To the Dark Unseen

For String Dectet

Nigel Morgan

Symbolic Composer score files, annotated by Phil Legard

To the Dark Unseen

For String Dectet

Nigel Morgan

This three-movement work for two solo violins and two string quartets (one tuned a whole tone lower) is the result of a very particular idea that in all probability could only have been realised using algorithmic means.

The composition collects together a number of separate ideas. The first, that there would be a single tonality created from the bringing together of open strings and second harmonics.

The second is that change of texture and mix of pitches would take place on each quarter beat – although for convenience sake the work is organised in a 2/4 metre. This change of texture / mix of pitches would be achieved algorithmically through the use of a magic square. The result of this process was to output a graphically 'map' of when instruments of the ensemble are playing or are silent.

The third idea, which developed a little later than the first two, was that two solo violins would engage above the ensemble in a conversation based on the gen-hopalong algorithm, a kind of 'strange attractor'.

As the movements progress the overall pitch of the music falls. The *ripieno* string quartets develop more complex sub-divisions of rhythm than found at the opening.

```
; Additional functions required

(defun instrument-to-string (instrument array-output)
  (do-quietly
    (symbol-to-string
     (e-substitute '(=) '(a)
        (mapcar 'integer-to-symbol
          (flatten (mapcar
           (function (lambda (x)
            (e-count instrument x))) array-output)))))))
```

Some additional functions are necessary to execute the code for *Into the Dark Unseen*. **Instrument-to-string** can be used to generate 'beat-space' timesheets. These are visualisations of when each instrument is playing, and are ultimately templates for orchestration. This function looks at a series of lists of instrumental action, for example `((a2 v3 v4 c2) (c2 d1 v4 v3 c1))`. If we used the above list and executed the function for the instrument 'a2' we would get the following beat-space list `"- "`. For the instrument 'c2' we would get `"--"`.

```
(defun string-to-length (length string)
 (let (out)
 (dovector (x string)
 (if (equal x #\Space)
 (push (* (abs (get-ratio length)) -1) out)
 (push length out)))
 (nreverse out)))
```

String-to-length can be used to map a series of durations to a beat-space string. So, `(string-to-length '1/8 "- - -")` yields `(1/8 -1/8 1/8 -1/8 1/8)`. Negative values equate to rests (spaces) in the beat-space string. This function is only used internally by **zone-convertor** function and is not called as part of the main code.

```
(defun zone-convertor (list-of-zones timesheet-string)
 "maps beat/space values onto a list of zones"
 (do-quietly
  (get-ratio-sc (mapcar (function (lambda (x y) (* (get-ratio-cl x) y)))
                (mapcar 'get-ratio list-of-zones)
                (string-to-length '1 timesheet-string)))))
```

Zone-convertor is used to map a series of zone-lengths (somewhat analogous to bars) into areas of musical activity and rest. The functionality is very similar to **string-to-length**, the difference being that there are multiple lengths passing through the function. So `(zone-convertor '(1/4 1/2 1/8) "- -")`would yield `(1/4 -1/2 1/8)`.

```
(defparameter *flat-them-stack* nil)

(defun flat-them-sup (l)
  (cond ((null l) nil)
        ((is-flat l)
         (push l *flat-them-stack*))
        (t
         (flat-them-sup (car l))
         (flat-them-sup (cdr l)))))

(defun flat-them (l)
       (setq *flat-them-stack* nil)
       (flat-them-sup l)
       (nreverse *flat-them-stack*))
```

Flat-them will remove any 'lists-within-lists' that have cropped up during the coding process.

So, `(flat-them '((a b) ((c d)) (((e f g)))))` yields `((a b) (c d) (e f g))`.

```
(defun gen-process-list (f-expr values patterns)
"processes a list of lists with a list of differing values
- gen-process only allows a single list to be processed "
(diagnostic2 "gen-process-list" $cr$)
(setq f-expr (eval (list 'function
             (append '(lambda) (list '(x y) f-expr)))))
             (prog (out)
             (let* ((initial diagnose-verbose)
               (diagnose-verbose nil))
             (setq out (mapcar f-expr values patterns ))
             (setq diagnose-verbose initial))
          (return out)))
```

Gen-process-list is a flexible function that allows a function to be applied to two 'lists of lists' with differing values. For example: `(gen-process-list '(change-length :divide x y :ratio) '(1 2 4) '((1/4) (1/4) (1/4)))` yields `((1/4) (1/8) (1/16))`.

```
(defun zone-expand (x-by zne-lis) ; adjusted 15.3.04
"expanding values of chosen zone-lengths to create repeats"
   (prog ( out)
     loop
     (cond ((null zne-lis) (return (get-ratio-sc out))))
     (setq out (append out (list (* (car x-by) (get-ratio-cl (car zne-lis))))))
     (setq x-by (cdr x-by))
     (setq zne-lis (cdr  zne-lis))
     (go loop)))
```

