```
<CsoundSynthesizer>
<CsOptions>
-m128
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 128
nchnls = 2
0dbfs = 1
seed 0


;===============================================================================;
;===============================================================================;
;===============================================================================;
;                                                                               ;
;                              Joachim Heintz                                   ;
;                                                                               ;
;===============================================================================;
;                                                                               ;
;                              EIN SEETEUFEL                                    ;
;                 Musik zu einer Lesung von Yoko Tawada                         ;
;                                                                               ;
;===============================================================================;
;                                                                               ;
;              Aufführung in der Akademie der Künste Berlin                     ;
;                              25 Mai 2018                                      ;
;                                                                               ;
;                                                                               ;
;                                                                               ;
;                www.joachimheintz.de/ein-seeteufel.html                        ;
;                                                                               ;
;===============================================================================;
;===============================================================================;
;===============================================================================;




/*****************************************************************/
/******************** GLOBALE VARIABLEN ************************/
/*****************************************************************/


//INPUT / OUTPUT

//mikrofon input
giInChn = 1

//output
giOutChnL = 1
giOutChnR = 2


//LAUTSTÄRKE

//globale lautstärke (db)
gkGlobVolDb init 0

//lautstärke des raschelns (db) extern
gk_R_VolDbLive init 0

//lautstärke rascheln intern (db)
gkRaschVolDbIntern init -18

//lautstärke text schicht A (db) extern
gk_A_VolDbLive init 0

//lautstärke text schicht B (db) extern
gk_B_VolDbLive init 0

//lautstärke text schicht C (db) extern
gk_C_VolDbLive init 0

//lautstärke arpeggiato intern (db)
gkArpeggVolDbIntern init 0

//yoko dummy (playback)
gkDummy init 0
gkYokoDummyVolDb init 0


//RASCHELN

//tempo für die raschel rhythmen
gkRaschRtmTempo init 80

//auslassungen bei den raschel rhythmen
//1 = jede note wird gespielt, 2 = durchschnittlich jede zweite, etc
gkRaschRtmAuslassungen init 1

//wackeln der raschel rhythmen (0-100%)
gkRaschRtmWackeln init 50


//KNUTH

//knuth trigger impulse
gkKnuthTrigImp init 0

//knuth minimaler abstand zwischen zwei trigger impulsen (sec)
gkKnuthMinTimIntv init 1/7
```

```
//knuth threshold (db)
gkKnuthThreshDb init -30


//HALL

//hall wet-dry mix (0=dry 1=wet)
gkRaschHallWdMix init .2
gkArpeggHallWdMix init 0

//hall roomsize (für reverbsc)
gkRaschHallRoomSiz init .8
gkArpeggHallRoomSiz init .5

//ANDERES

//input
gaLiveIn init 0

//tastatur
gkKey init 0
gkKeyReturn init 0

//maus als ersatz für tastatur
gkMausReturn init 0

//T6 cue
gkT6Cue init 0

//T8 index durch die tonhöhen schicht eins
gkT8_Pitches init 0

//Z5 teilton lautstärken
gkZ5_db_TT1 init 0
gkZ5_db_TT2 init 0
gkZ5_db_TT3 init 0
gkZ5_db_TT4 init 0

//wenn T1 fertig ist (für die piekser in Z2)
gkT1_End init 0


strset 1, "EinRaschler"
strset 2, "EinRaschlerZ5"


/*****************************************************************/
/******************** WICHTIGE LAUTSTÄRKEN ***********************/
/*****************************************************************/

//mittlere lautstärken beim rascheln
giDb_Z1 = -20
giDb_Z2 = -20
giDb_Z3 = -20
giDb_Z4 = -20
giDb_Z5 = -25
giDb_Z6 = -30
giDb_Z7 = -25
giDb_Z8 = -25


/*****************************************************************/
/******************** SOFTWARE CHANNELS **************************/
/*****************************************************************/

//audio submix rascheln und textschichten
chn_a "collect_rasch_L", 3
chn_a "collect_rasch_R", 3
chn_a "collect_text_A_L", 3
chn_a "collect_text_A_R", 3
chn_a "collect_text_B_L", 3
chn_a "collect_text_B_R", 3
chn_a "collect_text_C_L", 3
chn_a "collect_text_C_R", 3

//input von den widgets
chn_k "mic_in_gain_db", 1
chn_k "knuth_thresh_db", 1
chn_k "glob_vol_db", 1
chn_k "R_vol_db", 1
chn_k "A_vol_db", 1
chn_k "B_vol_db", 1
chn_k "C_vol_db", 1
chn_k "yoko_dummy_vol_db", 1
chn_k "mausreturn", 1

//output zu den widgets
chn_k "mic_in", 2
chn_k "mic_in_over", 2
chn_k "outL", 2
chn_k "outL_over", 2
chn_k "outR", 2
chn_k "outR_over", 2
chn_k "knuth_min_interv_tim", 2
chn_k "knuth_trig_imp", 2
chn_k "ABC_vol_db_intern", 2
chn_k "R_vol_db_intern", 2
chn_k "R_rtm_tempo", 2
chn_k "R_rtm_wackeln", 2
chn_k "R_rtm_auslass", 2
chn_k "R_hall_roomsiz", 2
chn_k "R_hall_wdmix", 2
chn_k "ABC_hall_roomsiz", 2
chn_k "ABC_hall_wdmix", 2
```

```
chn_k "T6_cue", 2
chn_k "dieser_teil", 2
chn_S "aktive_instrumente", 2
chn_S "aktive_instrumente_2", 2
chn_k "T2_seq_count", 2
chn_k "R_outL", 2
chn_k "R_outL_over", 2
chn_k "R_outR", 2
chn_k "R_outR_over", 2
chn_k "A_outL", 2
chn_k "A_outL_over", 2
chn_k "A_outR", 2
chn_k "A_outR_over", 2
chn_k "B_outL", 2
chn_k "B_outL_over", 2
chn_k "B_outR", 2
chn_k "B_outR_over", 2
chn_k "C_outL", 2
chn_k "C_outL_over", 2
chn_k "C_outR", 2
chn_k "C_outR_over", 2

//input und output
chn_k "next_teil", 3
chn_k "dummy", 3
chn_k "zeig_aktive", 3
chn_k "zeig_aktive_2", 3



/*****************************************************************/
/******************** UDOS UND ÄHNLICHES ************************/
/*****************************************************************/


giHalfSine ftgen 0, 0, 8193, 9, 1/2, 1, 0

opcode StepIncr, k, kkkk
 kMinVal, kMaxVal, kNumSteps, kThisStep xin
 kOut = ((kMaxVal-kMinVal) / kNumSteps) * kThisStep + kMinVal
 xout kOut
endop

opcode Drei, k, kkkkkk
 ;jetzt in stationen gezählt statt in schritten: 1, 2, 3, ...
 kStart, kMid, kEnd, kStationen, kMittelStation, kDieseStation xin
 if kDieseStation <= kMittelStation then
  kOut StepIncr kStart, kMid, kMittelStation-1, kDieseStation-1
 else
  kOut StepIncr kMid, kEnd, kStationen-kMittelStation, kDieseStation-kMittelStation
 endif
 xout kOut
endop

opcode Meter, 0, SSak
 S_chan_sig, S_chan_over, aSig, kTrig   xin
 iDbRange = 60 ;shows 60 dB
 iHoldTim = 1 ;seconds to "hold the red light"
 kOn init 0
 kTim init 0
 kStart init 0
 kEnd init 0
 kMax max_k aSig, kTrig, 1
 if kTrig == 1 then
  chnset (iDbRange + dbfsamp(kMax)) / iDbRange, S_chan_sig
  if kOn == 0 && kMax > 1 then
   kTim = 0
   kEnd = iHoldTim
   chnset k(1), S_chan_over
   kOn = 1
  endif
  if kOn == 1 && kTim > kEnd then
   chnset k(0), S_chan_over
   kOn =        0
  endif
 endif
 kTim += ksmps/sr
endop

opcode Laufen, k, kkkk
 kMidiPrevNote, kCentDir, kMidiLimit, kCentMaxDev xin
 kNote = kMidiPrevNote + kCentDir/100 + rand:k(kCentMaxDev/100,2,1)
 xout kNote
endop

opcode ArrPermRnd, i[], i[]j
 iInArr[], iN xin
 iLen       =          lenarray(iInArr)
 ;get output length
 iN = (iN == -1) ? iLen : iN
 ;create out array and set index
 iOutArr[]  init       iN
 iIndx      =          0
 ;for iN elements:
 until iIndx == iN do
 ;get one random element and put it in iOutArr
  iRndIndx   random     0, iLen-.0001
  iRndIndx   =          int(iRndIndx)
  iOutArr[iIndx] =      iInArr[iRndIndx]
  ;shift the elements after this one to the left
  until iRndIndx == iLen-1 do
   iInArr[iRndIndx] = iInArr[iRndIndx+1]
   iRndIndx   +=         1
  od
```

```
  ;reset iLen and increase counter
  iLen       -=         1
  iIndx      +=         1
  od
             xout       iOutArr
endop

opcode ArrPermRnd, k[], k[]j
 kInArr[], iN xin
 iLen       =          lenarray(kInArr)
 iN = (iN == -1) ? iLen : iN
 kOutArr[]  init       iN
 kIndx      =          0
 kLen = iLen
 ;for kN elements:
 until kIndx == iN do
  ;get one random element and put it in kOutArr
  kRndIndx   random     0, kLen-.0001
  kRndIndx   =          int(kRndIndx)
  kOutArr[kIndx] =      kInArr[kRndIndx]
  ;shift the elements after this one to the left
  until kRndIndx == kLen-1 do
   kInArr[kRndIndx] = kInArr[kRndIndx+1]
   kRndIndx   +=         1
  od
 ;reset kLen and increase counter
  kLen       -=         1
  kIndx      +=         1
  od
             xout       kOutArr
endop

opcode PrtArr1k, 0, k[]POVVO
 kArr[], ktrig, kstart, kend, kprec, kppr xin
 kprint     init       0
 kndx       init       0
 if ktrig > 0 then
  kppr       =          (kppr == 0 ? 10 : kppr)
  kend       =          (kend == -1 || kend == .5 ? lenarray:k(kArr) : kend)
  kprec      =          (kprec == -1 || kprec == .5 ? 3 : kprec)
  kndx       =          kstart
  Sformat    sprintfk   "%%%d.%df, ", kprec+3, kprec
  Sdump      sprintfk   "%s", "["
  loop:
  Snew       sprintfk   Sformat, kArr[kndx]
  Sdump      strcatk    Sdump, Snew
  kmod       =          (kndx+1-kstart) % kppr
  if kmod == 0 && kndx != kend-1 then
             printf     "%s\n", kprint+1, Sdump
  Sdump      strcpyk    " "
  endif
  kprint     =          kprint + 1
             loop_lt    kndx, 1, kend, loop
  klen       strlenk    Sdump
  Slast      strsubk    Sdump, 0, klen-2
             printf     "%s]\n", kprint+1, Slast
 endif
endop

opcode ArrRndEl, k, k[]OV
 kInArr[], kStart, kEnd xin
 kLen lenarray (kInArr)
 kEnd = (kEnd == 0.5) ? kLen-1 : kEnd
 kElIndx random kStart, kEnd+0.999
 kEl = kInArr[int(kElIndx)]
 xout kEl
endop

opcode ArrRndEl, i, i[]oj
 iInArr[], iStart, iEnd xin
 iLen lenarray (iInArr)
 iEnd = (iEnd == -1) ? iLen-1 : iEnd
 iElIndx random iStart, iEnd+0.999
 iEl = iInArr[int(iElIndx)]
 xout iEl
endop

opcode ArrPermRnd, k[], k[]j
 kInArr[], iN xin
 iLen       =          lenarray(kInArr)
 iN = (iN == -1) ? iLen : iN
 kOutArr[]  init       iN
 kIndx      =          0
 kLen = iLen
 ;for kN elements:
 until kIndx == iN do
