Into the Green Inverted Dawn

For String Quartet

Nigel Morgan

Symbolic Composer score files, annotated by Phil Legard

Into the Green Inverted Dawn

For String Quartet

Nigel Morgan

The basic material used to generate the music is found in the words of a poem *Deep Sea Diver* by Robert Francis. The poem acts as a cantus or reference part to which all four instruments make reference. Sometimes this reference is exact and in unison. Other times this reference may be a variant and in a different octave.

There's a new function created for this piece called *gen-accompaniment.* This read the words / letters of the poem and invers the pitches around a triton. It then creates a rhythmic variation by simply erasing some pitches, like this: '(a b c d e) > '(m l k j i) > '(= l k = i).

The use of functions such as *gen-accompaniment*, which are able to create secondary parts, are distributed across the text by the use of template patterns ex. (x = = = x = x x = . . .). The letter x indicates where in the list of words *gen-accompaniment* might be mapped on to a particularly word. In this case the templates are organised to highlight verbs or nouns or connections and prepositions.

Other functions enable diminution and augmentation and the score is often sculpted by judicious deletion of phrases to enable changes of texture.

```
;; Functions Necessary for this Score

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

Very much like **gen-process-list** (see the annotations of *To the Dark Unseen*), gen-process-list enables two sets of data to be processed against one another. The difference is that **gen-process-values** manipulates numerical values, rather than lists of symbolic. So, `(gen-process-values '(gen-dim 80 35 x) '(5 4 3 2))` would give us decresecendos of 5, 4, 3 and 2 steps: `((80 69 58 46 35) (80 65 50 35) (80 58 35) (80 35))`.

```
(defun symbol-skip-i  (min max symbols &optional seed)
"allows length of symbol-list to control n of symbols skipped"
(symbol-select-skip (cond ((< max (length symbols)) (get-random min max))
       (t 0)) symbols seed))
```

Symbol-skip-i will introduce rests into a series of symbols using a random seed. So, `(symbol-skip-i 1 3 '(a b c d e f g))` will produce between 1 and 3 rests, yielding output such as: `(a b c d e = =)`, or `(a b c = = f g)`, or `(a = c d = f =)` etc.

```
(defun symbol-cut-i  (min max symbols &optional seed how)
"allows length of symbol-list to control n of symbols cut"
(if how
 how
(setq how (pick1 nil '(:end :rnd :start))))
(symbol-select-cut (cond ((< max (length symbols)) (get-random min max))
       (t 0)) symbols seed how))
```

This function operates in a similar manner to **symbol-skip-i**, except that all the rests are then sorted to either the beginning or end of the phrase. So, `(symbol-cut-i 1 3 '(a b c d e f g))` would cut between 1 and 3 symbols, before sorting the rests to the start or end depending on random choice, yielding `(= = a b c e g)`, `(a c d e g = =)` and so on.

```
(defun symbol-thin (min max lis &optional seed)
"calls either symbol-cut and symbol-skip"
(if seed
(init-rnd seed))
(let ((selection (pick-random '(:this :that ))))
         (case selection
           (:this
              (symbol-cut-i min      max lis ))
           (:that
              (symbol-skip-i min max      lis )))))
```

applying **symbol-skip-i** or **symbol-cut-i** to them by a random selection within the function, So, `(symbol-thin 1 3 '(a b c d e f g))` could potentially become `(a = c d = f =)` if processed with **symbol-skip-i**, or `(= = = a c d f)` if processed with **symbol-cut-i**.

```
(defun gen-accompaniment (lis &optional seed)
"creates a secondary part"
(symbol-mix (symbol-thin 1 (- (length lis) 1)
                      (symbol-shuffle (symbol-inversion 'g lis) seed) seed) lis))
```

Gen-accompaniment creates a randomised accompaniment to a phrase based on an inversion around symbol g. So, `(gen-accompaniment '(a b c d e f g))` would first generate an inverted version of the list: `(m l k j i h g)`. The symbols are shuffled, so our example may become: `(k j l m g i h)`. **Symbol-thin** is used to remove a number of random values between 1 and the total length, e.g. `(= j l = g = h)`. Finally the symbols are mixed with the original input, for example: `(a bj cl d eg f gh)`.