Zone-expand can be used to create repeats by multiplying the length of a given zone, so `(zone-expand '(1 2) '(1/4 1/4))` yields `(1/4 1/2)`.

```
; To the Dark Unseen  - string dectet
; Part 1

#|
 v1 - violins
 v2
 v3
 v4
 a1- violas
 a2
 c1 - cellos
 c2
 d1 - bass
|#
(setq moments
(build-array
 ; column 0    1     2    3    4   5    6
      '(( c1   a1    c2   a2   v3  d1   a2)     ; 0
        (v3   v4    a1   a2   c1  c2   d1)     ; 1
        (v4   a1    c1   a2   c2  v3   v4)     ; 2
        (a2   a1    c2   v4   v3  d1   c1)     ; 3
        (a1   v3    v4   a2   d1  c2   a1)     ; 4
        (c1   c2    v3   d1   v4  a2   v3)     ; 5
        (d1   a1    a2   c2   v3  v4   c2)     ; 6
)))

(setq lisn '(0 1 2 3 4 5 6)
)


(setq output-all
     (mapcar 'remove-duplicates
             (delete 'nil
               (gen-collect 0.2 57 :list
               '(pick-array
                    (pick1 nil lisn)
                    (pick1 nil lisn) (get-random 2 6) moments
                                     (pick1 nil '(:left
                                                  :right
                                                  :up
                                                  :down
                                                  )))))))
```

This array is used to define instrumental activity in each zone (or bar) of the piece. The table can be read up, down, left or right. For a composition using similar techniques to this one, see the annotations for *String Trio (2012)*.

Here, a series of orchestrations are chosen from the array above. This code is executed 57 times by **gen-collect** with a random seed of 0.2. A row and column is picked from the array using a value from **lisn** (0 to 6). A number of items between (2 and 6) are picked from the selected cell in the array, moving either left, right, up or down from that cell. So, `(pick-array 0 0 4 moments :down)` would start from the top left corner of the **moments** array (0,0). Starting with the value at 0,0, we would pick four values, moving down one cell at a time. The picker wraps back around to the opposing value if the number of picks exceeds the table length., yielding `(c1 v3 v4 a2)`.

Finally, any duplicates in each pick are removed, so `(a1 a1 v3 c2)` becomes `(a1 v3 c2)`.

```
; output of all 57 picks:
#|
((a2 v3 v4 c2) (c2 d1 v4 v3 c1) (v4 v3 c1) (a2 d1 c2 a1 v3) (v3 v4 c2 a1 a2) (a2 c2 v3 a1 c1 v4 d1)
(a2 c2 v3 v4 a1) (c2 v3 a1 c1 v4) (v4 a1 a2 c1 c2) (v3 d1 c1) (v4 v3 c1) (c2 v3 d1 a2 c1 a1)
(v3 c2 a2 d1 v4 c1 a1) (c1 a2 d1) (c2 d1 a2 v4 v3 a1) (v4 c2 a1 a2 c1 d1) (d1 v3 a2 c2 a1 c1)
(c2 d1 v3 v4) (v4 a2) (c1 c2 v3 d1) (c2 c1 v3) (c2 c1 v3 a2) (v4 a1 c1 a2 c2 v3) (c2 a1 c1 a2 d1)
(a2 v4 d1) (c1 a1 v4 v3 c2 a2) (v4 c2 d1 a2) (d1 c1 v3 v4) (c2 a1 v3 v4 a2 d1) (v4 a2 c2) (v3 c2 a1 v4)
(v3 d1 a2 c1 a1) (c2 v3 d1 v4 a2) (v4 v3 a1 c2) (v3 v4 a2 c2 d1) (v4 d1 v3 c2 c1) (v4 c2 v3 a1) (c2 v3 a1)
(d1 v4 a2) (v3 c2 d1 v4) (a2 v3 v4 c2) (d1 a2 c1 a1 c2) (v3 d1 c1 a1 a2) (v4 v3 c2 a2) (d1 c2 v4 a2) (v3 d1 a2)
(v4 a1) (a2 v4 d1 v3 c2 c1) (v4 v3 d1 c2 c1 a2) (v4 d1 c2 a2) (v4 v3 a2) (d1 c1 v3 v4 a2 a1) (v4 a2 v3 c1)
(a2 a1 v4) (a1 c2 a2 v3) (a2 v4 d1 v3 c2) (v4 d1 c2 a2))

|#

;; instrumentation per section

(instrument-to-string 'v3 output-all)

(instrument-to-string 'v4 output-all)

(instrument-to-string 'a1 output-all)

(instrument-to-string 'a2 output-all)

(instrument-to-string 'c1 output-all)

(instrument-to-string 'c2 output-all)

(instrument-to-string 'd1 output-all)

#| ;; one minute of the timesheet for ensemble

v1  "   -------   - - - -   - ---- - - - - ---- ----  - - -  -"
v2  "   - - - -   -------  - - --- - - - - ---- ----  - - -  -"
v3  "--- ---- ---- - -- ---   - -- -------- -- -- - -- ---  - "
v4  "--- --- - - - --  -   - --- --  ----- - -  -- -------  --"
a1  "   ------  -- ---     -- -  - -- -  --   --   -   - --  "
a2  "-  - -- -  ------ -   --- - -- -  -   - --  -- - - ------"
c1  "     - --  --- -   ----- - -   -         --     -  -    "
c2  "-- -----   -- ---- ----- -- - - - ---- -  --  ---    - -"
d1  " - -     - -- - --        --   -- -- -- -- -- --- -   --"
|#
```

