Interactions

For piano (left hand)

Annotated Symbolic Composer Scores

Nigel Morgan

Interactions

for piano (left hand)

Nigel Morgan

About the piece

The five movements of this work were created with the interaction of the Symbolic Composer software, an application co-developed by the composer. The Symbolic Composer score files now follow with annotations to give an insight into the processes used in the composition of the piece. An introduction to Symbolic Composer, as well as the complete score for Interactions can be found on the composer's website www.nigel-morgan.co.uk

```
;;;; Interactions for Piano (left hand) No.1

;;; Slonimsky Patterns 27-31 (Symmetrical Interpolations of One Note)


;; material

(setq tonal (activate-tonality (chromatic C 5)))


(setq mat1 '(-m a a a a a m m m m m y m m m m a a a a a))

(setq mat1x '((-m a a a a) (a m m m m) (y m m m m) (m a a a a a)))

(setq sl27 '(a b g l)) ; Slonimsky Pattern 27

(setq sl27tr (gen-process  '(symbol-transpose x y) '(-12 0 12 0) sl27))
; > (-m -l -g -b a b g l m n s x a b g l)


(setq mat1x1 (car mat1x))
(setq sl27a '(-l -g -b b))

(setq mat1x2 (cadr mat1x))
(setq sl27b '(n m l g b))

(setq mat1x3 (caddr mat1x))
(setq sl27c '(l m n s x))

(setq mat1x4 (cadddr mat1x))
(setq sl27d '(l g b -b))

(setq var1
      (gen-collect 0.4 7 :list '(append
                                 (gen-random-keep nil (get-random 1 4) '(1 3) mat1x1 sl27a)
                                 (gen-random-keep nil (get-random 1 4) '(1 3) mat1x2 sl27b)
                                 (gen-random-keep nil (get-random 1 4) '(1 3) mat1x3 sl27c)
                                 (gen-random-keep nil (get-random 1 4) '(1 3) mat1x4 sl27d))))
```

Pre-composed material.

Create a list of values derived from transpositions of Slonimsky pattern 27.

Material consisting of interpolations of the composed material (mat1) with transpositions of the Slonimsky pattern (sl127) are created using gen-random-keep.

For each pattern of material, the first and third values are kept, while the rest of the pattern is randomly selected from the transposed Slonimsky pattern.

This process is repeated seven times using gen-collect, creating the symbol stream that will, with further processing, become the pitches used in the compositon.

```
(setq accents '(21
                15 6
                5 16
                13 8
                4 8 9
                12 6 3
                3 6 4 2 6
                7 4 2 5 3))

;; score

(def-symbol
 l-hand (append mat1 (flatten var1) (flatten (reverse var1)) (reverse mat1))
)

(def-length
 l-hand '(1/8)
)

(def-tonality
 l-hand tonal
)

(def-velocity
 l-hand (gen-accent 90 (append accents (reverse accents))
                       (gen-cresc-dim 25 90 (length (symbol-of l-hand))))
)

(def-zone
 l-hand  (make-zone (symbol-repeat (length (flatten (symbol-of l-hand))) '(1/8)))
)

(def-tempo 160)

(compile-instrument-p "ccl;output:" "interaction-1"
 l-hand
)
```

The material for the pitches generated above is finally processed by adding the precomposed material (mat1) to the beginning and creating a palindromic structure to the whole by using the reverse function.

A crescendo and diminuendo is generated for as many notes as the symbol list of the left hand has. Strong accents are inserted into this list, the incidence of which is controlled by the list of accents defined above.

```
;;;; Interactions for Piano (left hand) No.2

;;;  Slonimsky Patterns 27-31 (Symmetrical Interpolations of One Note)


;; functions

(defun rhy-mkr (lis)
(cond ((is-chord  lis) '(1/16 1/16))
       (t '1/8)))

;; material
(setq tonal (activate-tonality (chromatic d& 5)))

(setq mat1 '(-m a a a a a m m m m m y m m m m a a a a a))

(setq mat1x '((-m a a a a) (a m m m m) (y m m m m) (m a a a a a)))

(setq sl28 '(a c g k)) ; Slonimsky Pattern 28

(setq sl28tr (gen-process  '(symbol-transpose x y) '(-12 0 12 0) sl28))
; > (-m -k -g -c a c g k m o s w a c g k)

(setq mat1x1 (car (p-select 0 mat1x))) ; material from which random selection is made
(setq sl28a '(-k -ga -c c))

(setq mat1x2 (car (p-select 1 mat1x)))
(setq sl28b '(o sm k ga c))

(setq mat1x3 (car (p-select 2 mat1x)))
(setq sl28c '(k sm o s w))

(setq mat1x4 (car (p-select 3 mat1x)))
(setq sl28d '(k gm c -c))

(setq var1 ; 7 variations in which selected areas of mat1x groups are filled with SL28
      (gen-collect 0.4 7 :list '(append
                                  (gen-random-keep nil (get-random 1 4) '(1 3) mat1x1 sl28a)
                                  (gen-random-keep nil (get-random 1 4) '(1 3) mat1x2 sl28b)
                                  (gen-random-keep nil (get-random 1 4) '(1 3) mat1x3 sl28c)
                                  (gen-random-keep nil (get-random 1 4) '(1 3) mat1x4 sl28d))))
```