```
(defun make-rest (lis)
 (fill-rest lis '=))
```

Turn a list into rests, so `(make-rest '(a b c))` becomes `(= = =)`.

```
;; String Quartet Into the Green Inverted Dawn

(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
                 diver fall
                 and failing fight
                 your weed dense way
                 until you crawl
                 until you touch
                 weird water land
                 and stand
                 ))


(setq phrase-2 '(diver come up up through the green into the light
                 the sun the seen but in the clutch of your dripping hand
                 diver bring some uncouth thing
                 that we could sweargen-process-values
                 and would have sworn
                 was never born
                 or could ever be
                 anywhere
                 blaze on our sight
                 make us see
                 ))
```

Like *Blaze*, this piece is based on the symbolic values of the original poem *Deep Sea Diver* by Robert Francis.

This score realises the opening and closing sections of the piece. The material in **phrase-1** is fundamental to the first 47 bars of the piece. There is then an 'insert' (the code of which follows this part), after which **phrase-2** picks up from bar 102 to the end.

```
(setq p1-div (append (mapcar 'sort-descending (do-section :all
                '(mapcar 'symbol-melodize (mapcar 'list x)) phrase-1)) '((= = =)))
      p2-div (mapcar 'sort-ascending (do-section :all
                '(mapcar 'symbol-melodize (mapcar 'list x)) phrase-2)))

(mapcar 'compress p1-div)

; (vried og wond wond utrohhg the rngee vtrnieed wnda to the rkda usnnee to the vrnee yda the urned tnihg
; tsssrlea nda tspee peed tnheeba peed vried llfa nda nliigfa tihgf yuro weed sneed ywa utnli yuo wrlca
; utnli yuo utohc wried wtrea nlda nda tsnda ===)

(mapcar 'compress p2-div)

;(deirv cemo pu pu ghhortu eht eegnr inot eht ghilt eht nsu eht eens btu in eht cchltu fo oruy dgiinppr
; adhn deirv bginr emos chnotuu ghint ahtt ew cdlou aersw adn dlouw aehv norsw asw eenrv bnor or cdlou
; eerv be aeehnrwy abelz no oru ghist aekm su ees)
```

Here the words that constitute each 'phrase' (or section) of the poem are broken up into lists which are then sorted into either ascending or ascending order, so phrase-1 yields: `((v r i e d) (o g) (w o n d) ...` and so on. Compress is used to collapse the symbols into a continuous list of compounds such as `(vried og wond ...` and so on.

```
(setq p1-dyn (gen-process-values '(gen-dim 80 35 x) (mapcar 'length p1-div))
      p2-dyn (gen-process-values '(gen-cresc 40 95 x) (mapcar 'length p2-div)))
```

Each word, or phrase, in the piece has an accompanying diminuendo or crescendo. So, `(vried)` has the five-step diminuendo `(80 69 58 46 35)`.

```
(init-      rnd 0.137)
```

By setting the random seed here we can ensure that every random operation (such as **gen-accompaniment** and **symbol-thin**) will give the same results every time the code is run.

```
(setq sym-x-q (do-section '(x = x x = = x x x
                           = = x x = = x x
                           = = x x = x
                           x = x
                           x x
                           = x x
                           = x x x
                           = = x
                           = = x
                           x x x  x x
                           = )  '(gen-accompaniment x 0.137) p1-div))

(setq sym-x-r (do-section '(x = x x = = x x x
                           = = x x = = x x
                           = = x x = x
                           x = x
                           x x
                           = x x
                           = x x x
                           = = x
                           = = x
                           x x x  x x
                           = )  '(gen-accompaniment x 0.12) p1-div))

(setq  sym-y-q (do-section '(x = x x = = x = = x = x = x = = = x = = x x
                           x = x x x
                           = = = x
                           = = = x
                           = = x
                           = = x x
                           x
                           x x x x
                           x x x)  '(gen-accompaniment x 0.137) p2-div))
(setq  sym-y-r (do-section '(x = x x = = x = = x = x = x = = = x = = x x
                           x = x x x
                           = = = x
                           = = = x
                           = = x
                           = = x x
                           x
                           x x x x
                           x x x)  '(gen-accompaniment x 0.12) p2-div))
```

Here a template is generated which corresponds to the words in **phrase-1**. **Gen-accompaniment** is run through twice using two different random seeds in order to create a further pair of accompanying pitch symbols.

The same process is executed on **phrase-2**, below.

