Blaze

For Percussion Ensemble and Live Electronics

Nigel Morgan

Symbolic Composer score files, annotated by Phil Legard

Blaze

For Percussion Ensemble and Live Electronics

Nigel Morgan

Blaze is a five-movement work for percussion ensemble and electroacoustic sounds. It is based on a short poem *Deep Sea Diver* by the American poet Robert Francis. The words were inspired by the exploits of the American zoologist and explorer William Beebe. It is a remarkable poem and its economy of form and simplicity of structure fascinating source material for setting to music, or as here, used as the basis for a play of rhythms, melodies, harmonies and timbres.

The material that forms the rhythm and rhythmic structure is generated almost entirely by the poem. In music for unpitched instruments it is the length of each word that is crucial. In scoring for a collection of three drums random pitches orders are generated for the letters of each word. So if word lengths of 5,2,3 letters are indicated these might produce ‘(b a c c b) (b a) (d a c).

Melodic material played by tuned percussion instruments uses a symbolic pitch conversion from the letters in the words of the poem. This enables different tonalities to be used. See the Fifth Movement for an idea of how varied and complex such tonality organisation can become.

```
;; Blaze (1)

;; You need these functions

(defun make-length-lists (len-value lists)
"makes lists of lengths to match lists of symbols
 - to use when lengths are to be processed or varied"
 (mapcar (function (lambda (x)
            (build-list len-value x)))
            (mapcar 'length lists)))
```

Make-length-lists will produce a rhythmic schema based on one length value and a series of list data. So, `((c b c b c) (b a) (b c a a))` would become `((1/8 1/8 1/8 1/8 1/8) (1/8 1/8) (1/8 1/8 1/8 1/8))` when the function is called with a 1/8 length value.

```
(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)))
```

This function can be used to multiply the length of zones by a given number or list. So, `zone-expand '(3 1 4) '(5/8 1/4 1/2)` yields `(15/8 1/4 |2/1|)`.

```
;; Use this tonality to export

(set-tonality ng-1
  f 6       ; mid bongo
  d 6       ; hi conga
  b 5       ; lo bongo

  g 5       ; e mid tom
  e 5       ; d lo tom
)

;; NB: Zone-ratio-sc is now z-ratio-sc!
```

Here a unique tonality is defined for the skin percussion so that symbols can be mapped successfully to a percussion staff in a notation package. To see download a PDF of the output, click here.

```
;; Blaze (1)

(setq phrase-1 '(diver go down down through the green inverted dawn
                 to the dark unseen to the never day
                 the under night starless and steep
                 deep beneath deep))

(setq p1-div (mapcar 'sort-descending (do-section :all
                '(mapcar 'symbol-melodize (mapcar 'list x)) phrase-1)))

(setq temp-1 '(= = = = x = = = x
               = = = x = = = x
               = = x = =
               x = x = ))

(setq p1-divx (do-section temp-1 '(reverse x) p1-div))

(setq bar-lis  '(5 2 4 4 7 3 5 8 4
                 2 3 4 6 2 3 5 3 3 5 5 8 3
                 5 4 7 4))

(setq p-1 (gen-process '(gen-random nil x y) bar-lis '(a b c) :list)
      l-1 (make-length-lists '1/8 p-1))

(setq l-2 '((3/8 2/8)(1/4)(2/4)(2/4)(7/8)(3/8)(5/8)(3/8 3/8 2/8)(2/4)
        (1/4)(3/8)(2/4)(1/4 2/4)(1/4)(3/8)(1/4 3/8)(3/8)(3/8)(1/4 3/8)(5/8)(2/4 2/4)(3/8)
        (5/8)(2/4)(2/8 5/8)(2/4)))
```

Each movement of *Blaze* is based on a section of the poem *Deep Sea Diver* by Robert Francis. The 'symbols', which ultimately translate into 'pitches', are derived through manipulating the text of the original poem.

mirror the action in the poem. Here each word is considered a 'phrase' and sorted as a series of descending symbols. So, `(diver go down)` becomes `((v r i e d) (o g) (w o n d))`.

There are moments when the descending order is reversed to create an ascending phrase, but these are in the minority, as indicated by this template. Note that each unit on the template relates to one word in the seed phrase above.

This code generates a series of percussion symbols and note-lengths. First, **gen-process** is used to examine the values in **bar-lis** and use them as length values to create randomised phrases of the symbols (a b c). So the output, which is stored in **p-1** might be something like `((a a c a c) (b a) (a a a c) . . .)`. **Make-length-lists** is used to create a corresponding number of 1/8 note lengths for each phrase.