  ;get one random element and put it in kOutArr
  kRndIndx   random     0, kLen-.0001
  kRndIndx   =          int(kRndIndx)
  kOutArr[kIndx] =      kInArr[kRndIndx]
  ;shift the elements after this one to the left
  until kRndIndx == kLen-1 do
   kInArr[kRndIndx] = kInArr[kRndIndx+1]
   kRndIndx   +=         1
  od
  ;reset kLen and increase counter
  kLen       -=         1
  kIndx      +=         1
 od
             xout       kOutArr
endop

opcode ArrPermRnd2, k[], k[]kk
 kInArr[], kStart, kEnd xin
 kLen lenarray kInArr
```

```
 kOutArr[] = kInArr
 kIndx = kStart
 kLast = kEnd
 until kIndx > kLast do
  kRndIndx random kStart, kEnd+.9999
  kRndIndx = int(kRndIndx)
  kOutArr[kIndx] = kInArr[kRndIndx]
  until kRndIndx >= kEnd do
   kInArr[kRndIndx] = kInArr[kRndIndx+1]
   kRndIndx += 1
  od
  kIndx += 1
  kEnd -= 1
 od
 xout kOutArr
endop



/*****************************************************************/
/************* ARRAYS FÜR RHYTHMEN TEILTÖNE TONHÖHEN *************/
/*****************************************************************/

//RHYTHMEN

/* index 0 = länge des rtm in tönen
index 1 = summe der zeiteinheiten
index 2 = dynamischer höhepunkt, ausgedrückt in 1 = erster note etc */

gkRaschRtmKlein_1[] fillarray 4, 14, 3,     3, 2, 5, 4
gkRaschRtmKlein_2[] fillarray 5, 16, 3,     4, 1, 3, 2, 6
gkRaschRtmKlein_3[] fillarray 5, 19, 4,     6, 2, 2, 5, 4
gkRaschRtmKlein_4[] fillarray 4, 14, 3,     4, 2, 3, 5
gkRaschRtmKlein_5[] fillarray 5, 15, 3,     5, 1, 4, 2, 3

gkRaschRtmMittel_1[] fillarray 7, 18, 4,    3, 1, 2, 3, 2, 2, 5
gkRaschRtmMittel_2[] fillarray 7, 21, 4,    5, 1, 2, 4, 2, 1, 6
gkRaschRtmMittel_3[] fillarray 7, 22, 4,    6, 2, 1, 5, 3, 1, 4
gkRaschRtmMittel_4[] fillarray 9, 22.5, 5,  4, 1/2, 1, 2, 3, 3, 1, 3, 9
gkRaschRtmMittel_5[] fillarray 8, 23, 5,    5, 2, 1, 1, 6, 3, 2, 3

gkRaschRtmGross_1[] fillarray 10, 23, 5,    3, 1, 2, 1, 4, 2, 1, 2, 2, 5
gkRaschRtmGross_2[] fillarray 12, 29, 7,    5, 1/2, 1, 1, 1/2, 1, 5, 1, 2, 3, 2, 7
gkRaschRtmGross_3[] fillarray 11, 31, 6,    7, 1, 1, 3, 2, 6, 3, 1, 2, 1, 4
gkRaschRtmGross_4[] fillarray  9, 25, 5,    4, 1, 2, 1, 5, 2, 1, 3, 6
gkRaschRtmGross_5[] fillarray 10, 27, 5,    6, 2, 1, 2, 6, 1, 2, 2, 2, 3

gkHakRtm_1_Phase_1[] fillarray 7, 19,       3, 5, 2, 1, 4, 1.5, 2.5
gkHakRtm_1_Phase_2[] fillarray 9, 7,        1, 1/2, 1, 1, 1/2, 1/2, 1, 1, 1/2
gkHakRtm_2_Phase_1[] fillarray 9, 22,       1, 4, 3, 1, 5, 4, 1, 3
gkHakRtm_2_Phase_2[] fillarray 14,12.5,     2, 3/2, 1, 1/2, 1, 1/2, 1/2, 1, 1, 1/2, 1/2, 1, 1/2, 1
gkHakRtm_3_Phase_1[] fillarray 8, 21,       5, 4, 1, 3, 4, 3, 1
gkHakRtm_3_Phase_2[] fillarray 14,12.5,     3/2, 1, 1/2, 1, 1/2, 3/2, 1/2, 1, 3/2, 1/2, 1/2, 1, 1/2, 1

gkRaschRtmAlle[][] init 21, 20


//TEILTÖNE

/* index 0 = anzahl der teiltöne */

gkRaschPartials_1[] fillarray 3,  1, 3, 4
gkRaschPartials_2[] fillarray 4,  1, 3, 4, 8
gkRaschPartials_3[] fillarray 4,  1, 3, 5, 11
gkRaschPartials_4[] fillarray 3,  1, 3/2, 5/3
gkRaschPartials_5[] fillarray 4,  2, 6, 8, 12

gkRaschPartialsAlle[][] init 5, 10


//TONHÖHEN

//T1 (die letzten vier eigentlich nicht mehr erwartet
gkT1_Pitches[] fillarray 75, 67, 75, 89, 75, 74, 76, 82, 76, 82, 83, 82, 74, 76, 93, 58, 93, 75, 58, 75, 93, 58, 93, 75

//T7
gkT7_Pitchs[] fillarray 85, 100, 106, 110

//T8 schicht eins
gkT8_Pitchs[]  fillarray 93, 58, 93, 75, 58, 75,
                         76, 74, 82, 83, 82, 76, 74, 76,
                         75, 73, 71, 72, 71, 71, 69, 67, 58, 67


instr Fill2dArray

 gkRaschRtmAlle setrow gkRaschRtmKlein_1, 0
 gkRaschRtmAlle setrow gkRaschRtmKlein_2, 1
 gkRaschRtmAlle setrow gkRaschRtmKlein_3, 2
 gkRaschRtmAlle setrow gkRaschRtmKlein_4, 3
 gkRaschRtmAlle setrow gkRaschRtmKlein_5, 4

 gkRaschRtmAlle setrow gkRaschRtmMittel_1, 5
 gkRaschRtmAlle setrow gkRaschRtmMittel_2, 6
 gkRaschRtmAlle setrow gkRaschRtmMittel_3, 7
 gkRaschRtmAlle setrow gkRaschRtmMittel_4, 8
 gkRaschRtmAlle setrow gkRaschRtmMittel_5, 9

 gkRaschRtmAlle setrow gkRaschRtmGross_1, 10
 gkRaschRtmAlle setrow gkRaschRtmGross_2, 11
 gkRaschRtmAlle setrow gkRaschRtmGross_3, 12
 gkRaschRtmAlle setrow gkRaschRtmGross_4, 13
 gkRaschRtmAlle setrow gkRaschRtmGross_5, 14
```

```
 gkRaschRtmAlle setrow gkHakRtm_1_Phase_1, 15
 gkRaschRtmAlle setrow gkHakRtm_1_Phase_2, 16
 gkRaschRtmAlle setrow gkHakRtm_2_Phase_1, 17
 gkRaschRtmAlle setrow gkHakRtm_2_Phase_2, 18
 gkRaschRtmAlle setrow gkHakRtm_3_Phase_1, 19
 gkRaschRtmAlle setrow gkHakRtm_3_Phase_2, 20

 gkRaschPartialsAlle setrow gkRaschPartials_1, 0
 gkRaschPartialsAlle setrow gkRaschPartials_2, 1
 gkRaschPartialsAlle setrow gkRaschPartials_3, 2
 gkRaschPartialsAlle setrow gkRaschPartials_4, 3
 gkRaschPartialsAlle setrow gkRaschPartials_5, 4

 turnoff

endin
schedule("Fill2dArray",0,1)



/*****************************************************************/
/************************* IMMER AN ******************************/
/*****************************************************************/

instr Immer

 //live input in kanal 1 empfangen
 if gkDummy == 0 then
  gaLiveIn inch giInChn
 endif

 //tasten empfangen
 gkKey sensekey

 //return key identifizieren
 if gkKey == 13 && changed(gkKey) == 1 then
  gkKeyReturn = 1
 else
  gkKeyReturn = 0
 endif

endin
schedule "Immer", 0.1, 99999


instr GUI

 //display metro
 gkTrigDisp metro 10

 //mic input
 Meter "mic_in", "mic_in_over", gaLiveIn, gkTrigDisp

 //knuth threshold empfangen
 gkKnuthThreshDb chnget "knuth_thresh_db"

 //knuth min interv tim zeigen
 chnset gkKnuthMinTimIntv, "knuth_min_interv_tim"

 //lautstärken empfangen und glätten
 gkGlobVolDb chnget "glob_vol_db"
 gk_R_VolDbLive chnget "R_vol_db"
 gk_A_VolDbLive chnget "A_vol_db"
 gk_B_VolDbLive chnget "B_vol_db"
 gk_C_VolDbLive chnget "C_vol_db"
 gkYokoDummyVolDb chnget "yoko_dummy_vol_db"
 gkGlobVolDb port gkGlobVolDb, .01
 gk_R_VolDbLive port gk_R_VolDbLive, .01
 gk_A_VolDbLive port gk_A_VolDbLive, .01
 gk_B_VolDbLive port gk_B_VolDbLive, .01
 gk_C_VolDbLive port gk_C_VolDbLive, .01

 //dummy an/aus empfangen
 gkDummy chnget "dummy"

 //instrumentenbeobachtung ein- oder ausschalten
 kZeigAktive chnget "zeig_aktive"
 kZeigAktive_2 chnget "zeig_aktive_2"
 if changed2(kZeigAktive) == 1 then
  if kZeigAktive == 1 then
   event "i", "ShowActive", 0, 999
  else
   turnoff2 "ShowActive", 0, 0
   chnset "off", "aktive_instrumente"
  endif
 endif
 if changed2(kZeigAktive_2) == 1 then
  if kZeigAktive_2 == 1 then
   event "i", "ShowActive_2", 0, 999
  else
   turnoff2 "ShowActive_2", 0, 0
   chnset "off", "aktive_instrumente_2"
  endif
 endif

 //verschiedenes zeigen
 chnset gkRaschVolDbIntern, "R_vol_db_intern"
 chnset gkArpeggVolDbIntern, "ABC_vol_db_intern"
 chnset gkRaschRtmTempo, "R_rtm_tempo"
 chnset gkRaschRtmWackeln, "R_rtm_wackeln"
 chnset gkRaschRtmAuslassungen, "R_rtm_auslass"
 chnset gkRaschHallRoomSiz, "R_hall_roomsiz"
 chnset gkRaschHallWdMix, "R_hall_wdmix"
 chnset gkArpeggHallRoomSiz, "ABC_hall_roomsiz"
 chnset gkArpeggHallWdMix, "ABC_hall_wdmix"
```

```
 chnset gkT6Cue, "T6_cue"

 //button als ersatz für return taste
 kMausreturn chnget "mausreturn"
 if kMausreturn == 1 && changed(kMausreturn) == 1 then
  gkMausReturn = 1
 else
  gkMausReturn = 0
 endif

endin
schedule "GUI", 0.1, 99999


instr Init

 chnset 1, "next_teil"
 chnset 0, "dummy"
 chnset 0, "zeig_aktive"
 chnset 0, "zeig_aktive_2"
 chnset "off", "aktive_instrumente"
 chnset "off", "aktive_instrumente_2"
 turnoff

endin
schedule "Init", 0, 1


instr Teile

 kNextTeil chnget "next_teil"

 //nächsten teil aktivieren durch return taste oder button
 if gkKeyReturn == 1 || gkMausReturn == 1 then
  kAktiviereTeil = 1
 else
  kAktiviereTeil = 0
 endif

 if kAktiviereTeil == 1 then

  kDieserTeil = kNextTeil

  if kDieserTeil == 1 then
   event "i", "Z1_Start", 0, 1

  elseif kDieserTeil == 2 then
   event "i", "T1_Anf", 0, 1

  elseif kDieserTeil == 3 then
   event "i", "T1_End", 0, 1

  elseif kDieserTeil == 4 then
   event "i", "T2_Anf", 0, 1

  elseif kDieserTeil == 5 then
   event "i", "T2_End", 0, 1

  elseif kDieserTeil == 6 then
   event "i", "T3_Anf", 0, 1

  elseif kDieserTeil == 7 then
   event "i", "Z4_Start", 0, 1

  elseif kDieserTeil == 8 then
   event "i", "T4_Anf", 0, 1

  elseif kDieserTeil == 9 then
   event "i", "Z5_Start", 0, 1

  elseif kDieserTeil == 10 then
   event "i", "T5_Start", 0, 1

  elseif kDieserTeil == 11 then
   event "i", "T5_End", 0, 1

  elseif kDieserTeil == 12 then
   event "i", "T6_Anf", 0, 1

  elseif kDieserTeil == 13 then
   event "i", "T6_End", 0, 1

  elseif kDieserTeil == 14 then
   event "i", "T7_Anf", 0, 1

  elseif kDieserTeil == 15 then
   event "i", "Z8_Start", 0, 1

  elseif kDieserTeil == 16 then
   event "i", "T8_Anf", 0, 1

  elseif kDieserTeil == 17 then
   event "i", "T8_End", 0, 1

  endif

  chnset kDieserTeil, "dieser_teil"
  chnset kDieserTeil+1, "next_teil"

 endif

 //vor oder zurück für das nächste auslösen
 if gkKey == 43 && changed(gkKey) == 1 then
  chnset kNextTeil+1, "next_teil"
 endif
```

```
 if gkKey == 45 && changed(gkKey) == 1 then
  chnset kNextTeil-1, "next_teil"
 endif

endin
schedule "Teile", 0.1, 99999


instr Knuth

 //zeit seit dem letzten trigger impuls
 kLastTrigTim init 0

 //zeit zählen
 kLastTrigTim += ksmps/sr

 //vorigen status der rms schwellüberschreitung auf null setzen
 kRmsPrev init 0

 //input analysieren
 kRms rms gaLiveIn

 /*ask for the new value being the first one jumping over kthresh*/
 if kRms > ampdb(gkKnuthThreshDb) && kRms > kRmsPrev && kLastTrigTim > gkKnuthMinTimIntv then
  //anzeige
  event "i", "KnuthShowTrigImp", 0, .1
  gkKnuthTrigImp = 1
  //zeit zurücksetzen
  kLastTrigTim = 0
 else
  gkKnuthTrigImp = 0
 endif

 kRmsPrev = kRms

endin
maxalloc "Knuth", 1


instr Knuth_Stop

 turnoff2 "Knuth", 0, 0
 turnoff

endin


instr Knuth_key

 if gkKey == 32 && changed(gkKey) == 1 then
  gkKnuthTrigImp = 1
  event "i", "KnuthShowTrigImp", 0, .1
 else
  gkKnuthTrigImp = 0
 endif

endin
maxalloc "Knuth_key", 1


instr Knuth_key_Stop

 turnoff2 "Knuth_key", 0, 0
 turnoff

endin


/*****************************************************************/
/*********************** DUMMIES *********************************/
/*****************************************************************/

instr T1_dummy

 SF = "../Yoko_Texte_und_Aufnahmen/Richard III.mp3"
 p3 mp3len SF
 aL, aR mp3in SF
 gaLiveIn = aL*ampdb(gkYokoDummyVolDb) * 1.2
 out gaLiveIn, gaLiveIn

endin
maxalloc "T1_dummy", 1

instr T2_dummy

