```lisp
;;;; arciorgano code for "Tochter aus Elysium"
;;;; theater play by Joëdine)
;;;;    - (in-package :scratch)
;;;;    - (rt-start)
;;;; 3. load definitions.lisp
;;;; 4. compile rest of elysium2.lisp

;;;; 5. in REPL:
;;;;    - (cern-init)
;;;;    - (motor 1)

;;;; live-commands: (burst-first), (burst n) [0 <= n < 7], (cern),
;;;; (next-quarta), (next-model), (next-genus), (next-limit),
;;;; (speed-random-range a b) [a, b: BPM, (< a b)], (panic),
;;;; (lamento-panic)


(require 'incudine)
(in-package :scratch)

(rt-start)

(defparameter *osc-out* (osc:open :port 5900 :direction :output))


(load "~/Coding/pd/arciorgano-pd-sender/lisp/definitions.lisp")



(defun parse-tetrachord (origin-name octave tetrachord)
  (let ((start-pitch (+ (* 31 octave) (name->pitch origin-name))))
    (labels ((rec (val lst)
               (unless (null lst)
                 (let ((new-val (- val (interval->pitch (car lst)))))
                   (cons new-val (rec new-val (rest lst)))))))
      (cons start-pitch (rec start-pitch tetrachord)))))

(defun key-off (index)
  (when (< 0 index 147)
    (format t "~a:off " index)
    (osc:message *osc-out* "/incudine-bridge" "ii" index 0)))

(defun key-on (index &optional duration-in-sec)
  (when (< 0 index 147)
    (osc:message *osc-out* "/incudine-bridge" "ii" index 1)
    (format t "~a:on " index)
    (when duration-in-sec
      (at (+ (now) #[duration-in-sec sec]) #'key-off index))))



(defun play-pitch (pitch duration)
  (key-on (pitch->key pitch) duration))



(defparameter *score* '(() () ()))

(defparameter *multiplexer* nil)

(defun multi (val)
  (if (zerop val)
      (setf *multiplexer* nil)
      (setf *multiplexer* val)))

(defun play-latest-keyframe (score duration)
  (mapc (lambda (voice)
          (play-pitch (first voice) duration)
          (when *multiplexer*
            (let ((bottom (- 0 (floor *multiplexer* 2))))
              (loop for i from bottom to (+ bottom *multiplexer*) do
                (play-pitch (+ (first voice) (* i 31)) duration)))))
```

```lisp
        score))

(defun make-harmony-server (interval-list)
  (let ((current-list interval-list))
    #'(lambda (&optional selection)
        (cond ((null current-list) (setf current-list interval-list))
              (selection (setf current-list (remove selection current-list)))
              (t current-list)))))

(defun find-voiceleading (origin last-note consonance-options)
  (let ((choice (first (sort (mapcar (lambda (interval)
                                        (cons interval (+ origin (interval->pitch interval))))
                                      (funcall consonance-options))
                              #'< :key (lambda (pitch-candidate)
                                          (abs (- last-note (cdr pitch-candidate))))))))
    (funcall consonance-options (car choice))
    (cdr choice)))

(defun write-to-score (value-list score)
  (cond ((null score) nil)
        (t (cons (cons (car value-list) (car score))
                 (write-to-score (rest value-list) (rest score))))))

(defun compose-keyframe (tetrachord-position origin model-position score &optional start-harmony)
  (let ((current-pitch (nth tetrachord-position (parse-tetrachord
                                                  (car origin)
                                                  (cdr origin)
                                                  (funcall *tetrachord-generator*)))))

    (when start-harmony
      (setf score (write-to-score (cons current-pitch
                                        (mapcar (lambda (interval)
                                                  (+ current-pitch (interval->pitch interval)))
                                                start-harmony))
                                  score)))
    (let ((get-harmony-options (make-harmony-server (nth model-position (funcall *model-generator
*)))))
      (write-to-score (cons current-pitch
                            (mapcar (lambda (voice)
                                      (find-voiceleading current-pitch
                                                         (first voice)
                                                         get-harmony-options))
                                    (rest score)))
                      score))))

(defparameter *playing* t)

(defun my-start () (setf *playing* t))
(defun my-stop () (setf *playing* nil))

(defun bpm->sec (bpm)
  (/ 60.0 bpm))



;; performance override, safeguard for good tempo

(defparameter *duration-generator*
    #'(lambda (&key (reset nil) (factor nil) (rand nil) (rand-range nil))
        (declare (ignore reset factor rand rand-range))
        (bpm->sec 42)))


;; performance override, safeguard for ninfa ostinato model

(defparameter *model-generator*
  #'(lambda (&optional next)
      (declare (ignore next))
      '((terza-minore quinta ottava)
        (terza-maggiore sesta-maggiore ottava)
        (terza-maggiore sesta-maggiore)
        (terza-maggiore quinta ottava))))
```

```lisp
;; simplified duration generator, in case original version creates damaging behaviour

(defparameter *duration-generator*
  (let ((internal-speed 45))
    #'(lambda (&key (reset nil) (factor nil) (rand nil) (rand-range nil))
        (declare (ignore factor rand rand-range))
        (when reset (setf internal-speed reset))
        (bpm->sec internal-speed))))



;; complex and risky

(defparameter *duration-generator*