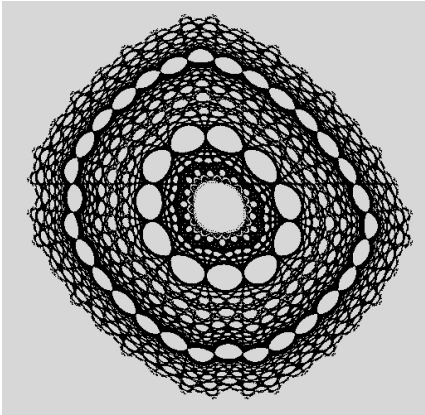
The orchestration has been saved into a series of lists called **output-all** – you can see the content of this list above. **Instrument-to-string** is used to create timesheets for each instrumental, which can be used as a template to control areas of activity and rest.

```
(create-tonality st-1 '(g 3 d 4 g 4 a 4 d 5 e 5 a 5 e 6))
 ;; viola and cello tonalities

(create-tonality st-2 '(e 2 g 2 d 3 e 3 g 3 a 3 d 4 g 4))
;; double bass

(create-tonality st-2i '(e 2 g 2 a 2 c 3 d 3 e 3 f 3 g 3 a 3 d 4))
;; double bass - extended across 2 tonalities

(create-tonality s-vn '(g 3 a 3 b& 3 c 4 d 4 e 4 f 4))
;; violin
```

A series of unique tonalities are defined for each instrument – note the two tonalities for the double bass.

```
(gen-hopalong-symbol x (a m) y (a m) 100 200 300 0.45 120 0)

(setq mel-1 x
      mel-2 y)
```

Gen-hopalong-symbol is an algorithm which creates two streams of related symbols (**x** and **y**). Here both streams of symbols have values between **a** and **m**. The output forms the melody for violins 1 and 2. The Hopalong algorithm is a model of 'strange attractor', and typically produces patterns such as this:



Source: Pythonism

```
(setq st1-mlis '(c d e f g h)
      st2-mlis '(c d e f g h i j)
      st3-mlis '(c d e f)
      st1-clis '(ce de eg gh)) ; not bass

;; pitch symbol ordering

(setq st-lisx (append st1-mlis st1-mlis st1-clis)
      st-lisy (append st1-mlis st1-mlis )
      st-lis3 (append st3-mlis st3-mlis )
      st-lisz st2-mlis
      st1-sh1 (gen-random 0.2 60 (symbol-shuffle st-lisy)) ; vn3
      st1-sh2 (gen-random 0.1 60 (symbol-shuffle st-lisy)) ; vn4
      st1-sh3 (gen-random 0.2 60 (symbol-shuffle st-lis3)) ; va1
      st1-sh4 (gen-random 0.21 60 (symbol-shuffle st-lis3)) ; va2
      st1-sh5 (gen-random 0.32 60 (symbol-shuffle st-lis3)) ; vc1
      st1-sh6 (gen-random 0.46 60 (symbol-shuffle st-lis3))  ; vc2
      st1-sh7 (gen-random 0.56 60 (symbol-shuffle st-lisy))) ; db
```

Melodic and chordal material is defined here for the other instruments. A series of lists are created containing 60 random picks from this material. Note in the first movement the chordal material is not used, only values from **st-lisy** and **st-lis3**. A sample pick might look like this: `(f c c f f e f f g g h c c h . . .`

```
(setq zne-1 (gen-random 0.1 57 '(1/4 1/4 1/4 1/2)))


;; zone-lengths

(setq zn-1 (zone-convertor zne-1 (instrument-to-string 'v3 output-all))  ;vn3
      zn-2 (zone-convertor zne-1 (instrument-to-string 'v4 output-all))  ;vn4
      zn-3 (zone-convertor zne-1 (instrument-to-string 'a1 output-all))  ;va1
      zn-4 (zone-convertor zne-1 (instrument-to-string 'a2 output-all))  ;va2
      zn-5 (zone-convertor zne-1 (instrument-to-string 'c1 output-all))  ;vc1
      zn-6 (zone-convertor zne-1 (instrument-to-string 'c2 output-all))  ;vc2
      zn-7 (zone-convertor zne-1 (instrument-to-string 'd1 output-all))) ;db

(setq zs-1 (zone-convertor zne-1 "   -------   - - - - --- ---- - - - - ---- ----  - ---- -")  ;vn1
      zs-2 (zone-convertor zne-1 "   - - - -   ------- -- ---- - - - - ---- ----  - - -  -")) ;vn2


;; lengths diminution

(setq len-x1  (gen-process-list '(change-length :divide x y :ratio)
                                (gen-random 0.21 120 '(1 1 1 1 2 3 4 4)) (mapcar 'list (l-rest-revert zn-1))))

(setq mel-1x (symbol-divide (gen-random 0.21 120 '(1 1 1 1 2 3 4 4)) nil nil mel-1)
      mel-2x  (symbol-divide (gen-random 0.31 120'(1 1 1 1 2 3 4 4)) nil nil mel-2))

(setq len-x2  (gen-process-list '(change-length :divide x y :ratio)
                                (gen-random 0.31 120 '(1 1 1 1 2 3 4 4)) (mapcar 'list (l-rest-revert zn-1))))

(setq len-x3  (gen-process-list '(change-length :divide x y :ratio)
                                (gen-random 0.21 60 '(1 1 1 1 2 3 4 4)) (mapcar 'list (l-rest-revert zn-1))))

(setq len-x4  (gen-process-list '(change-length :divide x y :ratio )
                                (gen-random 0.31 60 '(1 1 1 2 2 4 4 4)) (mapcar 'list (l-rest-revert zn-1))))
```

A series of zones of 1/4 to 1/2 duration are created. By default each instrument has a long series of symbols which could be theoretically played. They are silenced by turning the corresponding zone values into negative numbers. **Instrument-to-string** generates a template which is used to silence zones with zone-convertor. The first two violins play to a pre-composed pair of beat-space strings.