 SF = "../Yoko_Texte_und_Aufnahmen/Personenschaden.mp3"
 p3 mp3len SF
 aL, aR mp3in SF
 gaLiveIn = aL*ampdb(gkYokoDummyVolDb)
 out gaLiveIn, gaLiveIn

endin
maxalloc "T2_dummy", 1

instr T3_dummy

 SF = "../Yoko_Texte_und_Aufnahmen/Westerland.mp3"
 p3 mp3len SF
 aL, aR mp3in SF
 gaLiveIn = aL*ampdb(gkYokoDummyVolDb) * 2
 out gaLiveIn, gaLiveIn

endin
maxalloc "T3_dummy", 1
```

```
instr T4_dummy

 SF = "../Yoko_Texte_und_Aufnahmen/Schweigen.mp3"
 p3 mp3len SF
 aL, aR mp3in SF
 gaLiveIn = aL*ampdb(gkYokoDummyVolDb) * 2
 out gaLiveIn, gaLiveIn

endin
maxalloc "T4_dummy", 1

instr T5_dummy

 SF = "../Yoko_Texte_und_Aufnahmen/vor hellem Vokal.mp3"
 p3 mp3len SF
 aL, aR mp3in SF
 gaLiveIn = aL*ampdb(gkYokoDummyVolDb)
 out gaLiveIn, gaLiveIn

endin
maxalloc "T5_dummy", 1


instr T6_dummy

 SF = "../Yoko_Texte_und_Aufnahmen/Yokohama.mp3"
 p3 mp3len SF
 aL, aR mp3in SF
 gaLiveIn = aL*ampdb(gkYokoDummyVolDb)
 out gaLiveIn, gaLiveIn

endin
maxalloc "T6_dummy", 1


instr T7_dummy

 SF = "../Yoko_Texte_und_Aufnahmen/Taub im Himmel.mp3"
 p3 mp3len SF
 aL, aR mp3in SF
 gaLiveIn = aL*ampdb(gkYokoDummyVolDb)
 out gaLiveIn, gaLiveIn

endin
maxalloc "T7_dummy", 1


instr T8_dummy

 SF = "../Yoko_Texte_und_Aufnahmen/Hamlet.mp3"
 p3 mp3len SF
 aL, aR mp3in SF
 gaLiveIn = aL*ampdb(gkYokoDummyVolDb)
 out gaLiveIn, gaLiveIn

endin
maxalloc "T8_dummy", 1



/*****************************************************************/
/************************ ABLAUF *********************************/
/*****************************************************************/

instr Z1_Start
//beginn des ersten zwischenspiels

 gkRaschRtmTempo = 80
 gkRaschRtmWackeln = 50
 gkRaschHallRoomSiz = .8
 gkRaschHallWdMix = .2
 gkRaschVolDbIntern = 0
 gkRaschRtmAuslassungen = 1

 event "i", "Z1", 0, 999

 turnoff

endin


instr T1_Anf
//in dem moment wo yoko anfängt zu lesen

 //rascheln leiser machen
 schedule "ChangeRaschVolDbIntern", 0, 10, i(gkRaschVolDbIntern)-18

 //mehr auslassungen
 schedule "ChangeRaschRtmAuslassungen", 0, 10, 3

 //dummy oder nicht
 if i(gkDummy) == 1 then
  schedule "T1_dummy", 0, 1
 endif

 //hallparameter setzen
 gkArpeggHallRoomSiz = .5
 gkArpeggHallWdMix = .3

 //falls man mal zurückgeht
 gkT1_End = 0

 //arpeggiatios anschalten
 event "i", "T1", 0, 999
```

```
 turnoff

endin


instr Z1_Stop
//während des ersten textes

 turnoff2 "Z1", 0, 0

 turnoff

endin


instr Z2_Start
//noch während des ersten textes

 schedule "Z2", 0, 999

 turnoff

endin


instr Z2_Stop

 turnoff2 "Z2", 0, 0

 turnoff

endin


instr T1_End
//quasi nach dem letzten wort von text 1

 //tempo und lautstärke steigen
 schedule "ChangeRaschRtmTempo", 0, 5, 120
 schedule "ChangeRaschVolDbIntern", 0, 7, 0

 //keine auslassungen mehr
 schedule "ChangeRaschRtmAuslassungen", 0, 10, 1

 //arpeggiatios ausschalten
 turnoff2 "T1", 0, 0

 //für Z2 piekser
 gkT1_End = 1

 //falls dabei
 turnoff2 "T1_dummy", 0, 0

endin


instr T2_Anf

 //dummy oder nicht
 if i(gkDummy) == 1 then
  schedule "T2_dummy", 0, 1
 endif

 //knuth starten und mintiminterv setzen
 schedule "Knuth", 0, 999
 gkKnuthMinTimIntv = 1/7

 //Z2 wird leiser und ist dann ganz weg
 schedule "ChangeRaschVolDbIntern", 0, 10, -20
 schedule "ChangeRaschRtmAuslassungen", 0, 10, 3
 schedule "Z2_Stop", 10, 1

 //kein hall
 gkArpeggHallWdMix = 0

 //T2 starten
 event "i", "T2", 0, 999

 turnoff

endin


instr T2_Stop

 turnoff2 "Knuth", 0, 0
 turnoff2 "T2", 0, 0
 turnoff2 "T2_dummy", 0, 0

 turnoff

endin


instr T2_End

 //Knuth und alles ausschalten
 schedule "T2_Stop", 0, 1

 //Z3_Start rufen (wartet auf leertaste)
 schedule "Z3_Start", 0, 999

 turnoff

endin
```

```
instr Z3_Start

 //parameter setzen
 gkRaschRtmTempo = 60
 gkRaschRtmWackeln = 50
 gkRaschHallRoomSiz = .8
 gkRaschHallWdMix = .2
 gkRaschVolDbIntern = 0
 gkRaschRtmAuslassungen = 1

 //beim druck der leertaste Z3 aktivieren und weg
 if gkKey == 32 then
  event "i", "Z3", 0, 999
  turnoff
 endif

endin
maxalloc "Z3_Start", 1


instr Z3_direkt

 schedule "Z3", 0, 999
 turnoff

endin


instr Z3_Stop

 turnoff2 "Z3", 0, 0
 turnoff

endin



instr T3_Anf

 //hak-rtm wird leiser und löchriger und dann ausgeschaltet
 schedule "ChangeRaschVolDbIntern", 0, 10, -20
 schedule "ChangeRaschRtmAuslassungen", 0, 10, 3
 schedule "Z3_Stop", 10, 1

 //Knuth_key und T3 anschalten
 schedule "Knuth_key", 0, 999
 schedule "T3", 0, 999

 //hallvariablen setzen
 gkArpeggHallRoomSiz = .8
 gkArpeggHallWdMix = .3

 //dummy oder nicht
 if i(gkDummy) == 1 then
  schedule "T3_dummy", 0, 1
 endif

 turnoff

endin


instr T3_Stop
//nur not

 turnoff2 "Knuth_key", 0, 0
 turnoff2 "T3", 0, 0
 turnoff2 "T3_dummy", 0, 0

 turnoff

endin


instr Z4_Start

 //Knuth und T3 ausschalten
 turnoff2 "Knuth_key", 0, 0
 turnoff2 "T3", 0, 0
 turnoff2 "T3_dummy", 0, 0

 //werte für das nüchste zwischenspiel setzen und anfangen
 gkRaschRtmTempo = 80
 gkRaschRtmWackeln = 50
 gkRaschHallRoomSiz = .8
 gkRaschHallWdMix = .2
 gkRaschVolDbIntern = 0
 gkRaschRtmAuslassungen = 1

 event "i", "Z4", 0, 999

 turnoff

endin


instr Z4_Stop

 turnoff2 "Z4", 0, 0
 turnoff

endin
```

```
instr T4_Anf

 //Z4 ausschalten
 schedule "Z4_Stop", 0, 1

 //rascheln leiser machen
 schedule "ChangeRaschVolDbIntern", 0, 10, i(gkRaschVolDbIntern)-30

 //mehr auslassungen
 schedule "ChangeRaschRtmAuslassungen", 0, 10, 3

 //parameter setzen
 gkArpeggHallWdMix = 0

 //instrumente starten
 schedule "T4", 0, 999

 //dummy oder nicht
 if i(gkDummy) == 1 then
  schedule "T4_dummy", 0, 1
 endif

 turnoff

endin


instr T4_End

 turnoff2 "T4", 0, 0
 turnoff2 "T4_dummy", 0, 0

 turnoff

endin



instr Z5_Start

 //F4 ausschalten
 schedule "T4_End", 0, 1

 //parameter setzen
 gkRaschRtmTempo = 30
 gkRaschRtmWackeln = 0
 gkRaschHallRoomSiz = .8
 gkRaschHallWdMix = .2
 gkRaschVolDbIntern = 0
 gkRaschRtmAuslassungen = 0

 //nach einer freien pause
 event "i", "Z5", 0, 999

 turnoff

endin


instr Z5_Stop

 turnoff2 "Z5", 0, 0
 turnoff

endin


instr T5_Start

 //hallvariablen setzen
 gkArpeggHallRoomSiz = .8
 gkArpeggHallWdMix = .1

 //T5 rufen
 event "i", "T5", 0, 999

 //dummy nach 8 sekunden (oder gar nicht)
 if i(gkDummy) == 1 then
  schedule "T5_dummy", 8, 1
 endif

 turnoff

endin


instr T5_Stop

 turnoff2 "T5", 0, 0
 turnoff2 "T5_dummy", 0, 0
 turnoff

endin


instr T5_End

 schedule "T5_Stop", 0, 1
 schedule "Z6_Start", 0, 1

 turnoff

endin
```

```
instr Z6_Start

 //werte setzen
 gkRaschRtmTempo = 120
 gkRaschRtmWackeln = 30
 gkRaschHallRoomSiz = .8
 gkRaschHallWdMix = .2
 gkRaschVolDbIntern = 0

 event "i", "Z6", 0, 999

 turnoff

endin


instr Z6_Stop

 //Z6 ausschalten und laufenden klang relativ schnell ausblenden
 turnoff2 "Z6", 0, 0
 turnoff2 "Knuth_key", 0, 0
 schedule "ChangeRaschVolDbIntern", 0, 5, -20

 turnoff

endin


instr T6_Anf

 schedule "Z6_Stop", 0, 1

 //parameter setzen
 gkArpeggHallWdMix = 0
 gkKnuthMinTimIntv = 1/7

 //instrumente starten
 schedule "T6_Cues", 0, 999
 schedule "T6", 0, 999
 schedule "Knuth", 0, 999

 //dummy oder nicht
 if i(gkDummy) == 1 then
  schedule "T6_dummy", 0, 1
 endif

 turnoff

endin


instr T6_Stop

 turnoff2 "T6_Cues", 0, 0
 turnoff2 "T6", 0, 0
 turnoff2 "Knuth", 0, 0
 turnoff2 "T6_dummy", 0, 0

 turnoff

endin


instr T6_End

 schedule "T6_Stop", 0, 1
 schedule "Z7_Start", 0, 1

 turnoff

endin


instr Z7_Start

 //werte setzen und anfangen
 gkRaschRtmTempo = 150
 gkRaschRtmWackeln = 15
 gkRaschHallRoomSiz = .8
 gkRaschHallWdMix = .2
 gkRaschVolDbIntern = 0

 event "i", "Z7", 0, 999

 turnoff

endin


instr Z7_Stop

 turnoff2 "Z7", 0, 0
 turnoff2 "Z7_seq", 0, 0
 turnoff2 "Z7_tief", 0, 0
 turnoff2 "Knuth_key", 0, 0
 turnoff

endin


instr T7_Anf

 schedule "Z7_Stop", 0, 1
```

```
 gkArpeggHallWdMix = 0.2
 gkArpeggHallRoomSiz = .3

 event "i", "T7", 0, 999

 //dummy oder nicht
 if i(gkDummy) == 1 then
  schedule "T7_dummy", 0, 1
 endif

 turnoff

endin


instr T7_End

 turnoff2 "T7", 0, 0
 turnoff2 "T7_dummy", 0, 0
 turnoff

endin


instr Z8_Start

 schedule "T7_End", 0, 1

 gkRaschRtmTempo = 66
 gkRaschRtmWackeln = 50
 gkRaschHallRoomSiz = .8
 gkRaschHallWdMix = .2
 gkRaschVolDbIntern = 0
 gkRaschRtmAuslassungen = 1

 event "i", "Z8", 0, 999
 turnoff

endin


instr Z8_Stop

 turnoff2 "Z8", 0, 0
 turnoff

endin


instr T8_Anf

 //rascheln leiser machen und dann ausschalten
 schedule "ChangeRaschVolDbIntern", 0, 40, i(gkRaschVolDbIntern)-40
 schedule "Z8_Stop", 40, 1

 //mehr auslassungen
 schedule "ChangeRaschRtmAuslassungen", 0, 10, 3

 //dummy oder nicht
 if i(gkDummy) == 1 then
  schedule "T8_dummy", 0, 1
 endif

 //beginnen
 gkKnuthMinTimIntv = 3
 gkT8_pch = 93

 schedule "Knuth", 0, 999
 event "i", "T8_1", 3, 999
 event "i", "T8_2", 0, 999

 turnoff

endin


instr T8_End

 turnoff2 "Knuth", 0, 0
 turnoff2 "T8_1", 0, 0
 turnoff2 "T8_2", 0, 0

 turnoff2 "T8_dummy", 0, 0

 turnoff

endin



/*****************************************************************/
/********************** ZWISCHENSPIELE ***************************/
/*****************************************************************/



instr Z1
//generiert die raschel folgen in zwischenspiel 1
//geht auch weiter wenn text 1 kommt, nur leiser und mit auslassungen


/* input */

 kPauseMin = 1 ;sekunden
```

```
 kPauseMax = 5
 kTempo = gkRaschRtmTempo
 kWackeln = gkRaschRtmWackeln
 kDb[] fillarray giDb_Z1, giDb_Z1+6, giDb_Z1-6
 kDbMaxDev = 4
 kPitch[] fillarray 82, 83, 81
 kPitchMaxDev = 1 ;halbtöne

 kCount init 0

 ;q
 iQ_min = 1
 iQ_max = 2

 ;grain density (immer konstant für einen rascheler)
 iGrainDensMin = 60
 iGrainDensMax = 90

 ;verhältnis einblende zu gesamtdauer bei einzelnen raschelern
 iEnvRatioMin = .2
 iEnvRatioMax = .4

 ;verhältnis realdauer eines raschelers zur dauer im rtm (2 doppelt so lang)
 iRaschDurFacMin = 2
 iRaschDurFacMax = 3

 ;kurvernform für ansteigen und abfallen des raschelns (beides positiv, größer=konvexer)
 iTypeIn = 1
 iTypeOut = 6

 ;index in gkRaschPartialsAlle
 iPartials = 2

 ;abfolge aus raschel-rhythmen (erwartet dass nur die ersten drei gehört werden)
 kSeq[] fillarray 5, 4, 10, 6, 2, 13


/* eigentliches maschinchen hier */

 ;uhr initialisieren (in k-zyklen)
 kUhr_k init 0

 ;nächste sequenz initalisieren (1=ja)
 kPlaySeq init 1

 ;dauer der sequenz in k-zyklen (wird unten gleich überschrieben)
 kSeqDur_k init 0

 ;index in kSeq, gleich auf loop hin
 kIndx init 0
 kIndx = kIndx % lenarray(kSeq)

 ;index des raschel-rtm in gkRaschRtmAlle
