Ludger Brümmer

The Gates of H.

Score

1. The clm instrument

1.1. loops6

```
(definstrument (loops :exp-env nil)
    (start-time
     duration
     amp
     srt
     filename
    srt-env
     srt-scaler
     start-position
     start-positiont
     start-direction
     direction-switch
     distance
     pan-env
     amp-env
     reverb-amount
&optional (degree -1))
 (let* ((senv (make-env :envelope srt-env
                :scaler srt-scaler
                :offset 0.0
                :start-time start-time
                :duration duration))
     (beg (floor (* start-time sampling-rate)))
     (end (+ beg (floor (* duration sampling-rate))))
     (f (open-input filename))
     (loc (make-locsig :distance distance
            :degree degree
            :revscale reverb-amount))
    (amp-en (make-env :envelope amp-env
            :scaler amp
            :offset 0.0
            :start-time start-time
            :duration duration))
    (pan-en (make-env :envelope pan-env
            :scaler 1.0
            :offset 0.0
            :start-time start-time
            :duration duration))
    (panning 0)
    (length  (clm-get-samples f))
    (length (- length 20))
    (starting   (if  (= -1 start-positiont)
                     (/ start-position sampling-rate)
                      start-positiont))
    (starting2 (if  (=  0 direction-switch)
                     starting
                     (+ duration starting)))
    (src-gen (make-src :file f :srate srt :start-time starting2)))
  (setf (rdin-inc (sr-rd src-gen))
        start-direction)
  (print (list "start"  start-time "start-pos" (if (= -1 start-positiont) start-
position start-positiont)))
(Run
   (loop for i from beg to end do
    (let* ((position (rdin-i (sr-rd src-gen))))
     (when (or (>= position length)
     (<= position 10))
     (setf (rdin-inc (sr-rd src-gen)) (* start-direction -1)))
    (when  (minusp degree)
    (setf panning (env pan-en))
    (setf (locs-ascl loc)  panning)
    (setf (locs-bscl loc) (- 1 panning)))
```

```
    (locsig loc i (* (env amp-en) (src src-gen (env senv))))  )))
  (close-input f)))

(defpart loops (clm-part)
 (name  start-time duration amp srt filename srt-env srt-scalerstart-position
start-positiont start-direction direction-switch distance pan-env amp-env
reverb-amount &optional degree )
(name start-time duration amp srt filename srt-envsrt-scalerstart-position
start-positiont start-direction direction-switch distance pan-env amp-env
reverb-amount &optional degree ))
```

1.2. loops7

```
(definstrument (loops :exp-env nil)
    (start-time
     duration
     amp
     srt
     filename
     srt-env
     srt-scaler
     start-position
     start-positiont
     start-direction
     direction-switch
     amp-env
     reverb-amount
     distance
&optional (degree 1)  ;;; if positiv degree or distance, values are taken,
otherwise the                ;;; pan-env or distance-env
            (pan-env '(0 0 100 0))
            (distance-env '(0 0 100 0))
            (distance-scaler 1))
 (let* ((senv (make-env :envelope srt-env
                        :scaler srt-scaler
                        :offset 0.0
                        :start-time start-time
                        :duration duration))
(dis-env (make-env :envelope distance-env
                        :scaler distance-scaler
                        :offset 1.0
                        :start-time start-time
                        :duration duration))
       (beg (floor (* start-time sampling-rate)))
       (end (+ beg (floor (* duration sampling-rate))))
       (f (open-input filename))
       (loc (make-locsig :distance distance
                         :degree degree
                         :revscale reverb-amount))
      (amp-en (make-env :envelope amp-env
                        :scaler amp
                        :offset 0.0
                        :start-time start-time
                        :duration duration))
     (pan-en (make-env :envelope pan-env
                        :scaler .999
                        :offset 0.0001
                        :start-time start-time
                        :duration duration))
       (length  (clm-get-samples f))
       (length (- length 20))
       (starting   (if  (= -1 start-positiont)
                                (/ start-position sampling-rate)
                                start-positiont))
     (starting2 (if  (=  0 direction-switch)
                                starting
                                (+ duration starting)))
```

```
    (src-gen (make-src :file f :srate srt :start-time starting2)))
    (setf (rdin-inc (sr-rd src-gen))
          start-direction)
  (print (list "start"  start-time "start-pos" (if (= -1 start-positiont) start-
position start-positiont)))
(Run
    (loop for i from beg to end do
     (let* ((position (rdin-i (sr-rd src-gen))))
      (when (or (>= position length)
       (<= position 10))
       (setf (rdin-inc (sr-rd src-gen)) (* start-direction -1)))
      (when  (or (minusp distance)(minusp degree))
  (let* ((panning (if (minusp degree) (env pan-en) degree))
         (d-env (if (minusp distance) (env dis-env) distance))
         (dist (/ 1.0 (max d-env 1.0)))
         (sdist (/ 1.0 (sqrt (max d-env 1.0)))))
         (setf (locs-rscl loc)  (* reverb-amount sdist))
         (setf (locs-ascl loc)  (* dist (- 1.0 panning)))
         (setf (locs-bscl loc)  (* dist  panning))))
     (locsig loc i (* (env amp-en) (src src-gen (env senv))) )))
  (close-input f)))

(defpart loops (clm-part)
 (name  start-time  duration amp srt filename srt-env srt-scaler start-
position start-positiont start-direction direction-switch amp-env
reverb-amount distance &optional degree pan-env distance-env
distance-scaler)
(name  start-time duration amp srt filename srt-env srt-scaler start-position
start-positiont start-direction direction-switch amp-env reverb-amount
distance &optional degree pan-env distance-env distance-scaler))
```