  (let ((counter (bpm->sec 45))
        (internal-factor 1)
        (random-on 0)
        (random-range '(1/10 . 1)))
    #'(lambda (&key (reset nil) (factor nil) (rand nil) (rand-range nil))
        (when reset (setf counter (bpm->sec reset)))
        (when factor (setf internal-factor factor))
        (when rand (setf random-on rand))
        (when (consp rand-range) (setf random-range rand-range))
        (let ((result (cond ((zerop random-on)
                             (setf counter (* counter internal-factor))
                             (if (not (= internal-factor 1))
                                 (if (< counter (car random-range))
                                     (car random-range)
                                     (if (> counter (cdr random-range))
                                         (cdr random-range)
                                         counter))
                                 counter))
                            (t (+ (car random-range) (random (* 1.0 (cdr random-range))))))))
          (cond ((< result 1/20) (format t "~&min correction") 1/20)
                ((> result 10) (format t "~&max correction") 10)
                (t result))))))

(defun loop-tetrachord (position tetrachord origin model score)
  (when *playing*
    (let ((duration (funcall *duration-generator*)))
      (cond ((>= position 4) (loop-tetrachord 0 tetrachord origin model score))
            (t (play-latest-keyframe score duration)
               (at (+ (now) #[duration sec])
                   #'loop-tetrachord
                   (1+ position)
                   tetrachord
                   origin
                   model
                   (compose-keyframe position origin position score)))))))




(defun speed-reset (&optional (val 45))
  (funcall *duration-generator* :reset val))

(defun speed-factor (val)
  (funcall *duration-generator* :factor val))

(defun speed-random (toggle)
  (funcall *duration-generator* :rand toggle))

(defun speed-random-range (min-bpm max-bpm)
  (let ((rand-range (cons (bpm->sec max-bpm) (bpm->sec min-bpm))))
    (funcall *duration-generator* :rand-range rand-range)
    rand-range))
```

```lisp
(defun play-loop ()
  (loop-tetrachord 0
                   *tetrachord-generator*
                   '(g . 1)
                   *model-generator*
                   '((50) (58) (68))))

(defun panic ()
  (my-stop)
  (loop for i from 0 to 146 do
    (key-off i)))



(defun cern ()
  (my-start)
  (speed-factor 1)
  (speed-reset 1/10)
  (speed-random 0.6)
  (play-loop))



(defun cern-init ()
  (next-genus 'enarmonico)
  (next-quality 'dissonant)
  (next-quarta 'prima)
  (multi 4)
  (speed-random 1)
  (speed-random-range 100 600))

(defun lamento-init ()
  (next-genus 'diatonico)
  (next-quality 'consonant)
  (next-quarta 'seconda)
  (multi 0)
  (speed-reset 50)
  (speed-random 0))



;; (defparameter *oscin* (osc:open :port 5800 :host "127.0.0.1" :protocol :udp :direction :input)
)

;; (recv-start *oscin*)

;; (make-osc-responder *oscin* "/incudine/genere" "i"
;;                              (lambda (genus)
;;                             (msg warn "~a" genus)))


;; (make-osc-responder *oscin*
;;                     "/incudine/timer/range" "ii"
;;                        (lambda (min max)
;;                       (speed-random-range (cons (bpm->sec min)
;;                                                  (bpm->sec max)))))

;; (make-osc-responder *oscin* "/incudine/timer/factor" "f"
;;                              (lambda (factor)
;;                             (speed-factor factor)))

;; (make-osc-responder *oscin* "/incudine/timer/rand" "i"
;;                              (lambda (toggle)
;;                             (speed-random toggle)))

;; (make-osc-responder *oscin* "/incudine/multi" "i"
;;                              (lambda (id)
;;                             (multi id)))
```

```
(defun swipe (&key (start 1) (end 146) (delta 1/4) (duration 1/4))
  (cond ((>= start end) nil)
        (t (key-on start duration)
           (at (+ (now) #[delta sec])
               #'swipe
               :start (1+ start)
               :end end
               :delta delta
               :duration duration))))


(defun burst-swipe ()
  (swipe :start 50 :end 70 :delta 1/10 :duration 1/5)
  (swipe :start 55 :end 80 :delta 2/10 :duration 2/5)
  (swipe :start 20 :end 30 :delta 1/2 :duration 1/3)
  (swipe :start 100 :end 130 :delta 1/10 :duration 2/5)
  (swipe :start 110 :end 146 :delta 1/3 :duration 1/3))


(defun burst-random (&key (duration 10) (density 100))
  (loop repeat density do
    (at (+ (now) #[(random (* 1.0 duration)) s])
        #'key-on (random 146) (random (* 0.5 duration)))))



(defun motor (toggle)
    (osc:message *osc-out* "/incudine-bridge-motor" "i" toggle))

;; not to be used in performance, risk of system breakdown
(defun light (toggle)
    (osc:message *osc-out* "/incudine-bridge-light" "i" toggle))

(defun burst (id)
  (case id
    (0 (burst-random :duration 1 :density 1000))
    (1 (burst-random :duration 3 :density 5000))
    (2 (burst-random :duration 5 :density 10000))
    (3 (burst-random :duration 8 :density 15000))
    (4 (burst-random :duration 10 :density 30000))
    (5 (burst-random :duration 30 :density 80000))))


;; redundant, light is not triggered remotely anymore
(defun burst-first ()
  (light 1)
  (at (+ (now) #[0.5 s]) #'burst 0))

(defun lamento-panic ()
  (panic)
  (motor 0))
```