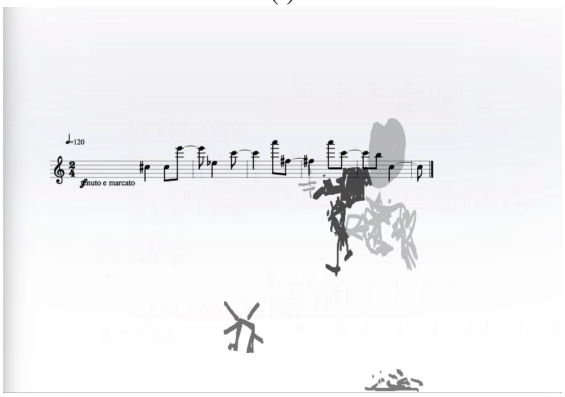
```
// then automated and strictly in time
(
~guitarTranspose = 0;
~guitarOctave = 0;
~picassoCol = [100, 255];

~autoGuitarTask = Task({
   var myWait = 0.125;

   ~autoGuitarTask.stop;
   ~strumTask.stop;

      ~strumTask = Task({ // the initial 'strum'
         rrand(4, 10).do({
            ~guitarNote = [ 50, 57, 62, 64, 67, 69, 71, 73 ].choose;
            ~playGuitar.value(2.1, 0.01, ~guitarNote+~guitarTranspose+~guitarOctave, 1.0, ~guitarAmp);
            if ( 0.9.coin, {
               ~picassoPartAppearFunc.value(rrand(0.01, 0.01), 16, false, false, ~picassoCol);
            }, { ~picassoPartAppearFunc.value(rrand(0.1, 1.2), 16, false, false, ~picassoCol);
            } );
         });

         rrand(16, 36).do({
            ~playGuitar.value(10.5, 0.01, (([ 50, 57, 62, 64, 67, 69, 71, 73 ].choose)-12)+~guitarTranspose+~guitarOctave, 1.0,
```

```
        ~guitarAmp);

                if ( 0.9.coin, {
                    ~picassoPartAppearFunc.value(rrand(0.01, 0.01),
        16, false, false, ~picassoCol);
                }, { ~picassoPartAppearFunc.value(rrand(0.1, 1.2),
        16, false, false, ~picassoCol);
                } );
                myWait.wait;
            });
        }).play;
        myWait.wait;
}).play;
)
```

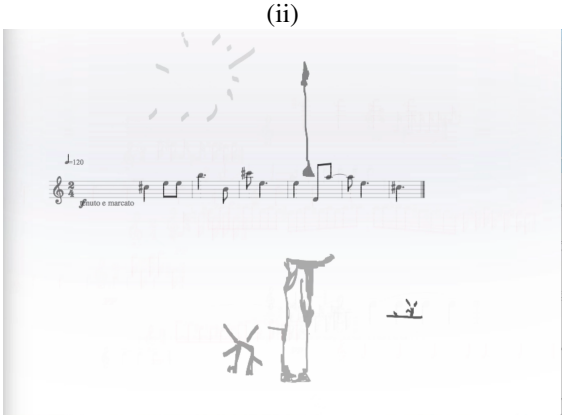
Code listing 9: The guitar texture algorithm



(i)



(ii)



(iii)

Figure 12: Three renditions of Picasso, guitars and melody.

The guitar-sound-based texture is then augmented by violin melodies, to be played by the live violinist, based on the original melody developed earlier in the piece, but now using the tonality of the rhythmic guitar texture as a tonal basis. These melodies gradually take over the texture, becoming more and more sustained as the guitar fades (from about 10:30, see **code listing 10**). The notes from which those played are chosen are provided by the array

```
~chordNotes
```

which can be coded live, or taken from a set of arrays (see code listing 10) based on the chords shown in figure 13. Harmonically, they comprise fourths and fifths with a more complex overtones and harmonics.