As in the first movement, only using Slonimsky pattern 28 as the material.

Note derivation of chords that can be played easily by one hand from Slonimsky material.

```
(setq mat2 (do-section :all '(find-change x) mat1))
(setq var2 (do-section :all '(find-change x) var1))
(setq var2a (append mat2 (flatten var2) (flatten (reverse var2)) (reverse mat2)))

(setq accents '(21
                15 6
                5 16
                13 8
                4 8 9
                12 6 3
                3 6 4 2 6
                7 4 2 5 3))

;; score

(def-symbol
 l-hand  (append mat2 (flatten var2)
                 (symbol-melodize (flatten (reverse var2)))
                 (reverse mat2))
)

(def-length
 l-hand (append (flatten (mapcar 'rhy-mkr mat2))
                (symbol-repeat (length (flatten var1)) '(1/8))
                (flatten (mapcar 'rhy-mkr (flatten (reverse var2))))
                (flatten (mapcar 'rhy-mkr mat2)))
)

(def-tonality
 l-hand tonal
)

(def-velocity
 l-hand (gen-accent 90 (append  (reverse accents) accents)
                       (gen-dim-cresc 100 30 (length (get-symbols-of 'l-hand))))
)

(def-zone
 l-hand  (make-zone (get-lengths-of 'l-hand))
)
```

Use find-change to replace repeating symbols with =. This is done for both the precomposed material and the processed Slonimsky pattern, which are then appended to each other and made into a palindrome.

Symbol list for the piece is created by appending the processed sections. Note the use of symbol-melodize to break apart the chords into individual notes in the middle section of the movement

The notelengths are derived from processing the symbl list through the function rhy-mkr (defined above). This is a simple logical function - if the symbol is a chord then it is given a duration of 1/16, as is the note which follows it. Otherwise the durations are of 1/8.

```
(def-tempo 130)

(compile-instrument-p "ccl;output:" "interaction-2"
 l-hand
)
```

```
;;;; Interactions for Piano (left hand) No.3

;;;  Slonimsky Patterns 27-31 (Symmetrical Interpolations of One Note)

;; functions

(defun make-chord-pairs (lis)
 "makes a sequence of chords from alternate pairs of symbols"
  (prog (out)
    loop
    (cond ((null lis) (return (but-last (but-last out)))))
    (setq out (append out (list (compress
                                (append (list (car lis)) (list (caddr lis))))))))
    (setq lis (cdr lis))
    (go loop)))

;; material

(setq tonal (activate-tonality (chromatic g& 5)))


(setq mat1 '(-m a a a a a m m m m m y m m m m a a a a a))

(setq mat1x '((-m a a a a) (a m m m m) (y m m m m) (m a a a a a)))

(setq sl29a '(a d g j)) ; Slonimsky Pattern 29
(setq sl29b '(b e h k))

(setq cm29 '(a b d e g h j k)) ; octotonic

(setq sl29tr (gen-process  '(symbol-transpose x y) '(-12 0 12 0) sl29a))
; > (-m -j -g -d a d g j m p s v a d g j)

(setq cm29tr (gen-process  '(symbol-transpose x y) '(-12 0 12 0) cm29))
; > (-m -l -j -i -g -f -d -c a b d e g h j k m n p q s t v w a b d e g h j k)

(setq thirds (make-chord-pairs cm29tr))
; > (-m-j -l-i -j-g -i-f -g-d -f-c -da -cb ad be dg eh gj hk
; > jm kn mp nq ps qt sv tw va wb ad be dg eh gj hk)

;;; (symbol-trim (- (length cm29tr) 2) (symbol-chordize-scroll -2 nil nil cm29tr))
```

Creation of material from Slonimsky pattern 29 and a transposition. Lists of transposed instances of these sequences are created using symbol-transpose, as before.