A secondary rhythmic line is hand-composed using values that add up to those of **l-1.** So, where **l-1** is `(1/8 1/8 1/8 1/8 1/8)` in the first phrase, **l-2** is `(3/8 2/8)`.

```
(setq l-2a (do-section '(x x x x x x x x
             x = = = x = = = = = = x x
             x x = x) '(lengths-to-rests x) l-2))
```

Some sections of **l-2** are converted into silences using a hand-composed template. So, `(3/8 2/8)` becomes `(-3/8 -1/4)`.

```
(gen-process '(gen-random nil x y) '(2 3 4 5 6 7 8) '(a b c) :list)
; ((c a) (a a b) (b a a c) (b c c b b) (b a a b b a)
;  (a a c b a c b) (c b a a c b b b))

(setq cx '(c a)
      dx '(a a b)
      ex '(b a a c)
      fx '(b c c b b)
      gx '(b a a b b a)
      hx  '(a a c b a c b)
      ix  '(c b a a c b b b))
```

A series of randomised phrases of 2 to 8 symbols in length are generated and assigned to variable **cx** to **ix**. These can be used by **eval-section**, below.

```
(setq s-2 '((d e d)))
```

This 'simple' rhythm is played by the second percussionist throughout – although the actual note-lengths and number of repetitions in each zone vary, fracturing the rhythm.

```
(setq b-lis (mapcar 'integer-to-symbol bar-lis))

(setq sym-lis (eval-section b-lis 'x 'list))
```

The list of phrase lengths, bar-lis is turned into symbolic form using integer-to-symbol, so `(5 2 4 4)` becomes `(f c e e)`. This is then passed to **eval-section** to create a series of phrases from the variables **cx** to **ix**. So, `(f c e e)` becomes `((b c c b b) (c a) (b a a c) (b a a c))`.

```
;; example of diminution / repetition

(setq div-lis '(= x = = = = = = =
                = = x = = = x = = x = x =
                x = = =))

(setq ds-1 (do-section div-lis '(symbol-repeat 2 x) sym-lis)
      dl-1 (do-section div-lis '(length-repeat-l 2 x) l-1))
```

The template **div-lis** is used to create a series of repetitions in the symbol list and corresponding repetitions and diminutions in the length values. So, the symbol set `(c a)` becomes `(c c a a)`, and the corresponding rhythm `(1/8 1/8)` becomes `(1/16 1/16 1/16 1/16)`.

```
(setq tonals (append (gen-repeat 3 (activate-tonality (chromatic b 4)))
                     (gen-repeat 6 (activate-tonality (chromatic f 4)))
                     (gen-repeat 4 (activate-tonality (chromatic c# 5)))
                     (gen-repeat 4 (activate-tonality (chromatic b 4)))
                     (gen-repeat 3 (activate-tonality (chromatic a 4)))
                     (gen-repeat 3 (activate-tonality (chromatic g 4)))
                     (gen-repeat 3 (activate-tonality (chromatic f 4)))))
```

The pitched percussion (mallets, bells) are given a series of shifting chromatic tonalities, rather than sticking to a single one such as F chromatic.

```
;; score-template

(set-tonality k-1
  e 5      ; b mid bongo
  d& 5     ; a hi conga
  c 5      ; c  lo bongo
  g 3      ; d lo tom
  c 4      ; e mid tom
)
```

The tonality mentioned above (**ng-1**) is used to map symbols to values on a percussion staff. However, these notes do not relate to the MIDI notes. If you wish to hear the piece correctly interpreted as a MIDI file this tonality will have to be used in **def-tonality** (below) instead of **ng-1**. To download an archive of the MIDI output, click here.

```
(def-tonality
 perc-1 ng-1
 perc-2 ng-1
 mallets-1 tonals
 mallets-2 tonals
 bells tonals
)
```