To create differing rhythms in each part the zone-length is divided by a number between 1 and 4. A value of 3, for example, would create triplet rhythms (dividing the entire zone into three equal parts). In the lists mel-1x and mel-2x, used for violins 1 and 2, the same random seed is used with symbol divide to ensure that the symbol stream is divided up into an appropriate number of notes for each division. So, the random series `(3 1 1 4)` would yield `((1/12) (1/4) (1/4) (1/16))` in length and `((k g e) (l) (h) (c m j a))` in symbols/pitches.

```
;; score

(def-tonality
  vn1 (activate-tonality  (s-vn g 5))
  vn2 (activate-tonality  (s-vn g 5))
  vn3 (activate-tonality  (st-1 g 4))
  vn4 (activate-tonality  (st-1 f 4))
  va1 (activate-tonality  (st-1 c 4))
  va2 (activate-tonality  (st-1 b& 3))
  vc1 (activate-tonality  (st-1 c 3))
  vc2 (activate-tonality  (st-1 b& 2))
  db  (activate-tonality  (st-2i e 2))
)
```

Assign tonalities to each instrument – note that only the first double bass tonality is used in this section.

```
(def-symbol
  vn1  (flat-them (mapcar 'list (symbol-swallow zs-1 mel-1x)))
  vn2  (flat-them (mapcar 'list (symbol-swallow zs-2 mel-2x)))
  vn3  (mapcar 'symbol-melodize (mapcar 'list (symbol-swallow zn-1 st1-sh1))) ; arpeggios
  vn4  (mapcar 'list (symbol-swallow zn-2 st1-sh2)) ; chords
  va1  (mapcar 'symbol-melodize (mapcar 'list (symbol-swallow zn-3 st1-sh3)))
  va2  (mapcar 'list (symbol-swallow zn-4 st1-sh4))
  vc1  (mapcar 'symbol-melodize (mapcar 'list (symbol-swallow zn-5 st1-sh5)))
  vc2  (mapcar 'list (symbol-swallow zn-6 st1-sh6))
  db   (symbol-swallow zn-7 st1-sh7)
)
```

The necessity of **flat-them** becomes apparent in the violin parts. Here a list of zones is used to tell when each pitch (themselves a list-of-lists) is silent. **Symbol-swallow** works well with lists of lengths and symbols, for example: `(symbol-swallow '(-1/4 1/4 -1/4) '(a b c))` yields `(= b =)`. However, we are dealing with zones and nested lists, which symbol-swallow does not treat, so if the symbolic data in the previous example was `'((a) (b) (c))`, the output would be `(= (b) =)`. Calling **mapcar 'list** helps in most situations and would yield the correct ((=)(b)(=)). However, as a consequence of the pitch generation process the voices may have data that contains extraneous brackets such as `((a)(b)((c d e f))(g))`, which **flat-them** will remove: `((a) (b)(c d e f)(g))`.

```
(def-length
  vn1 len-x1
  vn2 len-x2
  vn3 (l-rest-revert zn-1)
  vn4 (l-rest-revert zn-2)
  va1 (l-rest-revert zn-3)
  va2 (l-rest-revert zn-4)
  vc1 (l-rest-revert zn-5)
  vc2 (l-rest-revert zn-6)
  db  (l-rest-revert zn-7)
)
```

```
(def-velocity
  vn1 '(36)
  vn2 '(36)
  vn3 '(34)
  vn4 '(34)
  va1 '(34)
  va2 '(34)
  vc1 '(38)
  vc2 '(38)
  db '(34)
)

(def-zone
  tempo (but-last (append (l-rest-revert zn-1)'(1/2 1/2 1/2)))
  default (zone-expand (append  (build-list '1 56) '(3)) (l-rest-revert zn-1))
)

(def-tempo (append (build-list '50 56) '(47 45 40)))

(def-channel
  vn1 1
  vn2 2
  vn3 3
  vn4 4
  va1 5
  va2 6
  vc1 7
  vc2 8
  db  9
)

(compile-instrument-p "ccl;output:" "decet-1Final"
  vn1
  vn3
  va1
  vc1
  vn2
  vn4
  va2
  vc2
  db
)
```