Uses the function defined above (make-chord-pairs) to group the symbol list into two-note chords.

```
(setq template '(= x x = x = x x = x = x = x x = x x x x x))

(setq intro (fill-template template thirds))

(setq mat3rds (gen-evolve 6 '(gen-random nil 21 x) thirds :list))

(setq matpos (gen-evolve 6 '(symbol-shuffle x 0.1) template :list))

(setq matfin  (symbol-divide 21 nil nil
                             (fill-template (flatten matpos) (flatten mat3rds))))

(def-neuron dynamics
  (in 1 '=) '0
 (otherwise 64))

(setq dynamicsx (run-neuron 'dynamics
                            (flatten matfin)))


(def-neuron dynamicsy ; accenting syncopation
   (in 1 '0 -1)  (set-random 100 127)
   (otherwise (in 1 0)) )

(setq dynamicsz (run-neuron 'dynamicsy
                             dynamicsx))


;; score

(def-symbol
  l-hand  (flatten (append
                    (symbol-transpose -24 intro)
                    matfin
                    (reverse matfin)
                    (symbol-transpose 24 intro)))
)

(def-length
 l-hand  '(1/8)
)
```

Create material for the introduction to the movement by creating a template, which is filled by the chords generated above. Each x in the template is filled with a value.

```
(def-tonality
 l-hand tonal
)

(def-velocity
 l-hand   (append
           (gen-cresc 72 110 21) dynamicsz (reverse dynamicsz) (gen-cresc 72 110 21))
)

(def-zone
 l-hand  (make-zone (symbol-repeat (length (flatten (get-symbols-of 'l-hand))) '(1/8)))
)

(def-tempo 150)

(compile-instrument-p "ccl;output:" "interaction-3"
 l-hand
)
```

```
;;;; Interactions for Piano (left hand) No.4

;;;  Slonimsky Patterns 27-31 (Symmetrical Interpolations of One Note)

;; functions

(defun make-chord-pairs (lis)
 "makes a sequence of chords from alternate pairs of symbols"
  (prog (out)
    loop
    (cond ((null lis) (return (but-last (but-last out)))))
    (setq out (append out (list (compress
                                 (append (list (car lis)) (list (caddr lis))))))))
    (setq lis (cdr lis))
    (go loop)))

;; material

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

(create-tonality sl30t '(b 4 e& 5 f 5 g 5))

(setq chord1 (activate-tonality (sl30t b 4)))


(setq mat1 '(-m a a a a a m m m m m y m m m m a a a a a))

(setq mat1x '((-m a a a a) (a m m m m) (y m m m m) (m a a a a a)))

(setq sl30 '(a e g i)) ; Slonimsky Pattern 30


(setq sl30tr (gen-process  '(symbol-transpose x y) '(-24 -12 0 12) sl30))
 ; > (-y -u -s -q -m -i -g -e a e g i m q s u)

(setq chord '(abcd))

(setq basses (make-chord-pairs sl30tr))
; >(-y-s -u-q -s-m -q-i -m-g -i-e -ga -ee ag ei gm iq ms qu)
```

```
(setq tonal-2
      (distort-tonality 4 1.5 0.1
                        (activate-tonality
                         (sl30t b 4) (sl30t b 4) (sl30t b 4) (sl30t b 4)
                         (sl30t b 4) (sl30t b 4) (sl30t b 4)
                         (sl30t b 4) (sl30t b 4) (sl30t b 4)
                         (sl30t b 4) (sl30t b 4) (sl30t b 4) (sl30t b 4))))


(setq template '(= x x = x = x x = x = x = x x = x x x x x))

(setq intro (fill-template template chord))

(setq matchds (gen-evolve 13 '(gen-random nil 21 x) chord :list))
(setq matbass (gen-evolve 13 '(gen-random nil 21 x) basses :list))

(setq matpos1 (gen-evolve 13 '(symbol-shuffle x 0.1) template :list))