Some variations are added into the tuned percussion section at this stage. Mallets-1 has areas in which **find-change** is used to replace repeated notes with rests, so the symbols `(r n g e e)` become `(r n g e =)`. Mallets-2 plays chords based on groups of 2 symbols from each phrase, so `(v r i e d)` becomes `(vr ie d)`. Finally, **find-beat** applied to the chords so that the bells sound only on each beat, `(vr ie d)` becoming `(vr = =)`.

```
(def-symbol
 perc-1 ds-1
 perc-2 s-2
 mallets-1 (do-section (template-invert temp-1) '(find-change x) p1-divx)
 mallets-2 (do-section :all '(symbol-bundle 2 x) (mapcar 'find-unique p1-div))
 bells (mapcar 'find-beat (symbol-of mallets-2))
)

(def-length
 perc-1 dl-1
 perc-2 l-2
 mallets-1 l-1
 mallets-2 l-2
 bells l-2a
)
```

```
(def-velocity
 perc-1 '(90 70 60 50 65 75 80)
 perc-2 '(75 90)
 mallets-1 '(75)
 mallets-2 '(65)
 bells '(85)
)

(def-zone
 default (zone-expand '(3 1 4 1 1 1 1 1 1
                        1 1 2 1 1 1 2 1 1 2 3 1 1
                        1 4 2 4) (z-ratio-sc (length-of perc-1)))
)

(def-channel
 perc-1 10
 perc-2 10
 mallets-1 1
 mallets-2 2
 bells 3
)

#|
(def-program gm-sound-set
 mallets-1 13
 mallets-2 12
 bells 15
)
|#

(def-tempo 105)

(compile-instrument-p "ccl;output:" "blaze-ex1"
 perc-1
 perc-2
 mallets-1
 mallets-2
 bells
)
```

Velocity here is either fixed or a cycle which will either repeat or re-start depending on the zone length.

Zone-expand is used to create a series of repeats, for example the first zone is now tripled in length, giving us three repetitions of the material aligned with that zone in **def-symbol**, **def-length** and **def-velocity**.

Remove the comments here to assign the correct sounds to each voice (percussion being mapped above to the default MIDI percussion channel: 10).

```
;; Blaze (2)

(setq phrase-2 '(diver fall
                 and failing fight
                 your weed dense way
                 until you crawl
                 until you touch
                 weird water land
                 and stand
                 ))

(setq temp-2 '(= =
               = = x
               = = = x
               = = x
               = = x
               = x =
               = x))

(setq p2-div (mapcar 'sort-descending (do-section :all
                '(mapcar 'symbol-melodize (mapcar 'list x)) phrase-2)))

(setq p2-divx (do-section temp-2 '(reverse x) p2-div))

 (setq bar-lis2 '(5 4 3 7 5  4 4 5 3   5 3 5
                 5 3 5   5 5 4 3 5))

(setq z-ex-lis '(2 4 1 4 3   1 1 2 1 1 1 2
                 1 1 2   2 3 1 1 2 ))

(setq b-lis2 (mapcar 'integer-to-symbol bar-lis2))

(setq p-2 (gen-process '(gen-random nil x y) bar-lis2 '(a b c) :list)
      l-2a (make-length-lists '1/8 p-2))

(setq l-2b '((3/8 2/8)(2/4)(3/8)(2/4 3/8)(5/8)  (2/4)(2/4)(5/8)(3/8)
        (5/8)(3/8)(5/8)   (5/8)(3/8)(5/8) (5/8)(2/8 3/8)(2/4)(3/8)(5/8)))

(setq l-2c (do-section '(x x x x x  = = = = x x x
                        x x x = = = x x) '(lengths-to-rests x) l-2b))
```

The composition of this movement is similar to that of the first with any exceptions noted below.

```
(setq div-lis2 '(= = = = x = = x = = = = = = = x x = = =))
(setq exp-lis2 '(= = = = = = = = = = = = = x = = = = x))

(setq sym-lis2 (eval-section b-lis2 'x 'list))

(setq ds-2 (do-section div-lis2 '(symbol-repeat 2 x) sym-lis2)
      dl-2 (do-section exp-lis2 '(change-length :times 2 x :ratio)
(do-section div-lis2 '(length-repeat-l 2 x) l-2a)))


(gen-process '(gen-random nil x y) '(2 3 4 5 6 7 8) '(a b c) :list)
; ((c a) (a c c) (b a b c) (a b c b a) (b a b c c a) (a b b c
; a a a) (c a c a b a a a))

(setq cx '(c a)
      dx '(a c c)
      ex '(b a b c)
      fx '(a b c b a)
      gx '(b a b c c a)
      hx  '(a b b c a a a)
      ix  '(c a c a b a a a))

(setq s-2 '((e d e)))

(setq tonal (activate-tonality (chromatic f 4)))

;; score

(def-tonality
 perc-1 ng-1
 perc-2 ng-1
 mallets-1 tonal
 mallets-2 tonal
 bells tonal
)
```