1.3. loops8

```
(definstrument (loops :exp-env nil)
    (start-time
     duration
     amp
     srt
     filename
     srt-env
     srt-scaler
     start-position
     start-positiont
     start-direction
     direction-switch
     amp-env
     reverb-amount
     distance
&optional (degree 1)  ;;; if positiv degree or distance, values are taken,
otherwise the                ;;; pan-env or distance-env
            (pan-env '(0 0 100 0))
            (distance-env '(0 0 100 0))
            (distance-scaler 1))
(let ((f (open-input filename)))
     (unwind-protect
  (let* ((senv (make-env :envelope srt-env
                :scaler srt-scaler
                :offset 0.0
                :start-time start-time
                :duration duration))
(dis-env (make-env :envelope distance-env
                :scaler distance-scaler
                :offset 1.0
                :start-time start-time
                :duration duration))
```

```
  (beg (floor (* start-time sampling-rate)))
  (end (+ beg (floor (* duration sampling-rate))))
  (loc (make-locsig :distance distance
        :degree degree
        :revscale reverb-amount))
  (amp-en (make-env :envelope amp-env
        :scaler amp
        :offset 0.0
        :start-time start-time
        :duration duration))
  (pan-en (make-env :envelope pan-env
        :scaler .999
        :offset 0.0001
        :start-time start-time
        :duration duration))
  (length  (clm-get-samples f))
  (length (- length 20))
  (starting   (if  (= -1 start-positiont)
                   (/ start-position sampling-rate)
                   start-positiont))
  (starting2 (if  (=  0 direction-switch)
                  starting
                  (+ duration starting)))
  (src-gen (make-src :file f :srate srt :start-time starting2)))
  (setf (rdin-inc (sr-rd src-gen))
        start-direction)
 (print (list "start"  start-time "start-pos" (if (= -1 start-positiont) start-
position start-positiont)))
(Run
   (loop for i from beg to end do
    (let* ((position (rdin-i (sr-rd src-gen))))
    (when (or (>= position length)
    (<= position 10))
    (setf (rdin-inc (sr-rd src-gen)) (* start-direction -1)))
    (when  (or (minusp distance)(minusp degree))
 (let* ((panning (if (minusp degree) (env pan-en) degree))
        (d-env (if (minusp distance) (env dis-env) distance))
        (dist (/ 1.0 (max d-env 1.0)))
        (sdist (/ 1.0 (sqrt (max d-env 1.0)))))
        (setf (locs-rscl loc)   (* reverb-amount sdist))
        (setf (locs-ascl loc)  (* dist (- 1.0 panning)))
        (setf (locs-bscl loc)  (* dist  panning))))
    (locsig loc i (* (env amp-en) (src src-gen (env senv))))  ))))
  (progn (close-input f)) ) ))

(defpart loops (clm-part)
 (name  time duration amp srt filename srt-env srt-scalerstart-position
start-positiont start-direction direction-switch amp-env reverb-amount
distance &optional degree pan-env distance-env distance-scaler)
(duration amp srt filename srt-env  srt-scaler start-position start-positiont
start-direction direction-switch amp-env reverb-amount distance
&optional degreepan-env distance-env distance-scaler))
```

1.4. loops9

```
# | start-position   :in samples
start-positiont : default start-positiont in time, if -1 than start-position
start-direction : +1 forward -1 backwards
(direction-switch  0): 0 normal, other than 0:(+ start-time duration) is the
      begin
(degree 1): if positiv degree or distance, values are taken, otherwise the
      pan-env or distance-env | #

(definstrument (loops :exp-env nil)
    (start-time
      duration
      amp
```