 kRaschRtmIndex = kSeq[kIndx]

 ;wenn eine sequenz ausgelöst werden soll
 if kPlaySeq == 1 then

  ;verschiedene behandlung der ersten beiden gruppen
  if kCount == 0 then
   kExtraDb = -10
   kExtraPause = 2
  elseif kCount == 1 then
   kExtraDb = -5
   kExtraPause = 1
  else
   kExtraDb = 0
   kExtraPause = 0
  endif

  ;ermittele die zeit für das nächste auslösen
  kRtmTimUnits = gkRaschRtmAlle[kRaschRtmIndex][1] ;in zeiteinheiten
  kRtmTimSecs = (kRtmTimUnits / 4) * (60 / kTempo)
  kPause random kPauseMin, kPauseMax
  kSeqDur_k = round((kRtmTimSecs+kPause+kExtraPause)*kr)

  ;ruf das instrument
  event "i", "RaschSeq", 0, kRtmTimSecs, kRaschRtmIndex, kTempo, kWackeln,
        kDb[0]+random:k(0,kDbMaxDev)+kExtraDb, kDb[1]+random:k(0,kDbMaxDev)+kExtraDb, kDb[2]+random:k(0,kDbMaxDev)
+kExtraDb,
        kPitch[0]+random:k(0,kPitchMaxDev), kPitch[1]+random:k(0,kPitchMaxDev), kPitch[2]+random:k(0,kPitchMaxDev),
        iQ_min, iQ_max, iGrainDensMin, iGrainDensMax, iEnvRatioMin, iEnvRatioMax, iRaschDurFacMin, iRaschDurFacMax,
        iTypeIn, iTypeOut, iPartials, 1

  ;uhr zurücksetzen
  kUhr_k = 0

  kCount += 1

 endif

 ;schalter um sequenz auszulösen
 if kUhr_k == kSeqDur_k then
  kPlaySeq = 1
  kIndx += 1
 else
  kPlaySeq = 0
 endif

 ;zeit hoppelt weiter
 kUhr_k += 1

endin
maxalloc "Z1", 1
```

```
instr Z2

/* input */

 kPauseMin = 1 ;sekunden
 kPauseMax = 2.5
 kTempo = gkRaschRtmTempo
 kWackeln = gkRaschRtmWackeln
 kDb[] fillarray giDb_Z2, giDb_Z2+10, giDb_Z2-4
 kDbMaxDev = 4
 kPitch[] fillarray 95, 97, 94
 kPitchMaxDev = 1 ;halbtöne

 ;abfolge aus raschel-rhythmen
 kSeq[] fillarray 7, 1, 2, 3, 4

 ;q
 iQ_min = 1
 iQ_max = 2

 ;grain density (immer konstant für einen rascheler)
 iGrainDensMin = 60
 iGrainDensMax = 90

 ;verhältnis einblende zu gesamtdauer bei einzelnen raschelern
 iEnvRatioMin = .1
 iEnvRatioMax = .3

 ;verhältnis realdauer eines raschelers zur dauer im rtm (2 doppelt so lang)
 iRaschDurFacMin = 1.5
 iRaschDurFacMax = 2.5

 ;kurvernform für ansteigen und abfallen des raschelns (beides positiv, größer=konvexer)
 iTypeIn = 1
 iTypeOut = 6

 ;index in gkRaschPartialsAlle
 iPartials = 0

/* durchführung */

 ;uhr initialisieren (in k-zyklen)
 kUhr_k init 0

 ;nächste sequenz initalisieren (1=ja)
 kPlaySeq init 1

 ;dauer der sequenz in k-zyklen (wird unten gleich überschrieben)
 kSeqDur_k init 0

 ;index in kSeq, gleich auf loop hin
 kIndx init 0
 kIndx = kIndx % lenarray(kSeq)

 ;index des raschel-rtm in gkRaschRtmAlle
 kRaschRtmIndex = kSeq[kIndx]

 ;sequenzen mitzählen ab dann wenn T1 fertig
 kCount init 1

 ;wenn eine sequenz ausgelöst werden soll
 if kPlaySeq == 1 then

  ;ermittele die zeit für das nächste auslösen
  kRtmTimUnits = gkRaschRtmAlle[kRaschRtmIndex][1] ;in zeiteinheiten
  kRtmTimSecs = (kRtmTimUnits / 4) * (60 / kTempo)
  kPause random kPauseMin, kPauseMax
  kSeqDur_k = round((kRtmTimSecs+kPause)*kr)

  ;ruf das instrument
  event "i", "RaschSeq_Z2", 0, kRtmTimSecs, kRaschRtmIndex, kTempo, kWackeln,
        kDb[0]+random:k(0,kDbMaxDev), kDb[1]+random:k(0,kDbMaxDev), kDb[2]+random:k(0,kDbMaxDev),
        kPitch[0]+random:k(0,kPitchMaxDev), kPitch[1]+random:k(0,kPitchMaxDev), kPitch[2]+random:k(0,kPitchMaxDev),
        iQ_min, iQ_max, iGrainDensMin, iGrainDensMax, iEnvRatioMin, iEnvRatioMax, iRaschDurFacMin, iRaschDurFacMax,
        iTypeIn, iTypeOut, iPartials, 1, kCount

  ;uhr zurücksetzen
  kUhr_k = 0

  ;wenn T1 fertig mitzählen
  if gkT1_End == 1 then
   kCount += 1
  endif

 endif

 ;schalter um sequenz auszulösen
 if kUhr_k == kSeqDur_k then
  kPlaySeq = 1
  kIndx += 1
 else
  kPlaySeq = 0
 endif

 ;zeit hoppelt weiter
 kUhr_k += 1

endin
maxalloc "Z2", 1


instr Z3
```

```
/* input */

 kPauseMin = 1 ;sekunden
 kPauseMax = 3
 kTempo = gkRaschRtmTempo
 kWackeln = gkRaschRtmWackeln
 kDb[] fillarray giDb_Z3, giDb_Z3+10, giDb_Z3+16
 kDbMaxDev = 4
 kPitch[] fillarray 107, 108, 110
 kPitchMaxDev = 1 ;halbtöne

 iDurLangerTonMin = 3
 iDurLangerTonMax = 5

 iQ_min = 1
 iQ_max = 2

 ;grain density (immer konstant für einen rascheler)
 iGrainDensMin = 60
 iGrainDensMax = 90

 ;verhältnis einblende zu gesamtdauer bei einzelnen raschelern
 iEnvRatioMin = .05
 iEnvRatioMax = .2

 ;indices der ersten phasen in gkRaschRtmAlle
 iSeq[] fillarray 15, 17, 19
 iSeqPerm[] ArrPermRnd iSeq

 ;verhältnis realdauer eines raschelers zur dauer im rtm (2 doppelt so lang)
 iRaschDurFacMin = 1
 iRaschDurFacMax = 1.5

 ;kurvernform für ansteigen und abfallen des raschelns (beides positiv, größer=konvexer)
 iTypeIn = 1
 iTypeOut = 6

 ;index in gkRaschPartialsAlle
 iPartials = 3

/* durchführung */

 ;uhr initialisieren (in k-zyklen)
 kUhr_k init 0

 ;nächste sequenz initalisieren (1=ja)
 kPlaySeq init 1

 ;dauer der sequenz in k-zyklen (wird unten gleich überschrieben)
 kSeqDur_k init 0

 ;index in kSeq, gleich auf loop hin
 kIndx init 0
 kIndx = kIndx % lenarray(iSeqPerm)

 ;index des raschel-rtm in gkRaschRtmAlle
 kRaschRtmIndex = iSeqPerm[kIndx]

 ;wenn eine sequenz ausgelöst werden soll
 if kPlaySeq == 1 then

  ;zeit des langen tons
  kDurLangerTon random iDurLangerTonMin, iDurLangerTonMax

  ;ermittele die zeit für das nächste auslösen
  kRtmTimUnitsPhas_1 = gkRaschRtmAlle[kRaschRtmIndex][1] ;in zeiteinheiten
  kRtmTimUnitsPhas_2 = gkRaschRtmAlle[kRaschRtmIndex+1][1]
  kRtmTimUnits = kRtmTimUnitsPhas_1 + kRtmTimUnitsPhas_2

  kRtmTimSecs = (kRtmTimUnits / 4) * (60 / kTempo) + kDurLangerTon
  kPause random kPauseMin, kPauseMax
  kSeqDur_k = round((kRtmTimSecs+kPause)*kr)

  ;ruf das instrument
  event "i", "HakSeq", 0, kRtmTimSecs, kRaschRtmIndex, kTempo, kWackeln,
        kDb[0]+random:k(0,kDbMaxDev), kDb[1]+random:k(0,kDbMaxDev), kDb[2]+random:k(0,kDbMaxDev),
        kPitch[0]+random:k(0,kPitchMaxDev), kPitch[1]+random:k(0,kPitchMaxDev), kPitch[2]+random:k(0,kPitchMaxDev),
        kDurLangerTon, iQ_min, iQ_max, iGrainDensMin, iGrainDensMax, iRaschDurFacMin, iRaschDurFacMax,
        iEnvRatioMin, iEnvRatioMax, iTypeIn, iTypeOut, iPartials

  ;uhr zurücksetzen
  kUhr_k = 0

 endif

 ;schalter um sequenz auszulösen
 if kUhr_k == kSeqDur_k then
  kPlaySeq = 1
  kIndx += 1
 else
  kPlaySeq = 0
 endif

 ;zeit hoppelt weiter
 kUhr_k += 1

endin
maxalloc "Z3", 1


instr Z4

/* input */

 kPauseMin = 2 ;sekunden
```

```
 kPauseMax = 4
 kTempo = gkRaschRtmTempo
 kWackeln = gkRaschRtmWackeln
 kDb[] fillarray giDb_Z4, giDb_Z4+10, giDb_Z4-4
 kDbMaxDev = 4
 kPitch[] fillarray 82, 83, 81
 kPitchMaxDev = 1 ;halbtöne

 ;abfolge aus raschel-rhythmen
 kSeq[] fillarray 10, 11, 12, 14

 ;q
 iQ_min = 1
 iQ_max = 2

 ;grain density (immer konstant für einen rascheler)
 iGrainDensMin = 60
 iGrainDensMax = 90

 ;verhältnis einblende zu gesamtdauer bei einzelnen raschelern
 iEnvRatioMin = .2
 iEnvRatioMax = .4

 ;verhältnis realdauer eines raschelers zur dauer im rtm (2 doppelt so lang)
 iRaschDurFacMin = 2
 iRaschDurFacMax = 3

 ;kurvernform für ansteigen und abfallen des raschelns (beides positiv, größer=konvexer)
 iTypeIn = 1
 iTypeOut = 6

 ;index in gkRaschPartialsAlle
 iPartials = 2

/* durchführung */

 ;uhr initialisieren (in k-zyklen)
 kUhr_k init 0

 ;nächste sequenz initalisieren (1=ja)
 kPlaySeq init 1

 ;dauer der sequenz in k-zyklen (wird unten gleich überschrieben)
 kSeqDur_k init 0

 ;index in kSeq, gleich auf loop hin
 kIndx init 0
 kIndx = kIndx % lenarray(kSeq)

 ;index des raschel-rtm in gkRaschRtmAlle
 kRaschRtmIndex = kSeq[kIndx]

 ;wenn eine sequenz ausgelöst werden soll
 if kPlaySeq == 1 then

  ;ermittele die zeit für das nächste auslösen
  kRtmTimUnits = gkRaschRtmAlle[kRaschRtmIndex][1] ;in zeiteinheiten
  kRtmTimSecs = (kRtmTimUnits / 4) * (60 / kTempo)
  kPause random kPauseMin, kPauseMax
  kSeqDur_k = round((kRtmTimSecs+kPause)*kr)

  ;ruf das instrument
  event "i", "RaschSeq", 0, kRtmTimSecs, kRaschRtmIndex, kTempo, kWackeln,
        kDb[0]+random:k(0,kDbMaxDev), kDb[1]+random:k(0,kDbMaxDev), kDb[2]+random:k(0,kDbMaxDev),
        kPitch[0]+random:k(0,kPitchMaxDev), kPitch[1]+random:k(0,kPitchMaxDev), kPitch[2]+random:k(0,kPitchMaxDev),
        iQ_min, iQ_max, iGrainDensMin, iGrainDensMax, iEnvRatioMin, iEnvRatioMax, iRaschDurFacMin, iRaschDurFacMax,
        iTypeIn, iTypeOut, iPartials, 1

  ;uhr zurücksetzen
  kUhr_k = 0

 endif

 ;schalter um sequenz auszulösen
 if kUhr_k == kSeqDur_k then
  kPlaySeq = 1
  kIndx += 1
 else
  kPlaySeq = 0
 endif

 ;zeit hoppelt weiter
 kUhr_k += 1

endin
maxalloc "Z4", 1


instr Z5
//nur zwei durchläufe, dann pause

/* input */

 kPauseMin = 3 ;sekunden
 kPauseMax = 4
 kTempo = gkRaschRtmTempo
 kWackeln = gkRaschRtmWackeln
 kDb[] fillarray giDb_Z5, giDb_Z5+4, giDb_Z5-4
 kDbMaxDev = 2
 kPitch[] fillarray 71, 72, 70
 kPitchMaxDev = 1 ;halbtöne

 ;abfolge aus raschel-rhythmen
 kSeq[] fillarray 0, 8
```

```
 ;q
 iQ_min = 1
 iQ_max = 2

 ;grain density (immer konstant für einen rascheler)
 kGrainDensMin linseg 90, 10, 120
 kGrainDensMax linseg 100, 10, 150

 ;verhältnis einblende zu gesamtdauer bei einzelnen raschelern
 iEnvRatioMin = .3
 iEnvRatioMax = .5

 ;verhältnis realdauer eines raschelers zur dauer im rtm (2 doppelt so lang)
 iRaschDurFacMin = 3/2
 iRaschDurFacMax = 2

 ;kurvernform für ansteigen und abfallen des raschelns (beides positiv, größer=konvexer)
 iTypeIn = 3
 iTypeOut = 6

 ;index in gkRaschPartialsAlle
 iPartials = 4

 //amplitudenveränderung der teiltöne
 iDurTTChange = 25
 gkZ5_db_TT1 linseg 0, iDurTTChange, -10
 gkZ5_db_TT2 linseg 0, iDurTTChange, 0
 gkZ5_db_TT3 linseg 0, iDurTTChange, 10
 gkZ5_db_TT4 linseg 10, iDurTTChange, 20

/* durchführung */

 ;uhr initialisieren (in k-zyklen)
 kUhr_k init 0

 ;nächste sequenz initalisieren (1=ja)
 kPlaySeq init 1

 ;dauer der sequenz in k-zyklen (wird unten gleich überschrieben)
 kSeqDur_k init 0

 ;index in kSeq, ausschalten nach zwei durchläufen
 kIndx init 0
 if kIndx == lenarray(kSeq) then
  turnoff
 endif

 ;index des raschel-rtm in gkRaschRtmAlle
 kRaschRtmIndex = kSeq[kIndx]

 ;wenn eine sequenz ausgelöst werden soll
 if kPlaySeq == 1 then

  ;ermittele die zeit für das nächste auslösen
  kRtmTimUnits = gkRaschRtmAlle[kRaschRtmIndex][1] ;in zeiteinheiten
  kRtmTimSecs = (kRtmTimUnits / 4) * (60 / kTempo)
  kPause random kPauseMin, kPauseMax
  kSeqDur_k = round((kRtmTimSecs+kPause)*kr)

  ;ruf das instrument
  event "i", "RaschSeq", 0, kRtmTimSecs, kRaschRtmIndex, kTempo, kWackeln,
        kDb[0]+random:k(0,kDbMaxDev), kDb[1]+random:k(0,kDbMaxDev), kDb[2]+random:k(0,kDbMaxDev),