Note that the order of compilation is different, essentially dividing the instruments into two choirs (vn1, vn3, va1 & vc1 against vn2, cn4, va2 and vc2) and double bass (played in unison between two players in the final dectet).

```
; To the Dark Unseen  - string dectet (2)

(setq moments
(build-array
 ; column 0    1     2    3    4    5    6
      '(( c1   a1    c2   a2   v3   d1   a2)     ; 0
        (v3  v4     a1   a2   c1   c2   d1)     ; 1
        (v4   a1    c1   a2   c2   v3   v4)     ; 2
        (a2   a1    c2   v4   v3   d1   c1)     ; 3
        (a1   v3    v4   a2   d1   c2   a1)     ; 4
        (c1   c2    v3   d1   v4   a2   v3)     ; 5
        (d1   a1    a2   c2   v3   v4   c2)     ; 6
)))

(setq lisn '(0 1 2 3 4 5 6))


(setq output-all
     (mapcar 'remove-duplicates
             (delete 'nil
                     (gen-collect 0.21 60 :list
                             '(pick-array
                                    (pick1 nil lisn)
                                    (pick1 nil lisn) (get-random 2 6) moments
                                    (pick1 nil '(:left
                                                 :right
                                                 :up
                                                 :down
                                                 )))))))

; output of output/all/m
#|
((v4 a2 d1 c2) (a2 c2 v3 a1) (a1 a2 v4 v3 c1) (a2 c1 a1 c2) (a2 a1 d1 c1 v3) (v3 v4 a1) (d1 v3 c2)
(d1 c2 a2 v4) (v3 a1) (v3 c2 v4 a1) (d1 a1 a2 c2) (a2 d1 c2) (d1 c1 v3) (d1 v4 v3) (c2 v4 v3 d1)
(c1 c2 v3) (a1 a2) (a1 c2 v4) (a2 v4 d1 c2 v3) (c2 c1) (c2 v3 d1) (c2 v3 d1) (c2 c1 a2 a1) (v3 v4 a1)
(d1 v4 v3 c2) (a2 c2 d1) (c1 v3) (v4 v3 c2) (c2 a1 c1) (a1 c2 v3) (a2 c2 v3 v4) (d1 a1 a2) (v3 c1) (v4 a1)
(v3 d1 c2 c1 a2) (v4 d1 c2) (a2 c2 v3 a1) (c1 a1 c2 a2) (d1 c1 v3 v4 a2) (d1 a1 a2 c2) (v4 v3 a2) (v3 a1)
 (v4 c2) (c2 d1) (c2 a1) (a2 d1 c2 a1) (c1 v4 v3 a2 c2) (v4 a1 c1) (a2) (v4 d1 v3) (v4 d1 v3) (v4 d1 c2 v3)
 (d1 c2 a2) (v4 d1 v3 c2) (a1 c2) (c1 a1 v3) (a1 v3 v4 a2 d1) (c1 c2 v4 v3 a2) (a2 c2) (d1 v3 v4 c2))
|#
```

The second and third movements are broadly the same in structure, but with some key differences. Here fore example, the random seed is 0.21, rather than 0.2. This may seem a small change, but it yields an entirely different set of orchestrations.

```
;; instrumentation per section

(instrument-to-string 'v3 output-all)

(instrument-to-string 'v4 output-all)

(instrument-to-string 'a1 output-all)

(instrument-to-string 'a2 output-all)

(instrument-to-string 'c1 output-all)

(instrument-to-string 'c2 output-all)

(instrument-to-string 'd1 output-all)


#| ;; one minute of the timesheet for ensemble

v1  "        - - -    -----  ----  ---   - ----  -----    -  - --"
v2  "       -----       -- -- -- -- --  - - ---     ----- - -----"
v3  " -- --- --  ----  - -- -- -- -- - - - - --    -  --- - --- -"
v4  "- -  - - -   --  --    --  -  -  - -  - - -   -- --- -  -- -"
a1  " -----  ---     --    --    -- - -  -- - -  -- -      ---   "
a2  "-----  -  --    - -   -  -    --  - -----    -- -   -   --- "
c1  "  ---       -  -   -  -   - -   - -  --       --       - -  "
c2  "-- -  -- ---  -- ------ -- ----   ---- -  -----    ----  ---"
d1  "-   - --  -----   - --  --     -  --  --   - -   -----  -  -"
|#

(create-tonality st-1 '(g 3 d 4 g 4 a 4 d 5 e 5 a 5 e 6))
 ;; violin and viola tonalities

(create-tonality st-2 '(e 2 g 2 d 3 e 3 g 3 a 3 d 4 g 4))
;; double bass

(create-tonality st-2i '(e 2 g 2 a 2 c 3 d 3 e 3 f 3 g 3 a 3 d 4))
;; double bass - extended across 2 tonalities

(create-tonality s-vn '(g 3 a 3 b& 3 c 4 d 4 e 4 f 4))

(gen-hopalong-symbol x (-h h) y (-h h) 100 200 300 0.51 300 0)
```

To the Dark Unseen is based on the idea of gradual descent. Whereas the ranges of the hopalong algorithm in the first section ran from **a** to **m**, they now go from **–h** to **h**. This means that the lower end of the range has increased by a fifth (**-h** to **a**), while the upper end has attenuated by a fourth (**h** to **m**).

```
(setq mel-1 x
      mel-2 y)

(setq st1-mlis '(a b c d e f g h) ;
      st2-mlis '(a b c d e f g h i j) ;
      st1-clis  '(ab bd ce de eg gh)) ;

;; pitch symbol ordering

(setq st-lisx (append st1-mlis st1-mlis st1-clis)
      st-lisy (append st1-mlis st1-mlis )mel-1x
      st-lisz  st2-mlis
      st1-sh1 (gen-random 0.2 60 (symbol-shuffle st-lisx))
      st1-sh2  (gen-random 0.1 60 (symbol-shuffle st-lisx))
      st1-sh3  (gen-random 0.2 60 (symbol-shuffle st-lisx))
      st1-sh4  (gen-random 0.21 60 (symbol-shuffle st-lisx))
      st1-sh5 (gen-random 0.32 60 (symbol-shuffle st-lisx))
      st1-sh6  (gen-random 0.46 60 (symbol-shuffle st-lisx))
      st1-sh7 (gen-random 0.56 60 (symbol-shuffle st-lisy)))

(setq zne-1 (gen-random 0.1 60 '(1/4 1/4 1/4 1/2)))

;; zone-lengths

(setq zn-1 (zone-convertor zne-1 (instrument-to-string 'v3 output-all))
      zn-2 (zone-convertor zne-1 (instrument-to-string 'v4 output-all))
      zn-3 (zone-convertor zne-1 (instrument-to-string 'a1 output-all))
      zn-4 (zone-convertor zne-1 (instrument-to-string 'a2 output-all))
      zn-5 (zone-convertor zne-1 (instrument-to-string 'c1 output-all))
      zn-6 (zone-convertor zne-1 (instrument-to-string 'c2 output-all))
      zn-7 (zone-convertor zne-1 (instrument-to-string 'd1 output-all)))

(setq zs-1 (zone-convertor zne-1 "        - - -    -----  ----  ---   - ----  -----    -  - --")
      zs-2 (zone-convertor zne-1 "       -----       -- -- -- -- --  - - ---     ----- - -----"))

;; lengths diminution

(setq len-x1  (gen-process-list '(change-length :divide x y :ratio)
                                (gen-random 0.21 60 '(2 2 2 2 4 4 6 8 8)) (mapcar 'list (l-rest-revert zn-1))))

(setq mel-1x (symbol-divide (gen-random 0.21 60 '(2 2 2 2 4 4 6 8 8)) nil nil mel-1)
      mel-2x  (symbol-divide (gen-random 0.31 60 '(2 2 2 2 4 4 6 8 8)) nil nil mel-2))
```