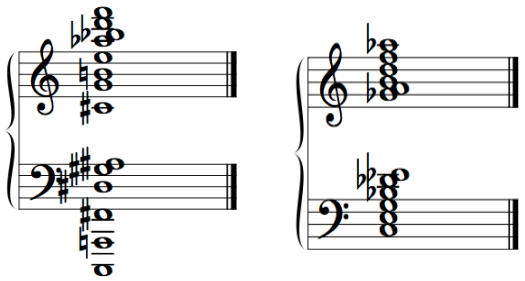


Figure 13: Two tonally ambiguous chords forming the basis of some of the harmonies of the 'sustained string' section of *Homenaje*

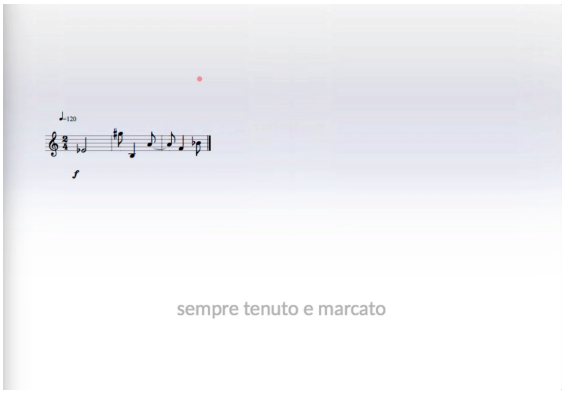
```

// chord Notes
~chordNotes = [ 50, 57, 62, 64, 67, 69 ]; // d, a, d, e, g, a
~chordNotes = [ 50, 57, 62, 64, 67, 69, 71 ]; // + b
~chordNotes = [ 50, 57, 62, 64, 67, 69, 71, 73 ]; // + c#

~chanceOfNote = 0.8;

~chordNotes = [ 50, 57 ]; // fifth
~chordNotes = [ 50, 57, 64 ]; // two fifths
~chordNotes = [ 50, 57, 64, 71 ]; // three fifths
~chordNotes = [ 50, 57, 64, 67, 71 ]; // three fifths

~chordNotes = [ 50, 57, 61, 64, 66, 68, 78 ];
~chordNotes = ~chordNotes + 8
~chordNotes = ~chordNotes - 8
~chordNotes = [ 43, 50, 57, 64, 71 ];
~chordNotes = [ 43, 50, 57, 61, 64, 66, 68, 78 ];

~chanceOfNote = 0.7;

~chordNotes = [ 37, 43, 50, 57, 66, 73, 80, 83, 87 ]; // d, a,
    f#, c#, g#, b, d#
~chordNotes = [ 50, 57, 66, 73, 80, 83 ];
~chordNotes = [ 50, 57, 66, 73, 80 ];

~chanceOfNote = 0.6;

~chordNotes = [ 50, 57, 66, 73 ]; // d, a, f#, c#
~chordNotes = [ 66, 74, 73 ];
~chordNotes = [ 73 ];
```

Code listing 10: Sustained string texture at end

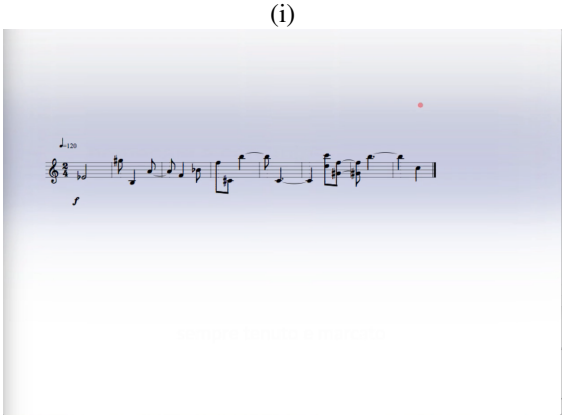
At this point the Picasso sketch is complete and its parts fade in and out, rotating, until they disappears altogether along with the violin music.