A fixed tonality for the tuned percussion this time.

```
(def-symbol
 perc-1 ds-2
 perc-2 s-2
 mallets-1  (do-section (template-invert temp-2) '(find-change x) p2-divx)
 mallets-2 (do-section :all '(symbol-bundle 2 x) (mapcar 'find-unique p2-div))
 bells (mapcar 'find-beat (symbol-of mallets-2))
)

(def-length
 perc-1 dl-2
 perc-2 l-2b
 mallets-1 l-2a
 mallets-2 l-2b
 bells l-2c
)

(def-velocity perc-1 (p-replace nil '(4 11 14 19) '((126)) (gen-repeat (length ds-2) '((70))))
 perc-2 (p-replace nil '(4 11 14 19) '((126)) (gen-repeat (length ds-2) '((70))))
 mallets-1 (p-replace nil '(4 11 14 19) '((126)) (gen-repeat (length ds-2) '((70))))
 mallets-2 (p-replace nil '(4 11 14 19) '((126)) (gen-repeat (length ds-2) '((70))))
 bells (p-replace nil '(4 11 14 19) '((126)) (gen-repeat (length ds-2) '((70))))
)

(def-zone
 default (zone-expand z-ex-lis (z-ratio-sc (length-of perc-1)))
)

(def-channel
 perc-1 10
 perc-2 10
 mallets-1 1
 mallets-2 2
 bells 3
)

(def-program gm-sound-set
 mallets-1 13
 mallets-2 12
 bells 15
)
```

For each zone a velocity of 70 is generated using **gen-repeat**. For selected zones (4, 11, 14 and 19) the value is replaced with 126 creating massed crescendos marked variously as ***ff*** to ***ffff*** in the score.

```
(def-tempo 110)

(compile-instrument-p "ccl;output:" "blaze-2ng"
 perc-1
 perc-2
 mallets-1
 mallets-2
 bells
)
```

```
;; Blaze (3)

(setq phrase-3 '(diver come up up
                 through the green
                 into the light
                 the sun the seen
                 but in the clutch
                 of your dripping hand))


(setq temp-3 '(= = = =
               = = x
               = =
               = x
               = = = x
               = = = x))

(setq p3-div (mapcar 'sort-ascending (do-section :all
             '(mapcar 'symbol-melodize (mapcar 'list x)) phrase-3)))

(setq p3-divx (do-section temp-3 '(reverse x) p3-div))

(setq bar-lis3 '(5 4 2 2 7 3 5 4 3 5
                 3 3 3 4 3 2 3 5 2 4 8 4))

(setq z-ex-lis3 '(3 2 1 8 2 1 2 1 1 2
                  1 2 1 2 1 1 1 2 1 1 1 2 ))

(setq b-lis3 (mapcar 'integer-to-symbol bar-lis3))

(setq p-3 (gen-process '(gen-random nil x y) bar-lis3 '(a b c) :list)
      l-3a (make-length-lists '1/8 p-3))

(setq l-3b '((3/8 2/8)(2/4)(2/8)(2/8)  (7/8)(3/8)(5/8) (4/8)(3/8)(5/8)
        (3/8)(3/8)(3/8)(4/8) (3/8)(2/8) (3/8)(5/8)  (2/8)(4/8)(4/8 4/8) (4/8)))

(setq l-3c (do-section '(x x x x = = = x x x = = = =
                         x x x x x x x x) '(lengths-to-rests x) l-3b)) ; bells

(setq div-lis3 '(= = = = = = x = = x
                   = x = x = = = = = = = =))
```

Note use of **sort-ascending**, with selected phrases reversed (descending).

```
(setq exp-lis3  '(= = = =  = = =  = = =
                  x = x =  = = = x  = = x =))

(setq ds-3 (do-section div-lis3 '(symbol-repeat 2 x) sym-lis3)
      dl-3 (do-section exp-lis3 '(change-length :times 2 x :ratio)
(do-section div-lis3 '(length-repeat-l 2 x) l-3a)))


(gen-process '(gen-random nil x y) '(2 3 4 5 6 7 8) '(a b c) :list)
; ((b a) (b c b) (a a c b) (b a b a a) (b a c b b a) (b b a c
; a b b) (a b c c a a a b))

(setq cx '(b a)
      dx '(b c b)
      ex '(a a c b)
      fx '(b a b a a)
      gx '(b a c b b a)
      hx  '(b b a c a b b)
      ix  '(a b c c a a a b))

(setq s-3 '((d d e)))

(setq sym-lis3 (eval-section b-lis3 'x 'list))

(setq tonal (activate-tonality (chromatic f 4)))

(setq trup (gen-process '(symbol-transpose x y) '(-8 -7 -6 -5 -4 -3 -2 -1 0) '(u p)))

;; score

(def-tonality
 perc-1 ng-1
 perc-2 ng-1
 mallets-1 tonal
 mallets-2 tonal
 bells tonal
)
```