Note that pitches in the lower end of the instrumental range is now available through the addition of symbols **a** and **b** to the melodic and chordal material.

```
(setq len-x2  (gen-process-list '(change-length :divide x y :ratio)
                                (gen-random 0.31 60 '(2 2 2 2 4 4 8 8)) (mapcar 'list (l-rest-revert zn-1))))

(setq len-x3  (gen-process-list '(change-length :divide x y :ratio)
                                (gen-random 0.21 60 '(1 1 1 1 2 3 4 4)) (mapcar 'list (l-rest-revert zn-1)))mel-1x

(setq len-x4  (gen-process-list '(change-length :divide x y :ratio)
                                (gen-random 0.31 60 '(1 1 1 2 3 4 4 4)) (mapcar 'list (l-rest-revert zn-1))))

;; score-

(def-tonality
  vn1 (activate-tonality  (s-vn g 5))
  vn2 (activate-tonality  (s-vn g 5))
  vn3 (activate-tonality  (st-1 g 4))
  vn4 (activate-tonality  (st-1 f 4))
  va1 (activate-tonality  (st-1 c 4))
  va2 (activate-tonality  (st-1 b& 3))
  vc1 (activate-tonality  (st-1 c 3))
  vc2 (activate-tonality  (st-1 b& 2))
  db  (activate-tonality  (st-2i e 2))
)

(def-symbol
  vn1  (flat-them (mapcar 'list (symbol-swallow zs-1 mel-1x)))
  vn2  (flat-them (mapcar 'list (symbol-swallow zs-2 mel-2x)))
  vn3  (mapcar 'symbol-melodize (mapcar 'list (symbol-swallow zn-1 st1-sh1))) ; arpeggios
  vn4  (mapcar 'list (symbol-swallow zn-2 st1-sh2)) ; chords
  va1  (mapcar 'symbol-melodize (mapcar 'list (symbol-swallow zn-3 st1-sh3)))
  va2  (mapcar 'list (symbol-swallow zn-4 st1-sh4))
  vc1  (mapcar 'symbol-melodize (mapcar 'list (symbol-swallow zn-5 st1-sh5)))
  vc2 (mapcar 'list (symbol-swallow zn-6 st1-sh6))
  db  (symbol-swallow zn-7 st1-sh7)
)
```

```
(def-length
  vn1 len-x1
  vn2 len-x2
  vn3 len-x3
  vn4 len-x4
  va1 len-x3
  va2 len-x4
  vc1 len-x3
  vc2 len-x4
  db  (l-rest-revert zn-7)
)

(def-velocity
  vn1 '(32)
  vn2 '(32)
  vn3 '(34)
  vn4 '(34)
  va1 '(34)
  va2 '(34)
  vc1 '(38)
  vc2 '(38)
  db  '(34)
)

(def-expression
  vn1  ((legato 99 30 0.4)(humanize -2 2 0.5) (velocity 20 0.7))
  vn2  ((legato 99 30 0.4)(humanize -2 2 0.5) (velocity 20 0.7))
)

(def-zone
  tempo (but-last (append (l-rest-revert zn-1)'(1/4 1/4 1/4)))
  default (zone-expand (append  (build-list '1 59)'(3))  (l-rest-revert zn-1))
)

(def-tempo (append (build-list '50 59) '(47 45 40)))




(def-channel
  vn1 1
```

```
  vn2 2
  vn3 3
  vn4 4
  va1 5
  va2 6
  vc1 7
  vc2 8
  db  9
)

(compile-instrument-p "ccl;output:" "decet-2"
  vn1
  vn3
  va1
  vc1
  vn2
  vn4
  va2
  vc2
  db
)
```

```
; To the Dark Unseen  - string dectet (3)

(setq moments
(build-array
 ; column 0    1     2    3    4    5    6
      '(( c1   a1    c2   a2   v3   d1  a2)     ; 0
         (v3  v4     a1   a2   c1   c2  d1)     ; 1
         (v4   a1    c1   a2   c2   d1  v4)     ; 2
         (a2   a1    c2   v4    v3  d1  c1)     ; 3
         (a1   c1    v4   a2   d1  c2  a1)     ; 4
         (c1   c2    v3  d1   v4   a2  v3)     ; 5
         (d1   a1    a2   c2   v3  v4   c1)     ; 6
)))

(setq lisn '(0 1 2 3 4 5 6))


(setq output-all
     (mapcar 'remove-duplicates
              (delete 'nil
                      (gen-collect 0.24 60  :list
                              '(pick-array
                                    (pick1 nil lisn)
                                    (pick1 nil lisn) (get-random 3 5) moments
                                    (pick1 nil '(:left