        kPitch[0]+random:k(0,kPitchMaxDev), kPitch[1]+random:k(0,kPitchMaxDev), kPitch[2]+random:k(0,kPitchMaxDev),
        iQ_min, iQ_max, kGrainDensMin, kGrainDensMax, iEnvRatioMin, iEnvRatioMax, iRaschDurFacMin, iRaschDurFacMax,
        iTypeIn, iTypeOut, iPartials, 2

  ;uhr zurücksetzen
  kUhr_k = 0

 endif

 ;schalter um sequenz auszulösen
 if kUhr_k == kSeqDur_k then
  kPlaySeq = 1
  kIndx += 1
 else
  kPlaySeq = 0
 endif

 ;zeit hoppelt weiter
 kUhr_k += 1

endin
maxalloc "Z5", 1


instr Z6

 kSeqLen init 1

 if gkKey == 32 && changed(gkKey) == 1 then

   event "i", "Z6_seq", 0, 99, kSeqLen
   kSeqLen += 1
   gkRaschRtmTempo += 20

 endif


endin
maxalloc "Z6", 1


instr Z6_seq
```

```
 //wieviele toene pro sequenz
 kSeqToene init p4

 //bei welchem ton sind wir
 kTon init 1

 //zeit als rückwärts laufende uhr
 kTime init 0

 //dauer der zeiten (1 = ein schlag von gkRaschRtmTempo)
 kKurz init 1/2
 kLang init 3/2

 //tonhöhe und abweichung
 iPitch = 60
 iPitchMaxDev = 2

 //lautstärke und abweichung
 iDb = giDb_Z6
 iDbMaxDev = 4

 ;q
 iQ_min = 5
 iQ_max = 10

 ;grain density (immer konstant für einen rascheler)
 iGrainDensMin = 60
 iGrainDensMax = 90

 ;verhältnis einblende zu gesamtdauer bei einzelnen raschelern
 iEnvRatioMin = .2
 iEnvRatioMax = .4

 ;verhältnis realdauer eines raschelers zur dauer im rtm (2 doppelt so lang)
 iRaschDurFacMin = 2
 iRaschDurFacMax = 3

 ;kurvernform für ansteigen und abfallen des raschelns (beides positiv, größer=konvexer)
 iTypeIn = 1
 iTypeOut = 6

 ;panning (später vielleicht anders)
 kPan randomi .1, .9, 1/10, 3

 ;index in gkRaschPartialsAlle
 iPartials = 2

 //wenn ein neues ereignis kommt
 if kTime <= 0 then

  //werte bestimmen
  kPitch = iPitch + rand:k(iPitchMaxDev,2,1)
  kDb = iDb + rand:k(iDbMaxDev,2,1)
  kEnvRatio random iEnvRatioMin, iEnvRatioMax
  kGrainDens random iGrainDensMin, iGrainDensMax

  //spezielle werte für kurzen ton
  if kTon < kSeqToene then

   kOffset random 0, kKurz * (gkRaschRtmWackeln/100)
   kDur = kKurz * random:k(iRaschDurFacMin,iRaschDurFacMax) * (60/gkRaschRtmTempo)
   kTon += 1
   kTime = kKurz * (60/gkRaschRtmTempo)

   event "i", "EinRaschler", kOffset, kDur, kPitch, kDb, kPan,
              kEnvRatio, iQ_min, iQ_max, kGrainDens,
              iTypeIn, iTypeOut, iPartials

  //für langen ton
  else

   kOffset random 0, kKurz * (gkRaschRtmWackeln/100) ;wirklich
   kDur = kLang * random:k(iRaschDurFacMin,iRaschDurFacMax) * (60/gkRaschRtmTempo)

   event "i", "EinRaschler", kOffset, kDur, kPitch, kDb, kPan,
              kEnvRatio, iQ_min, iQ_max, kGrainDens,
              iTypeIn, iTypeOut, iPartials

   turnoff

  endif

 endif

 kTime -= 1/kr

endin


instr Z7
//Z6 quasi rückwärts

 kSeqLen init 6
 kLangDur init 3
 kDb init giDb_Z8

 if gkKey == 32 && changed(gkKey) == 1 then

  if kSeqLen > 1 then

   event "i", "Z7_seq", 0, 99, kSeqLen
   kSeqLen -= 1

  else
```

```
   event "i", "Z7_tief", 0, kLangDur, kDb
   kLangDur *= 3
   kDb -= 9

  endif

 endif

endin
maxalloc "Z7", 1


instr Z7_seq
//spielt eine gruppe

 //wieviele toene pro sequenz am anfang
 kSeqToene init p4

 //bei welchem ton sind wir
 kTon init 1

 //zeit als rückwärts laufende uhr
 kTime init 0

 //dauer der zeiten (1 = ein schlag von gkRaschRtmTempo)
 kKurz init 1/3
 kLang init 3/2

 //tonhöhe und abweichung
 iPitch = 58
 iPitchMaxDev = 1

 //lautstärke und abweichung
 iDb = giDb_Z7
 iDbMaxDev = 4
 iDbPlus = 8 ;schlusston

 ;q
 iQ_min = 3
 iQ_max = 7

 ;grain density (immer konstant für einen rascheler)
 iGrainDensMin = 60
 iGrainDensMax = 90

 ;verhältnis einblende zu gesamtdauer bei einzelnen raschelern
 iEnvRatioMin = .1
 iEnvRatioMax = .2

 ;verhältnis realdauer eines raschelers zur dauer im rtm (2 doppelt so lang)
 iRaschDurFacMin = 2
 iRaschDurFacMax = 2.5

 ;kurvernform für ansteigen und abfallen des raschelns (beides positiv, größer=konvexer)
 iTypeIn = 1
 iTypeOut = 6

 ;panning (später vielleicht anders)
 kPan randomi .1, .9, 1/3, 3

 ;index in gkRaschPartialsAlle
 iPartials = 2

 //wenn ein neues ereignis kommt
 if kTime <= 0 then

  //werte bestimmen
  kPitch = iPitch + rand:k(iPitchMaxDev)
  kDb = iDb + rand:k(iDbMaxDev)
  kEnvRatio random iEnvRatioMin, iEnvRatioMax
  kGrainDens random iGrainDensMin, iGrainDensMax

  //spezielle werte für kurzen ton
  if kTon < kSeqToene then

   kOffset random 0, kKurz * (gkRaschRtmWackeln/100)
   kDur = kKurz * random:k(iRaschDurFacMin,iRaschDurFacMax) * (60/gkRaschRtmTempo)
   kTon += 1
   kTime = kKurz * (60/gkRaschRtmTempo)

   event "i", "EinRaschler", kOffset, kDur, kPitch, kDb, kPan,
              kEnvRatio, iQ_min, iQ_max, kGrainDens,
              iTypeIn, iTypeOut, iPartials

  //für langen ton
  else

   kOffset random 0, kKurz * (gkRaschRtmWackeln/100) ;wirklich
   kDur = kLang * random:k(iRaschDurFacMin,iRaschDurFacMax) * (60/gkRaschRtmTempo)

   event "i", "EinRaschler", kOffset, kDur, kPitch, kDb, kPan,
              kEnvRatio, iQ_min, iQ_max, kGrainDens,
              iTypeIn, iTypeOut, iPartials

   turnoff

  endif

 endif

 kTime -= 1/kr

endin
```

```
instr Z7_tief
//für die tiefen und immer längeren töne am schluss

 //tonhöhe und abweichung
 iPitch = 58
 iPitchMaxDev = 1

 //lautstärke und abweichung
 iDb = p4
 iDbMaxDev = 2

 ;q
 iQ_min = 3
 iQ_max = 7

 ;grain density (immer konstant für einen rascheler)
 iGrainDensMin = 60
 iGrainDensMax = 90

 ;verhältnis einblende zu gesamtdauer bei einzelnen raschelern
 iEnvRatioMin = .3
 iEnvRatioMax = .5

 ;verhältnis realdauer eines raschelers zur dauer im rtm (2 doppelt so lang)
 iRaschDurFacMin = 2
 iRaschDurFacMax = 2.5

 ;kurvernform für ansteigen und abfallen des raschelns (beides positiv, größer=konvexer)
 iTypeIn = 1
 iTypeOut = 6

 ;panning (später vielleicht anders)
 iPan random 1/4, 3/4

 ;index in gkRaschPartialsAlle
 iPartials = 2

 iPitch += rnd31:i(iPitchMaxDev,0)
 iDb += rnd31:i(iDbMaxDev,0)
 iEnvRatio random iEnvRatioMin, iEnvRatioMax
 iGrainDens random iGrainDensMin, iGrainDensMax


 schedule "EinRaschler", 0, p3, iPitch, iDb, iPan,
          iEnvRatio, iQ_min, iQ_max, iGrainDens,
          iTypeIn, iTypeOut, iPartials

 turnoff

endin


instr Z8

/* input */

 kPauseMin = 2 ;sekunden
 kPauseMax = 5
 kTempo = gkRaschRtmTempo
 kWackeln = gkRaschRtmWackeln
 kDb[] fillarray giDb_Z8, giDb_Z8+10, giDb_Z8-4
 kDbMaxDev = 4
 kPitch[] fillarray 93, 94, 92
 kPitchMaxDev = 1 ;halbtöne

 ;abfolge aus raschel-rhythmen
 kSeq[] fillarray 6, 0, 13

 ;q
 iQ_min = 1
 iQ_max = 2

 ;grain density (immer konstant für einen rascheler)
 iGrainDensMin = 40
 iGrainDensMax = 90

 ;verhältnis einblende zu gesamtdauer bei einzelnen raschelern
 iEnvRatioMin = .2
 iEnvRatioMax = .4

 ;verhältnis realdauer eines raschelers zur dauer im rtm (2 doppelt so lang)
 iRaschDurFacMin = 2
 iRaschDurFacMax = 3

 ;kurvernform für ansteigen und abfallen des raschelns (beides positiv, größer=konvexer)
 iTypeIn = 1
 iTypeOut = 6

 ;index in gkRaschPartialsAlle
 iPartials = 4

/* durchführung */

 ;uhr initialisieren (in k-zyklen)
 kUhr_k init 0

 ;nächste sequenz initalisieren (1=ja)
 kPlaySeq init 1

 ;dauer der sequenz in k-zyklen (wird unten gleich überschrieben)
 kSeqDur_k init 0

 ;index in kSeq, gleich auf loop hin
 kIndx init 0
 kIndx = kIndx % lenarray(kSeq)
```

```
 ;index des raschel-rtm in gkRaschRtmAlle
 kRaschRtmIndex = kSeq[kIndx]

 ;wenn eine sequenz ausgelöst werden soll
 if kPlaySeq == 1 then

  ;ermittele die zeit für das nächste auslösen
  kRtmTimUnits = gkRaschRtmAlle[kRaschRtmIndex][1] ;in zeiteinheiten
  kRtmTimSecs = (kRtmTimUnits / 4) * (60 / kTempo)
  kPause random kPauseMin, kPauseMax
  kSeqDur_k = round((kRtmTimSecs+kPause)*kr)

  ;ruf das instrument
  event "i", "RaschSeq", 0, kRtmTimSecs, kRaschRtmIndex, kTempo, kWackeln,
        kDb[0]+random:k(0,kDbMaxDev), kDb[1]+random:k(0,kDbMaxDev), kDb[2]+random:k(0,kDbMaxDev),
        kPitch[0]+random:k(0,kPitchMaxDev), kPitch[1]+random:k(0,kPitchMaxDev), kPitch[2]+random:k(0,kPitchMaxDev),
        iQ_min, iQ_max, iGrainDensMin, iGrainDensMax, iEnvRatioMin, iEnvRatioMax, iRaschDurFacMin, iRaschDurFacMax,
        iTypeIn, iTypeOut, iPartials, 1

  ;uhr zurücksetzen
  kUhr_k = 0

 endif

 ;schalter um sequenz auszulösen
 if kUhr_k == kSeqDur_k then
  kPlaySeq = 1
  kIndx += 1
 else
  kPlaySeq = 0
 endif

 ;zeit hoppelt weiter
 kUhr_k += 1

endin
maxalloc "Z8", 1




/*****************************************************************/
/************************ RASCHELN ***************************/
/*****************************************************************/



instr RaschSeq
//realisiert eine raschel sequenz vom typ R
//ruft unterinstrument das einen rascheler vollführt


/* input */

 ;array als zeile in gkRaschRtmAlle
 iArrIndex = p4
 kRtmArr[] getrow gkRaschRtmAlle, iArrIndex

 ;tempo (60 heisst das eine einheit in einem der input-arrays 1/4 sekunde ist)
 iTempo = p5

 ;wackeln des rtm, als mögliches offset (0..100%) zum nächsten wert realisiert
 iWackeln = p6

 ;db am anfang, in der mitte, und am ende der rtm sequenz
 iDbAnf = p7
 iDbMit = p8
 iDbEnd = p9

 ;tonhöhe in midi noten (anfang, mitte, ende)
 iPitchAnf = p10
 iPitchMit = p11
 iPitchEnd = p12

 ;q
 iQ_min = p13
 iQ_max = p14

 ;grain density
 iGrainDensMin = p15
 iGrainDensMax = p16

 ;verhältnis einblende zu gesamtdauer bei einzelnen raschelern
 kEnvRatioMin = p17
 kEnvRatioMax = p18

 ;verhältnis realdauer eines raschelers zur dauer im rtm (2 doppelt so lang)
 iRaschDurFacMin = p19
 iRaschDurFacMax = p20

 ;kurvernform für ansteigen und abfallen des raschelns (beides positiv, größer=konvexer)
 iTypeIn = p21
 iTypeOut = p22

 ;index in gkRaschPartialsAlle
 iPartials = p23

 ;eventuelle auslassung von tönen (1 = alle werden gespielt, 2 = im schnitt jeder zweite, etc)
 kAuslassung = gkRaschRtmAuslassungen

 ;welches instrument soll gerufen werden
 S_RaschelInstr strget p24
```

```
/* ditt un datt */

 ;länge der sequenz in noten
 kRtmSeqNumNotes = kRtmArr[0]

 ;dynamischer höhepunkt (menschliche zählweise: 3 = dritte note)
 kRtmSeqGipfel = kRtmArr[2]

 ;index im array für den trigger
 kIndx init 3

 ;wieviele sekunden ist eine zeiteinheit
 iZeiteinheitSec = (60/iTempo) / 4

 ;trigger setzen für das auslösen des raschel instruments
 kTrig init 1

 ;zeit in k-zyklen zum inkrementieren
 kZeit_k init 0

 ;bei welchem ereignis sind wir (menschliche zählung: 1, 2, ...)