7. CONCLUSIONS

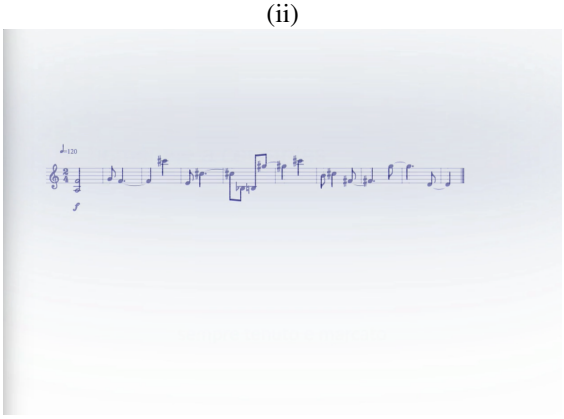
Homenaje is a composition that consciously extends the integration of text and image with music, a process that be-



(i)



(ii)



(iii)

Figure 6: Other renditions of the melody are shown here. Note that in (iii) the melody changes colour from black to dark blue when the fade out begins to indicate this to the instrumentalist.

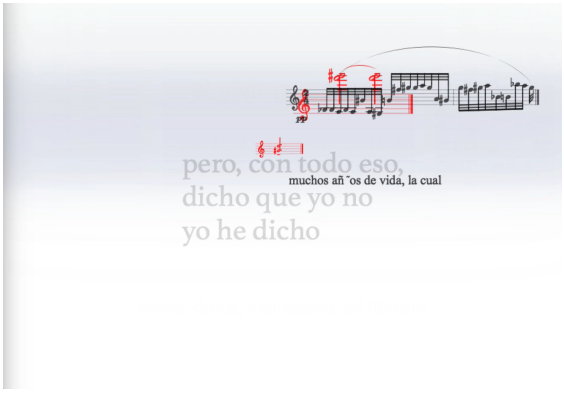


Figure 7: A rendition of text, lines and chords.

gan with the music-dance-text piece *Choreograms* in which 18th century dance notations appeared alongside contemporary poetry, audio and music notations. It formalises the structures put in place in earlier pieces, most notably the sets of object arrays that allow the storage of each objects state (e.g. location, colour, alpha value, rotation).

Each project spent composing with SuperCollider and INSCORE convinces me of the flexibility and power of these pieces of software, both individually and in combination. Each piece of software allows virtually complete control not only over the tools and mechanisms involved, they also allow maximum freedom over how they themselves can be controlled. Both SuperCollider and INSCORE can be used as standalone pieces of software, or as engines to be controlled from other preferred resources. The fact that **sclang** is able to deal so effectively with rather obscure and arbitrary text-based functionality such as converting text to all lower (or all upper, or any number of other string transformations) demonstrates both its flexibility and the importance of that flexibility in cross-domain work. There is little doubt that all of these factors make the investigation of links and mappings between diverse expressive domains particularly suitable for these resources and there is enormous potential for discovery and expression.

8. REFERENCES

[1] S. Wilson, D. Cottle, and N. Collins, Eds., *The SuperCollider Book*. Cambridge, MA: MIT Press, 2011.

[2] D. Fober, Y. Orlarey, and S. Letz, "Inscore – an environment for the design of live music scores," in *Proceedings of the Linux Audio Conference*, 2012, pp. 47–54.

[3] R. Hoadley, "Live coding, live notation, live performance," in *Electronic Visualisation and the Arts (EVA 2016)*, ser. Electronic Workshops in Computing (eWiC), J. P. Bowen, G. Diprose, and N. Lambert, Eds., British Computer Society. British Computer Society, 2016, pp. 34–41.

[4] N. Collins, A. McLean, J. Rohrhuber, and A. Ward, "Live coding in laptop performance," *Organised Sound*, vol. 8, no. 3, pp. 321–330, 2003.

[5] Google, "Google map," June 2017. [Online]. Available: https://goo.gl/maps/EmhqzZdmeA52

[6] R. Hoadley, "Calder's violin (performance)," in *Proceedings of the International Computer Music Conference*. Athens, Greece: ICMA, September 2014.

[7] ——, "How to play the piano," in *Proceedings of the ICMC*, H. Timmermans, Ed., ICMA. Utrecht, Netherlands: HKU University of the Arts Utrecht, 2016, pp. 176–180.

[8] A. G. L. Ré, "A possible source for picasso's drawing of don quixote," *Cervantes: Bulletin of the Cervantes Society of America*, vol. 12, no. 1, pp. 105–110, 1992.