This sequence, based on transpositions of the symbol set `(u p)` is inserted into the third zone of the composition. While the other instruments repeat their material, the mallet instruments make their ascent.

```
(def-symbol
 perc-1 ds-3
 perc-2 s-3
 mallets-1 (p-replace nil '(3) (list trup) (do-section (template-invert temp-3) '(find-change x) p3-divx))
 mallets-2 (p-replace nil '(3) (list trup) (do-section :all '(symbol-bundle 2 x) (mapcar 'find-unique p3-div)))
 bells (mapcar 'find-beat (symbol-of mallets-2))
)

(def-length
 perc-1 dl-3
 perc-2 l-3b
 mallets-1 l-3a
 mallets-2 l-3b
 bells l-3c
)


(def-velocity
 perc-1 (p-replace nil '(10 12 17 20 21) '((126)) (gen-repeat (length ds-3) '((75))))
 perc-2   (p-replace nil '(10 12 17 20 21) '((126)) (gen-repeat (length ds-3) '((70))))
 mallets-1 (p-replace nil '(10 12 17 20 21) '((126)) (gen-repeat (length ds-3) '((90))))
 mallets-2 (p-replace nil '(10 12 17 20 21) '((126)) (gen-repeat (length ds-3) '((80))))
 bells (p-replace nil '(10 12 17 20 21) '((126)) (gen-repeat (length ds-3) '((70))))
)


(def-zone
 default (zone-expand z-ex-lis3 (z-ratio-sc (length-of perc-1)))
 tempo '(15/8 1/1 1/4 2/1 7/4 3/8 5/4 1/2 3/8 5/4 3/4 3/4 3/4 1/1 3/8 1/4 3/8 5/4 1/4 1/2 2/1 1/2)
)

(def-tempo '(55 65 75 85 90 95 100  110 110 110 100 100 100 100 102 104 108 110 112 114 116 118))

(def-channel
 perc-1 10
 perc-2 10
 mallets-1 1
 mallets-2 2
 bells 3
)
```

The processing here is similar to the last movements – the introduction of rests in certain passages via find-change and the pairing of values in the second mallet part. However, note also the inclusion of the ascending pattern (**trup**) replacing the symbol data for the third zone.

mapping data from **def-tempo** against a schema defined in **def-zone**.

```
(def-program gm-sound-set
 mallets-1 13
 mallets-2 12
 bells 15
)
(compile-instrument-p "ccl;output:" "blaze-ex3"
 perc-1
 perc-2
 mallets-1
 mallets-2
 bells
)
```

```
;; Blaze (4)

(setq phrase-4 '(but in the clutch of your dripping hand
                 diver bring some uncouth thing
                 that we could swear
                 and would have sworn
                 was never born
                 or could ever be))

(setq temp-4 '(= = = x = = x =
               = = = = x
               = = = x
               = = = x
               = = x
               = = = x))

(setq p4-div (mapcar 'sort-ascending (do-section :all
             '(mapcar 'symbol-melodize (mapcar 'list x)) phrase-4)))

(setq p4-divx (do-section temp-4 '(reverse x) p4-div))

(create-tonality F-scale '(f 3 g 3 b 3 c 4 e 4))
; (a c g i j)

(setq bar-lis4 '(3 2 3 5  2 4 8 4 5 5
                 4 7 5  4 2 5 5  3 5 4
                 5  3 5 4  2 4 5 2))

(setq z-ex-lis4 '(1 1 1 1  1 1 1 1  1 1
                  1 1 1  1 1 1 1  1 1 1
                  1  1 1 1  1 1 1 3 ))

(setq b-lis4 (mapcar 'integer-to-symbol bar-lis4))

(setq p-4 (gen-process '(gen-random nil x y) bar-lis4 '(a b c) :list)
      l-4a (make-length-lists '1/8 p-4))

(setq l-4b '((3/8)(2/8)(3/8)(5/8)  (2/8)(4/8)(3/8 5/8)(4/8)  (5/8)(5/8)
        (4/8)(2/8 5/8)(5/8) (4/8)(2/8)(5/8)(5/8) (3/8)(5/8)(4/8)
        (5/8)  (3/8) (2/8 3/8) (4/8) (2/8)(2/8 2/8)(5/8) (2/8)))
```

```
(gen-process '(gen-random 0.1333567 x y) '(2 3 4 5 6 7 8) '(a b c) :list)
; ((b a) (c b a) (a c b a) (a a c b a) (b a a c b a) (c b a a
; c b a) (a c b a a c b a))

(gen-process '(gen-random 0.1333567 x y) '(2 3 4 5 6 7 8) '(g i j) :list)
; ((i g) (j i g) (g j i g) (g g j i g) (i g g j i g) (j i g g
; j i g) (g j i g g j i g))


(setq cx '(b a)
      dx '(c b a)
      ex '(a c b a)
      fx '(a a c b a)
      gx '(b a a c b a)
      hx  '(c b a a c b a)
      ix  '(a c b a a c b a))

(setq cy '(i g)
      dy '(j i g)
      ey '(g j i g)
      fy '(g g j i g)
      gy '(i g g j i g)
      hy  '(j i g g j i g)
      iy  '(g j i g g j i g))


(setq s-4 '(e e d)
      s-4i '(c a))


(setq sym-lis4 (eval-section b-lis4 'x 'list)
      mel-lis4 (eval-section b-lis4 'y 'list))

(setq div-lis4 '(= = = x  = = x = = =
                 = = =  = = = x  = = =
                 x   = = x  = = = x))

(setq dim-lis4  '(= = = =  = = = x x x
                  x x x = = = =  = = =
                  =  = = =  = = = =))
```