                                                 :right
                                                 :up
                                                 :down
                                                 )))))))

; output of output/all/m
#|
((v3 a2 c2 a1) (v4 c1 a1) (d1 v3 c2) (c1 v3 v4 a2 a1) (c2 a2 v4 d1) (d1 v4 v3) (c2 v4 v3 d1) (d1 v4 v3)
(a1 c1 c2) (v4 a1 c1) (c1 a1 v4) (d1 c2 c1 a2) (c2 v3 v4 c1 d1) (c2 a2 c1) (v3 a2 c2 a1) (c1 v3 a2 v4 d1)
(c1 d1 v3 v4 c2) (c1 v3 a1) (d1 c2 a2) (v3 c1 a2) (v3 c1 a2 d1) (a2 a1 c1) (a1 c2) (v4 c1 a1) (a2 c2 a1)
(v4 a1) (a2 v4 c2 d1) (a1 c1 a2 d1) (v3 d1 v4 a2) (c2 d1) (v4 a1) (v3 a2 c2) (c2 v3 d1 v4) (c2 c1 v3)
(c2 d1 a2 v4) (c1 a2 a1 c2) (v4 v3 c2 a2) (v4 a2 c2 d1) (c1 a2 a1 v4) (a2 d1 c2) (a1 c1) (a2 d1 c2 a1)
(d1 c2 a2) (v3 a2 c2) (c2 d1 v3) (c1 c2 v4) (a1 c2 d1) (d1 c1) (d1 v4 v3 c1) (v4 a2) (a1 a2 v4) (v4 a1 c1)
(a2 a1 c1 c2) (v4 a2 c2 d1) (a2 v4 d1 c2) (v4 d1) (a1 c2 a2) (d1 c2 a2) (v4 d1 c2 a2 c1) (a2 d1 v4))
|#
```

```
;; instrumentation per section

(instrument-to-string 'v3 output-all)

(instrument-to-string 'v4 output-all)

(instrument-to-string 'a1 output-all)

(instrument-to-string 'a2 output-all)

(instrument-to-string 'c1 output-all)

(instrument-to-string 'c2 output-all)

(instrument-to-string 'd1 output-all)


#| ;; one minute of the timesheet for ensemble

v1  " ---    ---- ----    - --   ---  ---     - --  --  -  -- - -"
v2  "   ----    ---        ---   -  --- --   - -  --  -- ---- - -"
v3  "- -- ---    - ----  --       -  ---  -      --   -          "
v4  " - ----- -- -  --      - -- - - - - ---      -  ---- ---  --"
a1  "-- -    ---   -  -   ----- -  -    -  - --    -   ---   -   "
a2  "-  --      - ---  ----  - ---  -  ------ ---     -- --- ----"
c1  " - -    ------ --- --- -   -     - -  - -    - --  --     - "
c2  "- - - - -  ---- - -   - - -  - ------- - ------     --- --- "
d1  "  - ----   --  -- - -     ----  - -  - - -- - ---    --- ---"
|#

(create-tonality st-1 '(g 3 d 4 g 4 a 4 d 5 e 5 a 5 e 6))
 ;; violin and viola tonalities

(create-tonality st-2 '(e 2 g 2 d 3 e 3 g 3 a 3 d 4 g 4))
;; double bass

(create-tonality st-2i '(e 2 g 2 a 2 c 3 d 3 e 3 f 3 g 3 a 3 d 4))
;; double bass - extended across 2 tonalities

(create-tonality s-vn '(g 3 a 3 b& 3 c 4 d 4 e 4 f 4))
```

```
(gen-hopalong-symbol x (-h e) y(-h e) 100 200 300 0.51 300 0)

(setq mel-1 x
      mel-2 y)

(setq st1-mlis '(a b c d e f) ; g h) ;
      st2-mlis '(a b c d e f g h i j) ;
      st1-clis  '(ab bd ce de)) ; eg gh)) ;

;; pitch symbol ordering

(setq st-lisx (append st1-mlis st1-mlis st1-clis)
      st-lisy (append st1-mlis st1-mlis )
      st-lisz  st2-mlis
      st1-sh1 (gen-random 0.2 60 (symbol-shuffle st-lisx))
      st1-sh2  (gen-random 0.1 60 (symbol-shuffle st-lisx))
      st1-sh3  (gen-random 0.2 60 (symbol-shuffle st-lisx))
      st1-sh4  (gen-random 0.21 60 (symbol-shuffle st-lisx))
      st1-sh5 (gen-random 0.32 60 (symbol-shuffle st-lisx))
      st1-sh6  (gen-random 0.46 60 (symbol-shuffle st-lisx))
      st1-sh7 (gen-random 0.56 60 (symbol-shuffle st-lisy)))

(setq zne-1 (gen-random 0.1 60 '(1/4 1/4 1/4 1/2)))

;; zone-lengths

(setq zn-1 (zone-convertor zne-1 (instrument-to-string 'v3 output-all))
      zn-2 (zone-convertor zne-1 (instrument-to-string 'v4 output-all))
      zn-3 (zone-convertor zne-1 (instrument-to-string 'a1 output-all))
      zn-4 (zone-convertor zne-1 (instrument-to-string 'a2 output-all))