 kEreignis init 1

 ;panning (später vielleicht anders)
 kPan randomi .1, .9, 1/10, 3


/* auslösen */

 if kTrig == 1 then

  ;db kalkulieren
  kDb Drei iDbAnf, iDbMit, iDbEnd, kRtmSeqNumNotes, kRtmSeqGipfel, kEreignis

  ;pitch kalkulieren
  kPitch Drei iPitchAnf, iPitchMit, iPitchEnd, kRtmSeqNumNotes, kRtmSeqGipfel, kEreignis

  ;dauer dieses ereignisses (sec)
  kDur = kRtmArr[kIndx] * iZeiteinheitSec

  ;offset kalkulieren
  kOffset random 0, kDur * (iWackeln/100)

  ;schauen ob dieser ton nicht unter den tisch fällt
  kWurf random 0, 1
  kLeiderNicht = (kWurf > 1/kAuslassung) ? 1 : 0

  ;instrument rufen wenn ton nicht ausgelassen wird
  if kLeiderNicht == 0 then

   kGrainDens random iGrainDensMin, iGrainDensMax
   kRaschDurFac random iRaschDurFacMin, iRaschDurFacMax

   event "i", S_RaschelInstr, kOffset, kDur*kRaschDurFac, kPitch, kDb, kPan,
              random:k(kEnvRatioMin,kEnvRatioMax), iQ_min, iQ_max, kGrainDens,
              iTypeIn, iTypeOut, iPartials

  endif

  ;trigger zurücksetzen
  kTrig = 0

  ;index heraufsetzen
  kIndx += 1

  ;ereignis auch
  kEreignis += 1

  ;zeit zurücksetzen
  kZeit_k = 0

  ;nächsten zeitpunkt berechnen (in k-zyklen)
  kNextZeit_k = round(kDur*kr)

 endif


/* nächsten trigger auslösen */

 if kZeit_k == kNextZeit_k then
   kTrig = 1
 endif

 ;k zyklus zählung heraufsetzen
 kZeit_k += 1

endin


instr RaschSeq_Z2
//macht zusätzlich zur normalen raschel-sequenz noch einen kleinen piekser
//wenn T1 fertig ist, in ansteigender lautstärke


/* input */

 ;array als zeile in gkRaschRtmAlle
 iArrIndex = p4
 kRtmArr[] getrow gkRaschRtmAlle, iArrIndex

 ;tempo (60 heisst das eine einheit in einem der input-arrays 1/4 sekunde ist)
 iTempo = p5
```

```
 ;wackeln des rtm, als mögliches offset (0..100%) zum nächsten wert realisiert
 iWackeln = p6

 ;db am anfang, in der mitte, und am ende der rtm sequenz
 iDbAnf = p7
 iDbMit = p8
 iDbEnd = p9

 ;tonhöhe in midi noten (anfang, mitte, ende)
 iPitchAnf = p10
 iPitchMit = p11
 iPitchEnd = p12

 ;q
 iQ_min = p13
 iQ_max = p14

 ;grain density
 iGrainDensMin = p15
 iGrainDensMax = p16

 ;verhältnis einblende zu gesamtdauer bei einzelnen raschelern
 kEnvRatioMin = p17
 kEnvRatioMax = p18

 ;verhältnis realdauer eines raschelers zur dauer im rtm (2 doppelt so lang)
 iRaschDurFacMin = p19
 iRaschDurFacMax = p20

 ;kurvernform für ansteigen und abfallen des raschelns (beides positiv, größer=konvexer)
 iTypeIn = p21
 iTypeOut = p22

 ;index in gkRaschPartialsAlle
 iPartials = p23

 ;eventuelle auslassung von tönen (1 = alle werden gespielt, 2 = im schnitt jeder zweite, etc)
 kAuslassung = gkRaschRtmAuslassungen

 ;welches instrument soll gerufen werden
 S_RaschelInstr strget p24

 ;welche sequenz nach dem ende von T1
 iSeqAfterT1 = p25


/* ditt un datt */

 ;länge der sequenz in noten
 kRtmSeqNumNotes = kRtmArr[0]

 ;dynamischer höhepunkt (menschliche zählweise: 3 = dritte note)
 kRtmSeqGipfel = kRtmArr[2]

 ;index im array für den trigger
 kIndx init 3

 ;wieviele sekunden ist eine zeiteinheit
 iZeiteinheitSec = (60/iTempo) / 4

 ;trigger setzen für das auslösen des raschel instruments
 kTrig init 1

 ;zeit in k-zyklen zum inkrementieren
 kZeit_k init 0

 ;bei welchem ereignis sind wir (menschliche zählung: 1, 2, ...)
 kEreignis init 1

 ;panning (später vielleicht anders)
 kPan randomi .1, .9, 1/10, 3


/* auslösen */

 if kTrig == 1 then

  ;db kalkulieren
  kDb Drei iDbAnf, iDbMit, iDbEnd, kRtmSeqNumNotes, kRtmSeqGipfel, kEreignis

  ;pitch kalkulieren
  kPitch Drei iPitchAnf, iPitchMit, iPitchEnd, kRtmSeqNumNotes, kRtmSeqGipfel, kEreignis

  ;dauer dieses ereignisses (sec)
  kDur = kRtmArr[kIndx] * iZeiteinheitSec

  ;offset kalkulieren
  kOffset random 0, kDur * (iWackeln/100)

  ;schauen ob dieser ton nicht unter den tisch fällt
  kWurf random 0, 1
  kLeiderNicht = (kWurf > 1/kAuslassung) ? 1 : 0

  ;instrument rufen wenn ton nicht ausgelassen wird
  if kLeiderNicht == 0 then

   kGrainDens random iGrainDensMin, iGrainDensMax
   kRaschDurFac random iRaschDurFacMin, iRaschDurFacMax
   kEnvRatio random kEnvRatioMin, kEnvRatioMax

   event "i", S_RaschelInstr, kOffset, kDur*kRaschDurFac, kPitch, kDb, kPan,
              kEnvRatio, iQ_min, iQ_max, kGrainDens,
              iTypeIn, iTypeOut, iPartials

   if gkT1_End == 1 then
```

```
    kQ_piekser random iQ_max*iSeqAfterT1/2, iQ_max*iSeqAfterT1
    if random:k(0,1) < iSeqAfterT1/10 then
     event "i", "EinfachArpegg", kOffset+kDur*kRaschDurFac*kEnvRatio, 1, random:k(kDb-10,kDb), 10, kPitch, kQ_piekser, kPan
    endif
   endif

  endif

  ;trigger zurücksetzen
  kTrig = 0

  ;index heraufsetzen
  kIndx += 1

  ;ereignis auch
  kEreignis += 1

  ;zeit zurücksetzen
  kZeit_k = 0

  ;nächsten zeitpunkt berechnen (in k-zyklen)
  kNextZeit_k = round(kDur*kr)

 endif


/* nächsten trigger auslösen */

 if kZeit_k == kNextZeit_k then
   kTrig = 1
 endif

 ;k zyklus zählung heraufsetzen
 kZeit_k += 1

endin



instr HakSeq
//realisiert eine raschel sequenz vom typ H
//ruft unterinstrument das einen rascheler vollführt

 ;bekomme index in gkRaschRtmAlle
 iRtmIndex = p4

 ;erzeuge daraus die beiden arrays
 kPhase_1[] getrow gkRaschRtmAlle, iRtmIndex
 kPhase_2[] getrow gkRaschRtmAlle, iRtmIndex+1

 ;tempo (60 heisst das eine einheit in einem der input-arrays 1/4 sekunde ist)
 iTempo = p5

 ;wackeln des rtm, als mögliches offset (0..100%) zum nächsten wert realisiert
 iWackeln = p6

 ;db am anfang, am ende von phase 2, und beim langen ton
 iDbAnf = p7
 iDbMit = p8
 iDbEnd = p9

 ;tonhöhe in midi noten (phase 1, phase 2, langer ton)
 iPitchAnf = p10
 iPitchMit = p11
 iPitchEnd = p12

 ;dauer des langen tons
 iDurLangerTon = p13

 ;q
 iQ_min = p14
 iQ_max = p15

 ;grain density
 iGrainDensMin = p16
 iGrainDensMax = p17

 ;verhältnis realdauer eines raschelers zur dauer im rtm (2 doppelt so lang)
 iRaschDurFacMin = p18
 iRaschDurFacMax = p19

 ;eventuelle auslassung von tönen (1 = alle werden gespielt, 2 = im schnitt jeder zweite, etc)
 kAuslassung = gkRaschRtmAuslassungen

 ;welches instrument soll gerufen werden
 S_RaschelInstr = "EinRaschler"

 ;panning (später vielleicht anders)
 kPan randomi .1, .9, 1/10, 3

 ;verhältnis einblende zu gesamtdauer bei einzelnen raschelern
 kEnvRatioMin = p20
 kEnvRatioMax = p21

 ;kurvernform für ansteigen und abfallen des raschelns (beides positiv, größer=konvexer)
 iTypeIn = p22
 iTypeOut = p23

 ;index in gkRaschPartialsAlle
 iPartials = p24

 //phase 1

  ;wie lang
  kNumNotesPhas_1 = kPhase_1[0]
```

```
 ;anfangsindex und -offset
 kIndx = 2
 kOffset = 0

 ;erzeuge events und aktualisiere offset
 while kIndx < kNumNotesPhas_1+2 do

  ;dauer des ereignisses
  kDur = (kPhase_1[kIndx]/4) * (60/iTempo)

  ;tonhöhe und db
  kPitch = iPitchAnf + rand:k(2/3,0,1)
  kDb = iDbAnf + rand:k(3,0,1)

  ;schauen ob dieser ton nicht unter den tisch fällt
  kWurf random 0, 1
  kLeiderNicht = (kWurf > 1/kAuslassung) ? 1 : 0

  ;instrument rufen wenn ton nicht ausgelassen wird
  if kLeiderNicht == 0 then
   kGrainDens random iGrainDensMin, iGrainDensMax
   kRaschDurFac random iRaschDurFacMin, iRaschDurFacMax
   event "i", S_RaschelInstr, kOffset, kDur*kRaschDurFac, kPitch, kDb, kPan,
              random:k(kEnvRatioMin,kEnvRatioMax), iQ_min, iQ_max, kGrainDens,
              iTypeIn, iTypeOut, iPartials
  endif

  kOffset += kDur
  kIndx += 1

 od

//phase 2

 ;länge
 kNumNotesPhas_2 = kPhase_2[0]

 ;index zurücksetzen
 kIndx = 2

 ;anfangswert für db
 kDbPhas_2 init iDbAnf

 ;events
 while kIndx < kNumNotesPhas_2+2 do

  ;crescendo erzeugen
  iDbDiff = iDbMit - iDbAnf
  kDbPhas_2 += iDbDiff / kNumNotesPhas_2

  ;dauer dieses ereignisses
  kDur = (kPhase_2[kIndx]/4) * (60/iTempo)

  ;tonhöhe hier erstmal ganz einfach
  kPitch = iPitchMit

  ;schauen ob dieser ton nicht unter den tisch fällt
  kWurf random 0, 1
  kLeiderNicht = (kWurf > 1/kAuslassung) ? 1 : 0

  ;instrument rufen wenn ton nicht ausgelassen wird
  if kLeiderNicht == 0 then
   kGrainDens random iGrainDensMin, iGrainDensMax
   kRaschDurFac random iRaschDurFacMin, iRaschDurFacMax
   event "i", S_RaschelInstr, kOffset, kDur*kRaschDurFac, kPitch, kDbPhas_2, kPan,
              random:k(kEnvRatioMin,kEnvRatioMax), iQ_min, iQ_max, kGrainDens,
              iTypeIn, iTypeOut, iPartials
  endif

  kOffset += kDur
  kIndx += 1

 od

//langer ton (nicht von auslassung betroffen)
kGrainDens random iGrainDensMin, iGrainDensMax
event "i", S_RaschelInstr, kOffset, iDurLangerTon, iPitchEnd, iDbEnd, kPan,
           random:k(kEnvRatioMin,kEnvRatioMax), iQ_min, iQ_max, kGrainDens,
           iTypeIn, iTypeOut, iPartials

;und tschüss
turnoff

endin


instr EinRaschler

 ;tonhöhe (midi noten) und lautstärke empfangen
 iMidiPitch = p4
 idB = p5

 ;grain density (hz)
 iGrainDens = p10
 kGrainDens init iGrainDens

 ;grain dauer (sec)
 kGrainDurMin = 60/1000
 kGrainDurMax = 80/1000

 ;rauschdauer
 iNoiseDur = p3

 ;zentralfrequenz
```

```
 iCf = cpsmidinn(iMidiPitch)

 ;panning
 iPan = p6
 iPanMaxDev = 0.05