(setq matfin  (symbol-divide 21 nil nil (fill-template (flatten matpos1) (flatten matchds))))

(setq basst (fill-rest (flatten matpos1) (flatten matbass)))
(setq silence (build-list '= 21))


(def-neuron dynamics1
  (in 1 '=) '0
 (otherwise 25))

(setq dynamicsx1 (run-neuron 'dynamics1
                             (flatten matfin)))

(def-neuron dynamicsy1 ; accenting syncopation
   (in 1 '0 -1)  (set-random 30 40)
   (otherwise (in 1 0)))

(setq dynamicsz1 (run-neuron 'dynamicsy1
                              dynamicsx1))
```

```
;; score

(def-symbol
 l-hand (append (list intro) matfin (list intro))
 bass (list silence basst silence)
)

(def-length
 l-hand  '(1/8)
 bass '(1/8)
)

(def-duration ; sets up a sustaining pedal effect
l-hand '(1/2)
bass '(1/1)
)

(def-tonality
 l-hand (append chord1 tonal-2 chord1)
 bass (append tonal tonal tonal)
)

(def-velocity
 l-hand (append (list (gen-cresc 12 40 21))
                (symbol-divide 21 nil nil dynamicsz1) (list (gen-dim 40 12 21)))
 bass  '(36 38 40 25 32 40)
)

(def-zone
 l-hand (build-list '21/8 16)
 bass (append '(21/8) (list (make-zone (build-list '21/8 14))) '(21/8))
)

(def-tempo 60)

(compile-instrument-p "ccl;output:" "interaction-4"
 l-hand
 bass
)
```

```
;;;; Interactions for Piano (left hand) No.5

;;;  Slonimsky Patterns 27-31 (Symmetrical Interpolations of One Note)

;; functions

(defun rhy-mkr (lis)
"creates rhythms by counting components in a list"
(cond ((equal (length lis) 2) '(1/8 1/8))
       ((equal (length lis) 3) '(1/16 1/8 1/16))
        (t (build-list '1/16 (length lis)))))

(defun act-mkr (lis)
(cond ((equal (length lis) 2) '(120 80))
       ((equal (length lis) 3) '(120 64 70))
        (t '(100 64 70 64))))


;; material

(setq tonal (activate-tonality (chromatic c 6)))


(setq mat1 '(-m a a a a a m m m m m y m m m m a a a a a))

(setq mat1x '((-m a a a a) (a m m m m) (y m m m m) (m a a a a a)))

(setq sl31 '(a f g h)) ; Slonimsky Pattern 31


(setq sl31tr (gen-process  '(symbol-transpose x y) '(-24 -12 0 12) sl31))
 ; > (-y -u -s -q -m -i -g -e a e g i m q s u)

(setq in/out (append (list '(-m -g))
                     (gen-evolve 9 '(change-order nil
                                     (symbol-transpose (set-random -1 -6) x))
                                   (reverse sl31) :list)))

(setq mat
    (gen-evolve 6 '(change-order nil (symbol-transpose (set-random -2 2) x)) in/out :list))
```

```
(setq mat-1 (do-section (append '(=) (gen-template nil 2 1 10))
                        '(symbol-bundle (set-random 2 3) x) (nth 0 mat))
      mat-2 (do-section (append '(=) (gen-template nil 2 1 10))
                        '(symbol-bundle (set-random 2 3) x) (nth 1 mat))
      mat-3 (do-section (append '(=) (gen-template nil 2 1 10))
                        '(symbol-bundle (set-random 2 3) x) (nth 2 mat))
      mat-4 (do-section (append '(=) (gen-template nil 2 1 10))
                        '(symbol-bundle (set-random 2 3) x) (nth 3 mat))
      mat-5 (do-section (append '(=) (gen-template nil 2 1 10))
                        '(symbol-bundle (set-random 2 3) x) (nth 4 mat))
      mat-6 (do-section (append '(=) (gen-template nil 2 1 10))
                       '(symbol-bundle (set-random 2 3) x) (nth 5 mat))
      mat-7 (do-section (append '(=) (gen-template nil 2 1 10))