Unlike previous movements with only set line of material generated by the use of **eval-section**, the fourth movement has two.

```
(setq ds-4 (do-section div-lis4 '(symbol-repeat 2 x) sym-lis4)
      dl-4 (do-section dim-lis4 '(change-length :divide 2 x :ratio)
           (do-section div-lis4 '(length-repeat-l 2 x) l-4a))
      dl-4i (do-section dim-lis4 '(change-length :divide 2 x :ratio) l-4a)
      dl-4a (do-section dim-lis4 '(change-length :divide 2 x :ratio) l-4b))

(setq tonal (activate-tonality (chromatic f 3)))

(setq tonals (append (gen-repeat 8 (activate-tonality (F-scale f 4)))
                     (gen-repeat 5 (activate-tonality (F-scale b 4)))
                     (gen-repeat 4 (activate-tonality (F-scale f 4)))
                     (gen-repeat 4 (activate-tonality (F-scale g 4)))
                     (gen-repeat 3 (activate-tonality (F-scale a 4)))
                     (gen-repeat 4 (activate-tonality (F-scale b 4)))))

(setq tonalx (append (gen-repeat 8 (activate-tonality (chromatic f 4)))
                     (gen-repeat 5 (activate-tonality (chromatic b 4)))
                     (gen-repeat 4 (activate-tonality (chromatic f 5)))
                     (gen-repeat 4 (activate-tonality (chromatic g 4)))
                     (gen-repeat 3 (activate-tonality (chromatic a 4)))
                     (gen-repeat 4 (activate-tonality (chromatic b 4)))))
```

This movement also makes use of tonal movement, transposing the F-scale up and down by differing steps and octaves (note in **tonalx** the movement to the fifth octave).

```
(setq word-play4 (do-section temp-4 '(symbol-shuffle x 0.1)
                             (do-section temp-4 '(gen-repeat 2 x)
                             (do-section (template-invert temp-4) '(find-change x)  p4-divx))))
```