      zn-5 (zone-convertor zne-1 (instrument-to-string 'c1 output-all))
      zn-6 (zone-convertor zne-1 (instrument-to-string 'c2 output-all))
      zn-7 (zone-convertor zne-1 (instrument-to-string 'd1 output-all)))

(setq zs-1 (zone-convertor zne-1 " ---    ---- ---     - --   ---  ---     - --  --  -  -- - -")
      zs-2 (zone-convertor zne-1 "   ----    ---        ---   -  --- --   - -  --  -- ---- - -"))

;; lengths diminution

(setq len-x1  (gen-process-list '(change-length :divide x y :ratio)
                                (gen-random 0.21 60 '(2 2 2 2 3 3 4 4 6)) (mapcar 'list (l-rest-revert zn-1))))
```

```
(setq mel-1x (symbol-divide (gen-random 0.21 60 '(1 2 2 2 3 3 4 4 6)) nil nil mel-1)
      mel-2x  (symbol-divide (gen-random 0.31 60 '(1 2 2 2 3 3 4 4 6)) nil nil mel-2))

(setq len-x2  (gen-process-list '(change-length :divide x y :ratio)
                                (gen-random 0.31 60 '(2 2 2 2 3 3 4 4 6)) (mapcar 'list (l-rest-revert zn-1))))

(setq len-x3  (gen-process-list '(change-length :divide x y :ratio )
                                (gen-random 0.21 60 '(1 1 1 1 2 3 4 4)) (mapcar 'list (l-rest-revert zn-1))))

(setq len-x4  (gen-process-list '(change-length :divide x y :ratio)
                                (gen-random 0.31 60 '(1 1 1 2 3 4 4 4)) (mapcar 'list (l-rest-revert zn-1))))

;; score-

(def-tonality
  vn1 (activate-tonality  (s-vn g 5))
  vn2 (activate-tonality  (s-vn g 5))
  vn3 (activate-tonality  (st-1 g 4))
  vn4 (activate-tonality  (st-1 f 4))
  va1 (activate-tonality  (st-1 c 4))
  va2 (activate-tonality  (st-1 b& 3))
  vc1 (activate-tonality  (st-1 c 3))
  vc2 (activate-tonality  (st-1 b& 2))
  db  (activate-tonality  (st-2i e 2))
)

(def-symbol
  vn1  (flat-them (mapcar 'list (symbol-swallow zs-1 mel-1x)))
  vn2  (flat-them (mapcar 'list (symbol-swallow zs-2 mel-2x)))
  vn3  (mapcar 'symbol-melodize (mapcar 'list (symbol-swallow zn-1 st1-sh1))) ; arpeggios
  vn4  (mapcar 'list (symbol-swallow zn-2 st1-sh2)) ; chords
  va1  (mapcar 'symbol-melodize (mapcar 'list (symbol-swallow zn-3 st1-sh3)))
  va2  (mapcar 'list (symbol-swallow zn-4 st1-sh4))
  vc1  (mapcar 'symbol-melodize (mapcar 'list (symbol-swallow zn-5 st1-sh5)))
  vc2  (mapcar 'list (symbol-swallow zn-6 st1-sh6))
  db   (symbol-swallow zn-7 st1-sh7)
)
```

```
(def-length
  vn1 len-x1
  vn2 len-x2
  vn3 (l-rest-revert zn-1) ; len-x3 ; varying rhythm ; ((1/4) (1/4) (1/4) (1/8) (1/16) (1/4) (1/8) (1/12)
  vn4 (l-rest-revert zn-2) ; len-x4  ; lengths as zone-length
  va1 len-x4 ;(l-rest-revert zn-3) ; len-x3
  va2 len-x3 ;(l-rest-revert zn-4) ; len-x4
  vc1 len-x4 ;(l-rest-revert zn-5) ; len-x3
  vc2 len-x3 ;(l-rest-revert zn-6) ; len-x4
  db  (l-rest-revert zn-7)
)

(def-velocity
  vn1 '(30)
  vn2 '(30)
  vn3 '(34)
  vn4 '(34)
  va1 '(34)
  va2 '(34)
  vc1 '(38)
  vc2 '(38)
  db '(34)
)

(def-expression
  vn1 ((legato 99 5 0.4)(humanize -2 2 0.5) (velocity 5 0.7))
  vn2 ((legato 99 5 0.4)(humanize -2 2 0.5) (velocity 5 0.7))
)

(def-zone
  tempo (but-last (append (l-rest-revert zn-1)'(1/4 1/4 1/4)))
  default (zone-expand (append  (build-list '1 59)'(3))  (l-rest-revert zn-1))
)

(def-tempo (append (build-list '50 59) '(45 40 35)))
```

```
(def-channel
  vn1 1
  vn2 2
  vn3 3
  vn4 4
  va1 5
  va2 6
  vc1 7
  vc2 8
  db  9
)

(compile-instrument-p "ccl;output:" "decet-3"
  vn1
  vn3
  va1
  vc1
  vn2
  vn4
  va2
  vc2
  db
)
```