 ;filter q
 kQ_min = p8
 kQ_max = p9

 ;mögliches grain offset (ratio)
 kGrainMaxOffset = 1

 ;verhältnis einblende zur gesamtdauer
 iFadeRatio = p7

 ;daraus reale zeiten
 iFadeIn = iFadeRatio * iNoiseDur
 iFadeOut = iNoiseDur - iFadeIn

 ;hüllkurve
 iTypeIn = p11
 iTypeOut = p12
 kGrainAmp transeg 0, iFadeIn, iTypeIn, ampdb(idB), iFadeOut, -iTypeOut, 0
 kGrainAmp *= ampdb(gkRaschVolDbIntern)

 ;array mit teiltönen
 iArrIndex = p13
 kPartials[] getrow gkRaschPartialsAlle, iArrIndex
 kPartialsLen = kPartials[0]

 ;grains rufen
 kTrig metro kGrainDens

 if kTrig == 1 then

  kGrainDur random kGrainDurMin, kGrainDurMax
  kQ random kQ_min, kQ_max
  kPan random iPan-iPanMaxDev, iPan+iPanMaxDev
  kIndx = 1
  while kIndx < kPartialsLen+1 do
   event "i", "GrainRasch", random:k(0, kGrainDurMin*kGrainMaxOffset), kGrainDur,
                        kGrainAmp/kPartials[kIndx], iCf*kPartials[kIndx], kQ, kPan
   kIndx += 1
  od

 endif

endin


instr EinRaschlerZ5
//mit veränderbaren lautstärken für die raschel-teiltöne

 ;tonhöhe (midi noten) und lautstärke empfangen
 iMidiPitch = p4
 idB = p5

 ;grain density (hz)
 iGrainDens = p10
 kGrainDens init iGrainDens

 ;grain dauer (sec)
 kGrainDurMin = 60/1000
 kGrainDurMax = 80/1000

 ;rauschdauer
 iNoiseDur = p3

 ;zentralfrequenz
 iCf = cpsmidinn(iMidiPitch)

 ;panning
 iPan = p6
 iPanMaxDev = 0.05

 ;filter q
 kQ_min = p8
 kQ_max = p9

 ;mögliches grain offset (ratio)
 kGrainMaxOffset = 1

 ;verhältnis einblende zur gesamtdauer
 iFadeRatio = p7

 ;daraus reale zeiten
 iFadeIn = iFadeRatio * iNoiseDur
 iFadeOut = iNoiseDur - iFadeIn

 ;hüllkurve
 iTypeIn = p11
 iTypeOut = p12
 kGrainAmp transeg 0, iFadeIn, iTypeIn, ampdb(idB), iFadeOut, -iTypeOut, 0
 kGrainAmp *= ampdb(gkRaschVolDbIntern)

 ;array mit teiltönen
 iArrIndex = p13
 kPartials[] getrow gkRaschPartialsAlle, iArrIndex
 kPartialsLen = kPartials[0]

 ;grains rufen
 kTrig metro kGrainDens

 if kTrig == 1 then
```

```
  kGrainDur random kGrainDurMin, kGrainDurMax
  kQ random kQ_min, kQ_max
  kPan random iPan-iPanMaxDev, iPan+iPanMaxDev

  event "i", "GrainRasch", random:k(0, kGrainDurMin*kGrainMaxOffset), kGrainDur,
                        kGrainAmp/kPartials[1]*ampdb(gkZ5_db_TT1), iCf*kPartials[1], kQ, kPan
  event "i", "GrainRasch", random:k(0, kGrainDurMin*kGrainMaxOffset), kGrainDur,
                        kGrainAmp/kPartials[2]*ampdb(gkZ5_db_TT2), iCf*kPartials[2], kQ, kPan
  event "i", "GrainRasch", random:k(0, kGrainDurMin*kGrainMaxOffset), kGrainDur,
                        kGrainAmp/kPartials[3]*ampdb(gkZ5_db_TT3), iCf*kPartials[3], kQ, kPan
  event "i", "GrainRasch", random:k(0, kGrainDurMin*kGrainMaxOffset), kGrainDur,
                        kGrainAmp/kPartials[4]*ampdb(gkZ5_db_TT4), iCf*kPartials[4], kQ, kPan

 endif

endin



/*****************************************************************/
/************************ ARPEGGIATO *************************/
/*****************************************************************/


instr T1
//richard

 kIndex init 0

 //tastatur (leertaste oder "n")
 if changed(gkKey) == 1 then
  if gkKey == 32 || gkKey == 110 then
   kTrig = 1
  else
   kTrig = 0
  endif
 endif

 //bei taste "n" vorigen ton nochmal spielen
 if gkKey == 110 then
  kIndex -= 1
 endif
 kIndex limit kIndex, 0, 999

 ;liest immer den nächsten ton im array
 if kTrig == 1 then

  kPitch = gkT1_Pitches[kIndex]
  kDur random 1, 5
  event "i", "T1_Ton", 0, kDur, kPitch

  ;wenn letzter ton des ersten akkordes: Z1 ausschalten
  if kIndex == 4 then
   event "i", "Z1_Stop", 0, 1
  endif

  ;wenn erster ton des dritten akkordes: Z2 anschalten
  if kIndex == 14 then
   event "i", "Z2_Start", 0, 1
  endif

   kIndex += 1

 endif

endin
maxalloc "T1", 1



instr T1_Ton

 ;tonhöhe (midi noten) und lautstärke empfangen
 iMidiPitch = p4
 idB random -36, -26

 ;grain density (hz)
 iGrainDens random 30, 90
 kGrainDens init iGrainDens

 ;grain dauer (sec)
 kGrainDurMin = 60/1000
 kGrainDurMax = 80/1000

 ;rauschdauer
 iNoiseDur = p3

 ;zentralfrequenz
 iCf = cpsmidinn(iMidiPitch)

 ;panning
 iPan random .1, .9
 iPanMaxDev = 0.05

 ;filter q
 kQ_min = 10
 kQ_max = 50

 ;mögliches grain offset (ratio)
 kGrainMaxOffset = 1

 ;verhältnis einblende zur gesamtdauer
 iFadeRatio random 1/10, 1/4
```

```
 ;daraus reale zeiten
 iFadeIn = iFadeRatio * iNoiseDur
 iFadeOut = iNoiseDur - iFadeIn

 ;hüllkurve
 iTypeIn = 3
 iTypeOut = 3
 kGrainAmp transeg 0, iFadeIn, iTypeIn, ampdb(idB), iFadeOut, -iTypeOut, 0
 kGrainAmp *= ampdb(gkArpeggVolDbIntern)

 ;grains rufen
 kTrig metro kGrainDens

 if kTrig == 1 then

  kGrainDur random kGrainDurMin, kGrainDurMax
  kQ random kQ_min, kQ_max
  kPan random iPan-iPanMaxDev, iPan+iPanMaxDev
  event "i", "GrainArpegg", random:k(0, kGrainDurMin*kGrainMaxOffset), kGrainDur, kGrainAmp, iCf, kQ, kPan
  event "i", "GrainArpegg", random:k(0, kGrainDurMin*kGrainMaxOffset), kGrainDur, kGrainAmp/3, iCf*3, kQ, kPan
  event "i", "GrainArpegg", random:k(0, kGrainDurMin*kGrainMaxOffset), kGrainDur, kGrainAmp/4, iCf*4, kQ, kPan

 endif

endin


instr T2
//personenschaden

 //empfang der basstoene
 if changed(gkKey) == 1 then
  if gkKey == 32 then
   kNeuerHauptton = 1
  elseif gkKey == 110 then
   kHaupttonRep = 1
  else
   kNeuerHauptton = 0
   kHaupttonRep = 0
  endif
 endif

 //array mit der tonfolge
 kToene[] fillarray 76, 74, 71, 68, 65, 63, 60, 55

 //zugeordnete basstöne
 kBasstoene[] fillarray 52, 50, 47, 44, 41, 39, 36, 31

 //transposition (halbtöne)
 iTransp = 36

 //mögliche q faktoren
 kQVorrat[] fillarray 5, 10, 20, 40, 80, 160
 kQIndxMin linseg 0, 50, 2, 50, 0
 kQIndxMax linseg 1, 30, 5, 30, 5, 30, 3

 //grundlautstärke
 iBasDb = -12

 //tonbewegung
 iCentDir = 50 ;standardschritt ist 50 cent
 kCentDir init iCentDir
 kCentDev init 0 ;ich glaube ist besser ohne
 kPrevTon init 76+iTransp
 kNeuerTon init 76+iTransp
 kLimitTon init 83+iTransp

 //tim intv grenzen für knuth
 kKnuthTimIntv[] fillarray 1/7, 1/5, 1/3

 //bewegung desselben (anfang und ende mehr pausen)
 kKnuthTimIntvAdd linseg 1, 30, 0, 30, 0, 30, 1

 //haupttöne immer abschnitte aus kToene von 2-4 tönen
 kNeueSequenz init 0

 //wenn zäsur:
 if kNeuerHauptton == 1 then

  //neuen anfangton holen falls neue sequenz
  if kNeueSequenz == 0 then
   kLenSeq = int(random:k(2,4.999))
   kIndexToene = int(random:k(0,lenarray(kToene)-kLenSeq+.999))
   kNeueSequenz = kLenSeq

  //andernfalls die sequenz weiterführen
  else
   kIndexToene += 1

  endif

  kNeuerTon = kToene[kIndexToene] + iTransp
  kPrevTon = kNeuerTon
  //grenze setzen
  kLimitTon = kPrevTon + 7
  //richtung bestimmen (öfter hoch als runter)
  kCentDir = (random:k(0,1)>0.3) ? iCentDir : -iCentDir
  //tiefen ton spielen
  event "i", "T2_tief", 0, random:k(3,7), kBasstoene[kIndexToene]

  //sequenz anzeigen und weiterrücken
  chnset kNeueSequenz, "T2_seq_count"
  kNeueSequenz -= 1

 endif
```

```
 //repetition
 if kHaupttonRep == 1 then
  event "i", "T2_tief", 0, random:k(3,7), kBasstoene[kIndexToene]
 endif

 //wenn trigger von knuth, ton auslösen
 if gkKnuthTrigImp == 1 then

   kNoiseDurMs = 10
   kPitch Laufen kPrevTon, kCentDir, kLimitTon, kCentDev
   kQ = kQVorrat[int(random:k(kQIndxMin,kQIndxMax+.999))]
   kDb = iBasDb - kQ/20 //ungefährer ausgleich der q-faktoren
   kDbLinie = kLimitTon-kPitch
   kDb -= kDbLinie
   kPan = (kPitch-86) / 32

   event "i", "EinfachArpegg", 0, 5, kDb, kNoiseDurMs, kPitch, kQ, kPan

   kPrevTon = kPitch
   //richtung umkehren falls grenze überschritten
   kCentDir = (kPrevTon > kLimitTon) ? -kCentDir : kCentDir

   gkKnuthMinTimIntv ArrRndEl kKnuthTimIntv
   gkKnuthMinTimIntv += kKnuthTimIntvAdd

 endif

endin
maxalloc "T2", 1


instr T2_tief
//ein sehr tiefer ton

 ;laustärke
 idB random -30, -24

 ;rauschdauer
 iNoiseDur random 10/1000, 20/1000

 ;pitch
 iPch = p4
 ;iPch += random:i(0,1)

 ;qualität
 iQ random 400, 500

 ;panning
 iPan random .3, .7

 ;glissando maximale abweichung (1=halbton)
 iGlissMaxDev random 5, 10

 ;fade in
 iFadIn random .1, .5

 ;freq mit gliss
 kPch linseg iPch, p3, iPch-iGlissMaxDev

 ;sicherheit
 kFreq limit cpsmidinn(kPch), 20, 14000

 ;hüllkurve
 aEnv transeg ampdb(idB), iNoiseDur, -6, 0

 ;rauschen
 aNoise rand aEnv, 2, 1

 ;bandpass
 aFilt mode aNoise, kFreq, iQ

 ;zusätzliche ausblende
 aEnv linen aFilt, iFadIn, p3, p3

 ;panning
 aL, aR pan2 aEnv, iPan

 ;output
 chnmix aL, "collect_text_B_L"
 chnmix aR, "collect_text_B_R"

endin


instr T3
//westerland

 //tonreihe
 kToene[] fillarray 110, 104, 97, 103, 81, 107, 88, 74, 88, 106, 81, 103, 97, 74
 kToeneIndx init 0

 //mögliche intervalle innerhalb der welle
 kInterv[] fillarray 3, 5, 7, 10

 //dauernreihe
 kDauern[] fillarray 2, 3, 5, 1, 4
 kDauernIndx init 0

 //panning ist langsame bewegung
 kPan randomi .1, .9, 1/10, 3

 //wenn taste "n" das vorige nochmal spielen (nur bezogen auf pitch)
 if gkKey == 110 && changed(gkKey) == 1 then
```

```
  kToeneIndx = (kToeneIndx-1) % lenarray(kToene)
  gkKnuthTrigImp = 1 ;geht doch oder?