Add repeating material to the percussion part by repeating and selected phrases twice and shuffling their contents.

```
;; score

(def-tonality
 perc-1 ng-1
 perc-2 ng-1
 mallets-1 tonalx
 mallets-2 tonals
 mallets-2a tonals
 glock tonalx
)
```

```
(def-symbol
 perc-1 ds-4
 perc-2 s-4
 mallets-1 (do-section (template-invert dim-lis4)  '(filter-delete '(i o u) x) word-play4)
 mallets-2 (do-section :all '(p-replace nil 'first '= x)(do-section :all '(find-change x) mel-lis4))
 mallets-2a s-4i
 glock (do-section (template-invert dim-lis4)  '(filter-delete '(a e) x) word-play4)
)

(def-length
 perc-1 dl-4
 perc-2 dl-4a
 mallets-1 dl-4
 mallets-2 dl-4i
 mallets-2a dl-4a
 glock dl-4
)

(def-velocity
 perc-1   (p-replace nil '(8 9 10 11 12) '((126)) (gen-repeat (length ds-4) '((60))))
 perc-2   (p-replace nil '(8 9 10 11 12) '((126)) (gen-repeat (length ds-4) '((50))))
 mallets-1 (p-replace nil 'last (list (gen-dim 70  30 12))
           (p-replace nil '(8 9 10 11 12) '((100)) (gen-repeat (length ds-4) '((70)))))
 mallets-2 (p-replace nil 'last (list (gen-dim 70  30 12))
           (p-replace nil '(8 9 10 11 12) '((100)) (gen-repeat (length ds-4) '((70)))))
 mallets-2a (p-replace nil 'last (list (gen-dim 70  30 12))
            (p-replace nil '(8 9 10 11 12) '((120)) (gen-repeat (length ds-4) '((110)))))
 glock (p-replace nil 'last (list (gen-dim 70  30 12))
       (p-replace nil '(8 9 10 11 12) '((120)) (gen-repeat (length ds-4) '((70)))))
)

(def-zone
 default (zone-expand z-ex-lis4 (z-ratio-sc (length-of perc-1)))
)

(def-tempo 60)
```

Rests are introduced into the parts by using **filter-delete** to replace selected symbols with rests.

```
(def-channel
 perc-1 10
 perc-2 10
 mallets-1 1
 mallets-2 2
 mallets-2a 2
 glock 4dl-4a
)

(def-program gm-sound-set
 mallets-1 13
 mallets-2 12
 bells 15
)

(compile-instrument-p "ccl;output:" "blaze-ex4"
 perc-1
 perc-2
 mallets-1
 mallets-2
 mallets-2a
 glock
)
```

```
;; Blaze (5)

(defun make-pauses (lis)
"creates a list of pause symbols"
 (e-substitute (build-list '= (length lis)) lis lis ))

(setq phrase-5  '(that we could swear
                  and would have sworn
                  was never born
                  or could ever be
                  anywhere
                  blaze on our sight
                  make us see
                  ))

(setq temp-5 '(= = = x
               = = = x
               = = x
               = = = x
               = = =
               = = = x
               =
               = = x
               =
))

(setq temp-5a '(= = = x
                = = = x
                = = x
                = = = x
                x = x
                = = = =
                x
                = = =
                x
))

(setq temp-5b
 '(x x x =  x x x =  x x = x x x =  = = =  = = = = =  =  = = = =))
```

This new function will create pauses for the length of a given list, e.g. `make-pauses '(a b c)` yields `(= = =)`.

```
(setq p5-div (e-insert '((= = = = =)) '(15 17 22 26) (mapcar 'sort-ascending
            (do-section :all '(mapcar 'symbol-melodize (mapcar 'list x)) phrase-5))))

(setq p5-divx (do-section temp-5 '(reverse x) p5-div))

(setq bar-lis5 '(4 2 5 5  3 5 4
                 5  3 5 4  2 4 5 2
                 5 8 5
                 5 2 3 5
                 5  4 2 3   5))

(setq z-ex-lis5 '(1 1 1 1  1 1 1
                  1  1 1 1  1 1 1 1
                  1 1 1
                  1 1 1 1
                  1  1 1 1  1))

(setq b-lis5 (mapcar 'integer-to-symbol bar-lis5))

(setq p-5 (gen-process '(gen-random nil x y) bar-lis5 '(a b c) :list)
      l-5a (make-length-lists '1/8 p-5))

(setq l-5b '((4/8)(2/8)(5/8)(5/8) (3/8)(5/8)(4/8)
        (5/8)  (3/8) (2/8 3/8) (4/8) (2/8)(2/8 2/8)(5/8) (2/8)
        (5/8) (1/8 2/8 5/8)(5/8)
        (5/8) (2/8) (3/8) (5/8)
        (5/8) (4/8) (2/8) (3/8)
        (5/8)))

(gen-process '(gen-random nil x y) '(2 3 4 5 6 7 8) '(a b c) :list)
; ((c b) (b b c) (b c c b) (a c b c a) (b b a c c b) (c c a a
;b c c) (c b a b c b a c))

(setq cx '(c b)
      dx '(b b c)
      ex '(b c c b)
      fx '(a c b c a)
      gx '(b b a c c b)
      hx  '(c c a a b c c)
      ix  '(c b a b c b a c))
```

```
(setq s-5 '((d e)))

(setq sym-lis5 (p-replace nil '(15 17 22 26) '((= = = = =))  (eval-section b-lis5 'x 'list)))


(setq div-lis5 '(= = = x  = = = x
                 = = x   = = = x
                 = = =
                 x = = x
                 =  x = x  =))

(setq ds-5  (p-replace nil '(15 17 22 26) '((= = = = =))
            (do-section div-lis5 '(symbol-shuffle x nil)
            (do-section div-lis5 '(symbol-repeat 2 x) sym-lis5))))

(setq p-5 (gen-process '(gen-random nil x y) bar-lis5 '(a b c) :list)
      l-5a (make-length-lists '1/8 ds-5)
      dl-5 (do-section div-lis5 '(change-length :divide 2 x :ratio) l-5a))

(setq word-play5 (do-section (template-invert temp-5) '(find-change x) p5-divx)
      word-play5i (do-section temp-5a '(make-pauses x) p5-divx))

(setq chord-play5 (p-replace nil '(15 17 22 26) '((= = = = =))
                             (do-section :all '(symbol-bundle '2 x) (mapcar 'find-unique p5-div)))
      chord-play5i (mapcar ' find-beat (do-section  temp-5b '(make-pauses x) chord-play5)))

(setq tonal (activate-tonality (chromatic f 4)))

(setq tonals (append (gen-repeat 4 (activate-tonality (chromatic f 4)))
             (gen-repeat 4 (activate-tonality (chromatic a 4)))
                     (gen-repeat 3 (activate-tonality (chromatic c# 4)))
                     (activate-tonality (chromatic a 4))
                     (activate-tonality (chromatic b 4))
                     (activate-tonality (chromatic c# 4))
                     (activate-tonality (chromatic d# 4))
                     (gen-repeat 3 (activate-tonality (chromatic f 5)))
                     (gen-repeat 5 (activate-tonality (chromatic a 4)))
                     (gen-repeat 4 (activate-tonality (chromatic b 4)))))
```