```
(setq rhy (build-list '(1/8) (length p1-div))
      rhy-x (p-replace nil '(1 4 5 7 9 10 13 14 17 18 21 24 28 30 31 35 36 38 39 44)
                       (list '(1/16)) rhy)

      rhy-a (build-list '(1/8) (length p2-div))
      rhy-ay (p-replace nil '(1 4 5 7 8 10 11 12 13 14 15 16 18 19
                              23 27 28 29  31 32 33 35 36  38 39 42  44 45 48 )
                        (list '(1/16)) rhy-a))
```

Values of 1/8 are the default length for all notes in **phrase-1**. P-replace is used to replace selected lengths with 1/16 values.

```
(setq sym-xv1 (do-section :all '(symbol-demix 2 x :sort) sym-x-q)
      sym-yv1 (do-section :all '(symbol-demix 2 x :sort) sym-y-q))

(setq sym-xv2 (do-section :all '(symbol-demix 2 x ) sym-x-r)
      sym-yv2 (do-section :all '(symbol-demix 2 x ) sym-y-r))
```

Symbol-demix is used to extract second value from the accompaniments generated from **phrase-1** and **phrase-2**. This becomes the material for the violin parts. So `(d e ji jr fv)` becomes `(= = i r v)`.

```
(setq sym-xvc (do-section (template-invert
                           '(x = x x = = x x x = = x x = = x x = = x x = x  x = x
                             x x = x x = x x = = x = = x  x x  x x x =))
                          '(make-rest x) p1-div)
      sym-yvc (do-section (template-invert
                           '(x = x x = = x = = x = x = x = = = x = = x x x = x x x = = = x
                             = = = x = = x = = x x  x  x x x x x x x))
                          '(make-rest x) p2-div))

(setq sym-xva (do-section  '(x = = x = = x x x = = = x = = = x = = x x = x x = x
                             x x = x x = x x x  = = x = = x  = = = = = =)
                           '(make-rest x)
                           (do-section :all '(symbol-demix 1 x :sort) sym-x-q))
      sym-yva (do-section  '(x = x x = = x = = = = = = = = = = = = = = x x = =
                             = = x = = = x = = = x = = =  = = x x  x  = = = = = = =)
                           '(make-rest x)
                           (do-section :all '(symbol-demix 1 x :sort) sym-y-q)))
```

Templates are used to create rests (x) for selected phrases. Note that **xva** and **xvc** ultimately correspond to **phrase-1** in the viola and cello, **yva** and **yvc** to **phrase-2** in the same. The cello plays material abstracted from the phrase contents themselves, while the viola plays material based on first values of the material generated to accompany **phrase-1** and **phrase-2**. So `(d e ji jr fv)` becomes `(d e j j f)`.

```
(setq zone-x (make-zone-list p1-div rhy-x)
      zone-y (make-zone-list p2-div rhy-ay))
```

```
(setq zone-z (zone-expand '(1 3 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 1 2
                            1 1 1 1
                            1 1 1
                            1 1 1
                            2 1 1
                            1 1 1

                            1 2 1 1 1 1 1 1 1 1
                            1 2 1 2 1 1 1 2 1 1 1 1
                            1 1 1 1 1
                            1 1 1 2
                            1 1 1 2
                            1 1 1
                            1 1 1 1
                            3
                            1 1 1 1
                            1 1 1
                            ) (append zone-x zone-y)))

;; score

(def-tonality
  v1 (activate-tonality (chromatic g 4))
  v2 (activate-tonality (chromatic g 4))
  va (activate-tonality (chromatic g 4))
  vc (activate-tonality (chromatic g 3))
)