                        '(symbol-bundle (set-random 2 3) x) (nth 6 mat)))

(setq mat-all (append mat-1 mat-2 mat-3 mat-4 mat-5 mat-6 mat-7))

(setq solos-r
      (gen-collect 0.1 7 :list
                   '(length-insert '21/8 (vector-to-list (vector-round 60 240 (gen-sin 0.75 0.5 21))) nil)))

(seq :i/o (flatten in/out))
(setq solos-m (gen-collect 0.1 7 :list '(gen-seq :i/o 20)))
(seq :i/o :reset)

(setq solos-dyn
      (gen-collect 0.1 7 :list
                   '(vector-to-list (vector-round 110 30 (gen-sin 0.75 0.5 21)))))


(setq mat-1z (flatten (car (nthcdr 0 (symbol-divide 11 nil nil mat-all))))
      mat-2z (flatten (car (nthcdr 1 (symbol-divide 11 nil nil mat-all))))
      mat-3z (flatten (car (nthcdr 2 (symbol-divide 11 nil nil mat-all))))
      mat-4z (flatten (car (nthcdr 3 (symbol-divide 11 nil nil mat-all))))
      mat-5z (flatten (car (nthcdr 4 (symbol-divide 11 nil nil mat-all))))
      mat-6z (flatten (car (nthcdr 5 (symbol-divide 11 nil nil mat-all))))
      mat-7z (flatten (car (nthcdr 6 (symbol-divide 11 nil nil mat-all)))))

(setq matzi (list mat-1z mat-2z mat-3z mat-4z mat-5z mat-6z mat-7z))

;;(setq matzi (mapcar 'flatten (list mat-1 mat-2 mat-3 mat-4 mat-5 mat-6 mat-7)) ; simplification!
```

```
(setq mat-1x (flatten (car (nthcdr 0 (symbol-divide 11 nil nil (mapcar 'rhy-mkr mat-all)))))
      mat-2x (flatten (car (nthcdr 1 (symbol-divide 11 nil nil (mapcar 'rhy-mkr mat-all)))))
      mat-3x (flatten (car (nthcdr 2 (symbol-divide 11 nil nil (mapcar 'rhy-mkr mat-all)))))
      mat-4x (flatten (car (nthcdr 3 (symbol-divide 11 nil nil (mapcar 'rhy-mkr mat-all)))))
      mat-5x (flatten (car (nthcdr 4 (symbol-divide 11 nil nil (mapcar 'rhy-mkr mat-all)))))
      mat-6x (flatten (car (nthcdr 5 (symbol-divide 11 nil nil (mapcar 'rhy-mkr mat-all)))))
      mat-7x (flatten (car (nthcdr 6 (symbol-divide 11 nil nil (mapcar 'rhy-mkr mat-all))))))

(setq matxi (list mat-1x mat-2x mat-3x mat-4x mat-5x mat-6x mat-7x))

(setq mat-sym (cons (flatten in/out)
                    (symbol-interleave matzi solos-m)))

(setq mat-sym1 (reverse (cons (flatten (reverse in/out)) mat-sym)))

(setq dyn-in/out (gen-accent 60 '(2 4 4 4 4 4 4 4 4 4 4) (gen-cresc 64 110 42)))

(setq dyn-1 (flatten (mapcar 'act-mkr mat-1))
      dyn-2 (flatten (mapcar 'act-mkr mat-2))
      dyn-3 (flatten (mapcar 'act-mkr mat-3))
      dyn-4 (flatten (mapcar 'act-mkr mat-4))
      dyn-5 (flatten (mapcar 'act-mkr mat-5))
      dyn-6 (flatten (mapcar 'act-mkr mat-6))
      dyn-7 (flatten (mapcar 'act-mkr mat-7)))

(setq dyn-all  (list dyn-1 dyn-2 dyn-3 dyn-4 dyn-5 dyn-6 dyn-7))

(setq mat-dyn (cons dyn-in/out (symbol-interleave dyn-all solos-dyn)))
(setq mat-dyn1 (reverse (cons (flatten (reverse dyn-in/out)) mat-dyn)))

;; score

(def-symbol
 l-hand   (reverse (cons (flatten (reverse in/out))  (reverse mat-sym))))

(def-length
  l-hand  (append (list (build-list '1/16 42))
                  (symbol-interleave
                   matxi solos-r)
                   (list (build-list '1/16 42)))
)
```

```
(def-tonality
 l-hand  tonal
)

(def-velocity
 l-hand  (reverse (cons (flatten (reverse dyn-in/out))  (reverse mat-dyn)))
)

(def-zone
 l-hand (build-list '21/8 16) ;(make-zone (get-lengths-of 'l-hand))
)

(def-tempo 100)

(compile-instrument-p "ccl;output:" "interaction-5d"
 l-hand
)
```