This section introduces chordal elements. **Find-unique** functionality removes any repeated values from a lists, so `(a h t t)` becomes `(a h t)`. **Symbol-bundle** is then used to pair the symbols – so we get `(ah t)` in this case. **P-replace** is used to replace selected five-symbol length lists with rests. In the second voice, a template (**temp-5b**) is used to create areas of rests with make-pauses. **Find-beat** will only keep the first element of a list, or an element occurring after a pause. In our example `(ah t)` becomes `(ah =)`. `(ah t = t ah)` would become `(ah = = t =)`.

```
;; score
(def-tonality
 perc-1 ng-1
 perc-2 ng-1
 mallets-1 tonals
 mallets-2 tonals
 mallets-3 tonals
 bells tonals
)

(def-symbol
 perc-1 ds-5
 perc-2 (p-replace nil '(15 17 22 26) '((=)) (gen-repeat (length ds-5) '((d e))))
 mallets-1 word-play5
 mallets-2 chord-play5
 mallets-3 word-play5i
 bells chord-play5i
)

(def-length
 perc-1 dl-5
 perc-2 l-5b
 mallets-1 l-5a
 mallets-2 l-5b
 mallets-3 l-5a
 bells l-5b
)

(def-velocity
 perc-1 (p-replace nil '(3 7 10 14 16 18 21 23 25) '((126)) (gen-repeat (length ds-5) '((60))))
 perc-2   (p-replace nil '(3 7 10 14 16 18 21 23 25) '((126)) (gen-repeat (length ds-5) '((60))))
 mallets-1 (p-replace nil '(3 7 10 14 16 18 19 20 21 22 23 25) '((110)) (gen-repeat (length ds-5) '((50))))
 mallets-2  (p-replace nil '(3 7 10 14 16 18 19 20 21 22 23 25) '((110)) (gen-repeat (length ds-5) '((70))))
 mallets-3 (p-replace nil '(3 7 10 14 16 18 19 20 21 22 23 25) '((110)) (gen-repeat (length ds-5) '((70))))
 bells (p-replace nil '(3 7 10 14 16 18 19 20 21 22 23 25) '((100)) (gen-repeat (length ds-5) '((80))))
)

(def-zone
 default (z-ratio-sc (length-of perc-1))
)
```

```
(def-channel
 perc-1 10
 perc-2 10
 mallets-1 1
 mallets-2 2
 mallets-2a 2
 mallets-3 7
 bells 3
)

(compile-instrument-p "ccl;output:" "blaze-ex5"
 perc-1
 perc-2
 mallets-1
 mallets-2
 mallets-3
 bells
)
```