(def-symbol
  v1 (mapcar 'find-change  (append sym-xv1  sym-yv1))
  v2 (mapcar 'find-anacrusis (append sym-xv2  sym-yv2))
  va (append sym-xva  sym-yva)
  vc (append sym-xvc  sym-yvc)
)
```

Selected zones are expanded, in order to create repeats (the symbol and pitch data is repeated to fill the zone lengths when the piece is compiled).

The two violins play material that was created using gen-accompaniment. This is further processed using **find-change** and **find-anacrucis**. The function of these is to add a rest when repeated symbols are found. **Find-change** turns the second symbol into a rest (so, `a a b` becomes `a = b`), while **find-anacrucis** turns the first symbol into a rest (so, `a a b` becomes `= a b`).

```
(def-length
  default (append rhy-x rhy-ay)
)


(def-velocity
  default (append p1-dyn p2-dyn)
)


(def-zone
  default zone-z
)

(def-channel
  v1 1
  v2 2
  va 3
  vc 4
)

(def-tempo 60)

(compile-instrument-p "ccl;output:" "Deep-Sea Quartet 1"
  v1
  v2
  va
  vc
)
```

```
;; String Quartet Into the Green Inverted Dawn – Middle Section

(defun calculate-positions (lis)
"calculates positions of positive integers in a list of attack values"
(cond ((symbolp (car lis))
       (e-position 'x lis))
      (t (e-position 'x
(string-to-symbol 'x (integer-to-string-1 lis))))))


(setq phrase-1/2 '(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
                   diver come up up through the green into the light
                   the sun the seen but in the clutch of your dripping hand
                   diver bring some uncouth thing))

(setq  p1/2-ord (do-section :all '(mapcar 'symbol-melodize (mapcar 'list x)) phrase-1/2))


(setq p1/2-dyn (gen-process-values '(gen-cresc-dim 40 90 x) (mapcar 'length  p1/2-ord)))

(init-rnd 0.137)

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

(setq sym-x-q1 (do-section template-1 '(gen-accompaniment x 0.137)  p1/2-ord)
      sym-x-r1 (do-section template-1 '(gen-accompaniment x 0.12)   p1/2-ord))


(setq rhy1 (build-list '(1/16) (length p1/2-ord))
      rhy1-x (p-replace nil (calculate-positions (template-invert template-1))
                        (list '(1/8)) rhy1))

(setq sym-xv3 (do-section :all '(symbol-demix 2 x :sort) sym-x-q1)
      sym-yv3 (do-section :all '(symbol-demix 2 x :sort) sym-x-r1))
```

```
(setq zone-w (make-zone-list p1/2-ord  rhy1-x))

;; score

(def-tonality
  v1 (activate-tonality (chromatic g 4))
  v2 (activate-tonality (chromatic g 4))
  va (activate-tonality (chromatic g 4))
  vc (activate-tonality (chromatic g 3))
)

(def-symbol
  v1 (mapcar 'find-anacrusis  sym-xv3)
  v2  (mapcar 'find-change sym-yv3)
  va  sym1-xva
  vc  sym1-xvc
)

(def-length
  default rhy1-x
)


(def-velocity
default  p1/2-dyn
)


(def-zone
default zone-w
; (5/16 1/4 1/4 1/4 7/8 3/8 5/16 1/2 1/4 1/4 3/8 1/4 3/8 1/4 3/8 5/16 3/16 3/8 5/8 5/16 1/2 3/8
; 5/16 1/4 7/8 1/4 5/16 1/2 1/8 1/8 7/8 3/8 5/16 1/2 3/8 5/16 3/8 3/16 3/8 1/4 3/8 1/4 3/8 3/8 1/4
; 1/2 1/2 1/4 5/16 5/8 1/4 7/16 5/16)
)

(def-channel
  v1 1
  v2 2
  va 3
  vc 4
)
```

```
(def-tempo 60)

(compile-instrument-p "ccl;output:" "Deep-Sea Quartet 1x"
  v1
  v2
  va
  vc
)
```