 endif

 //wenn ein impuls von Knuth_key kommt
 if gkKnuthTrigImp == 1 then
  //ton der dauernreihe
  kTon = kToene[kToeneIndx]
  //intervall zufällig
  kRange = kInterv[int(random:k(0,lenarray(kInterv)))]
  //dauer nach reihe
  kDur = kDauern[kDauernIndx]
  //rufe eine welle
  event "i", "T3_Welle", 0, kDur, kTon, kRange, kPan
  kToeneIndx = (kToeneIndx+1) % lenarray(kToene)
  kDauernIndx = (kDauernIndx+1) % lenarray(kDauern)
 endif

endin
maxalloc "T3", 1



instr T3_Welle

 //ton auf dem höhepunkt der welle (midi)
 iCenterPitch = p4

 //intervall (halbtöne), also töne vorne und hinten
 iRange = p5

 //kurve daraus
 kPitch linseg iCenterPitch-iRange, p3/2, iCenterPitch, p3/2, iCenterPitch-iRange

 //mögliche abweichungen davon durch ein grain (1=halbton)
 iGrainPitchMaxDev = 5

 //panning
 iPan = p6
 iPanMaxDev = 0.1

 //lautstärken min-max (db) auf dem höhepunkt
 iDbMin = -22
 iDbMax = -9
 iDb random iDbMin, iDbMax

 //kurve daraus
 iTypeIn random 0, 2
 iTypeOut random 3, 6
 iFadeIn random p3*1/4, p3*3/4
 iFadeOut = p3-iFadeIn
 kGrainAmp transeg 0, iFadeIn, iTypeIn, ampdb(iDb), iFadeOut, -iTypeOut, 0
 kGrainAmp *= ampdb(gkArpeggVolDbIntern)
 ;grain density (hz)
 kGrainDens randomi 60, 90, 1/5, 3

 ;grain dauer (sec)
 kGrainDurMin = 60/1000
 kGrainDurMax = 80/1000

 ;filter q
 kQ_min = 1
 kQ_max = 2

 ;mögliches grain offset (ratio)
 kGrainMaxOffset = 1

 ;array mit teiltönen
 iArrIndex = 3
 kPartials[] getrow gkRaschPartialsAlle, iArrIndex
 kPartialsLen = kPartials[0]

 ;grains rufen
 kTrig metro kGrainDens

 if kTrig == 1 then

  kGrainDur random kGrainDurMin, kGrainDurMax
  kQ random kQ_min, kQ_max
  kPan random iPan-iPanMaxDev, iPan+iPanMaxDev
  kGrainPitchDev random -iGrainPitchMaxDev, iGrainPitchMaxDev
  kCf = cpsmidinn(kPitch+kGrainPitchDev)
  kIndx = 1
  while kIndx < kPartialsLen+1 do
   event "i", "GrainArpegg", random:k(0, kGrainDurMin*kGrainMaxOffset), kGrainDur,
                        kGrainAmp/kPartials[kIndx], kCf*kPartials[kIndx], kQ, kPan
   kIndx += 1
  od

 endif

endin


instr T4

 iBasDb = -20

 kTrigIndx init 0

 //ungefähre dauern der strophen
 iDauern[] fillarray 8, 9, 16, 7, 20, 20 ;letzte strophe NACH "was hörst du"

 kAkk_1[] fillarray 52, 62, 70, 73
 kAkk_2[] fillarray 50, 61, 72, 76, 82
```

```
 kAkk_3[] fillarray 46, 64, 73, 75, 86
 kAkk_4[] fillarray 59, 61, 62, 76, 81
 kAkk_5[] fillarray 64, 65, 70, 73, 74
 kAkk_6[] fillarray 37, 64, 82, 98

 if gkKey == 32 && changed(gkKey) == 1 then

  if kTrigIndx == 0 then
   kIndx = 0
   while kIndx < lenarray(kAkk_1) do
    event "i", "T4_Ton", 0, iDauern[0], kAkk_1[kIndx], ampdb(iBasDb)/lenarray(kAkk_1), 1/4
    kIndx += 1
   od
   kTrigIndx += 1

  elseif kTrigIndx == 1 then
   kIndx = 0
   while kIndx < lenarray(kAkk_2) do
    event "i", "T4_Ton", 0, iDauern[1], kAkk_2[kIndx], ampdb(iBasDb-6)/lenarray(kAkk_2), 1/5
    kIndx += 1
   od
   kTrigIndx += 1

  elseif kTrigIndx == 2 then
   kIndx = 0
   while kIndx < lenarray(kAkk_3) do
    event "i", "T4_Ton", 0, iDauern[2], kAkk_3[kIndx], ampdb(iBasDb)/lenarray(kAkk_3), 1/7
    kIndx += 1
   od
   kTrigIndx += 1

  elseif kTrigIndx == 3 then
   kIndx = 0
   while kIndx < lenarray(kAkk_4) do
    event "i", "T4_Ton", 0, iDauern[3], kAkk_4[kIndx], ampdb(iBasDb-6)/lenarray(kAkk_4), 1/3
    kIndx += 1
   od
   kTrigIndx += 1

  elseif kTrigIndx == 4 then
   kIndx = 0
   while kIndx < lenarray(kAkk_5) do
    event "i", "T4_Ton", 0, iDauern[4], kAkk_5[kIndx], ampdb(iBasDb)/lenarray(kAkk_5), 1/9
    kIndx += 1
   od
   kTrigIndx += 1

  elseif kTrigIndx == 5 then
   kIndx = 0
   while kIndx < lenarray(kAkk_6) do
    event "i", "T4_Ton", 0, iDauern[5], kAkk_6[kIndx], ampdb(iBasDb-6)/lenarray(kAkk_6), 0
    kIndx += 1
   od
   kTrigIndx += 1

  endif

 endif

endin
maxalloc "T4", 1


instr T4_Ton

 iVibrDepth = p6 ;1 = halbton nach oben und unten

 aEnv transeg 0, p3/2, 3, p5, p3/2, -3, 0
 kVibr poscil iVibrDepth, p4/4
 aTon poscil aEnv, cpsmidinn(p4+kVibr)

 aL, aR pan2 aTon, random:i(.1, .9)

 chnmix aL, "collect_text_A_L"
 chnmix aR, "collect_text_A_R"

endin


instr T5

 //eine reihe von dauern (eigentlich einsatzabständen)
 kDauern[] fillarray 6, 9, 6, 7, 4, 6, 11, 5, 4, 9, 2, 8, 11

 //ein tempo für sie (als viertel, bezogen auf dauer 1 = 16tel)
 kTempo = 20

 //drei mögliche schichtfolgen
 kSchichtSeq_1[] fillarray 3, 1, 2, 4, 2, 3, 1, 3, 4, 2, 4, 3, 4
 kSchichtSeq_2[] fillarray 4, 2, 3, 1, 3, 2, 1, 4, 2, 3, 4, 3, 4
 kSchichtSeq_3[] fillarray 2, 4, 1, 3, 2, 1, 2, 4, 3, 2, 4, 3, 4

 //auswahl daraus
 iSchichtAuswahl random 0, 1
 if iSchichtAuswahl < 1/3 then
  kSchichtSeq[] = kSchichtSeq_1
 elseif iSchichtAuswahl < 2/3 then
  kSchichtSeq[] = kSchichtSeq_2
 else
  kSchichtSeq[] = kSchichtSeq_3
 endif

 //tonhöhen für die vier schichten
 kPitch[] fillarray 83, 102, 107, 114
```

```
 //db mittel für die vier schichten
 kDb[] fillarray -30, -20, -30, -20

 //q für die vier schichten
 kQ[] fillarray 5, 1, 10, 1

 //pan für die vier schichten
 kPan[] fillarray .2, .4, .6, .8

 //mögliche dauern eines tones
 kDauernPool[] fillarray 5, 8, 13, 21

 //aufwärtsglissandi (intervall in halbtönen)
 kPitchDev linseg 2, 60, 9

 //werte setzen
 kTime init 0
 kDurIndx init 0
 kSchichtIndx init 0
 kCount init 0

 //beim nächsten ereignis:
 if kTime <= 0 then

  //setze zeit bis zum nächsten ereignis
  kTime = kDauern[kDurIndx] * 60/(kTempo*4)

  //welche schicht ist es
  kSchicht = kSchichtSeq[kSchichtIndx]-1

  //dauer
  kDur ArrRndEl kDauernPool

  //instrument rufen
  event "i", "T5_Ton", 0, kDur, kPitch[kSchicht], kDb[kSchicht], kQ[kSchicht], kPan[kSchicht], kPitchDev

  //index und count aktualisieren
  kCount += 1
  kDurIndx = kCount % lenarray(kDauern)
  if kCount < lenarray(kSchichtSeq) then
   kSchichtIndx = kDurIndx
  else
   if kDurIndx % 2 == 1 then
    kSchichtIndx = lenarray(kSchichtSeq) - 2
   else
    kSchichtIndx = lenarray(kSchichtSeq) - 1
   endif
  endif

 endif

 //zeit zählen
 kTime -= 1/kr

endin
maxalloc "T5", 1


instr T5_Ton

 iPch random p4-1, p4+1
 iDb random p5-3, p5+3
 iQ = p6
 iPan = p7
 iPchDev = p8

 iAttTim random 5/1000, 10/1000
 iRisTim random p3/5, p3/3
 iDecayTim = p3-iAttTim-iRisTim

 aAmp transeg ampdb(iDb+15), iAttTim, -3, ampdb(iDb-40), iRisTim, 6, ampdb(iDb-20), iDecayTim, -6, 0
 aNoise rand aAmp, 2, 1
 kQ randomi iQ/5, iQ*5, 1/2, 2, iQ
 kPch linseg iPch, p3, iPch+iPchDev
 aTon mode aNoise, cpsmidinn(kPch), kQ

 aL, aR pan2 aTon, iPan

 chnmix aL, "collect_text_A_L"
 chnmix aR, "collect_text_A_R"

endin


instr T6_Cues
//reagiert auf die leertaste und setzt die cues für T6
//taste - (minus) geht einen zurück

 gkT6Cue init 0

 if changed(gkKey) == 1 then
  if gkKey == 32 then
   gkT6Cue += 1
  elseif gkKey == 45 then
   gkT6Cue -= 1
  endif
 endif

endin
maxalloc "T6_Cues", 1


instr T6
```

```
//yokohama

 //dritteltonskala
 kDrittelSkala[] genarray 0, 11.9, 2/3

 //indexreihen und transpositionen in den teilen
 kReihe_A[] fillarray 0, 1, 3, 5, 6, 8, 9, 11, 13, 15, 16
 kReihe_B[] fillarray 0, 2, 3, 5, 7, 8, 10, 12, 13, 15, 17
 iPitch_A = 76
 iPitch_B = 70

 //panning
 iPan_A = .5
 iPan_B = .5

 //index zurücksetzen wenn sich teil geändert hat
 if changed(gkT6Cue) == 1 then
  kIndx = 0

 //töne nach teil A und B beenden
  if gkT6Cue == 9 || gkT6Cue == 18 then
   turnoff2 "T6_Ton", 0, 1
  endif

 endif


 //impulse kommen von knuth und werden je nach cue verschieden behandelt
 if gkKnuthTrigImp == 1 then

  //TEIL A
  //Cue 1: nur die erste note spielen
  if gkT6Cue == 1 && kIndx < 1 then
   event "i", "T6_Ton", 0, 1, iPitch_A+kDrittelSkala[kIndx], iPan_A, kIndx

  //Cue 2: die ersten beiden
  elseif gkT6Cue == 2 && kIndx < 2 then
   event "i", "T6_Ton", 0, 1, iPitch_A+kDrittelSkala[kIndx], iPan_A, kIndx-1

  //Cue 3: die ersten vier
  elseif gkT6Cue == 3 && kIndx < 4 then
   event "i", "T6_Ton", 0, 1, iPitch_A+kDrittelSkala[kIndx], iPan_A, kIndx-3

  //Cue 4: die ersten sieben
  elseif gkT6Cue == 4 && kIndx < 7 then
   event "i", "T6_Ton", 0, 1, iPitch_A+kDrittelSkala[kIndx], iPan_A, kIndx-6

  //Cue 5: die ersten zehn
  elseif gkT6Cue == 5 && kIndx < 10 then
   event "i", "T6_Ton", 0, 1, iPitch_A+kDrittelSkala[kIndx], iPan_A, kIndx-9

  //Cue 6: die ersten 14
  elseif gkT6Cue == 6 && kIndx < 14 then
   event "i", "T6_Ton", 0, 1, iPitch_A+kDrittelSkala[kIndx], iPan_A, kIndx-13

  //Cue 7: die ersten 17
  elseif gkT6Cue == 7 && kIndx < 17 then
   event "i", "T6_Ton", 0, 1, iPitch_A+kDrittelSkala[kIndx], iPan_A, kIndx-16

  //Cue 8: leer


  //TEIL B
  //Cue 9: erster ton
  elseif gkT6Cue == 9 && kIndx < 1 then
   event "i", "T6_Ton", 0, 1, iPitch_B+kDrittelSkala[kIndx], iPan_B, kIndx

  //Cue 10: die ersten drei
  elseif gkT6Cue == 10 && kIndx < 3 then
   event "i", "T6_Ton", 0, 1, iPitch_B+kDrittelSkala[kIndx], iPan_B, kIndx-2

  //Cue 11: die ersten vier
  elseif gkT6Cue == 11 && kIndx < 4 then
   event "i", "T6_Ton", 0, 1, iPitch_B+kDrittelSkala[kIndx], iPan_B, kIndx-3

  //Cue 12: die ersten sechs
  elseif gkT6Cue == 12 && kIndx < 6 then
   event "i", "T6_Ton", 0, 1, iPitch_B+kDrittelSkala[kIndx], iPan_B, kIndx-5

  //Cue 13: die ersten neun
  elseif gkT6Cue == 13 && kIndx < 9 then
   event "i", "T6_Ton", 0, 1, iPitch_B+kDrittelSkala[kIndx], iPan_B, kIndx-8

  //Cue 14: die ersten 13
  elseif gkT6Cue == 14 && kIndx < 13 then
   event "i", "T6_Ton", 0, 1, iPitch_B+kDrittelSkala[kIndx], iPan_B, kIndx-12

  //Cue 15: die ersten 16
  elseif gkT6Cue == 15 && kIndx < 16 then
   event "i", "T6_Ton", 0, 1, iPitch_B+kDrittelSkala[kIndx], iPan_B, kIndx-15

  //Cue 16: die ersten 18
  elseif gkT6Cue == 16 && kIndx < 18 then
   event "i", "T6_Ton", 0, 1, iPitch_B+kDrittelSkala[kIndx], iPan_B, kIndx-17

  //Cue 17: leer

  endif

  kIndx += 1

 endif

 //TEIL C und D: töne werden über die cues ausgelöst
 if changed(gkT6Cue) == 1 then
```

```
  //C
  if gkT6Cue == 19 then
   event "i", "T6_Ton_CD", 0, 5, 74, .1, random:k(1,2)

  elseif gkT6Cue == 20 then
   turnoff2 "T6_Ton_CD", 0, 1

  elseif gkT6Cue == 21 then
   event "i", "T6_Ton_CD", 0, 5, 74.33, .1, random:k(1,2)

  elseif gkT6Cue == 22 then
   event "i", "T6_Ton_CD", 0, 5, 73, .2, random:k(1,2)

  elseif gkT6Cue == 23 then
   turnoff2 "T6_Ton_CD", 0, 1

  elseif gkT6Cue == 24 then
   event "i", "T6_Ton_CD", 0, 5, 74.667, .1, random:k(1,2)

  elseif gkT6Cue == 25 then
   event "i", "T6_Ton_CD", 0, 5, 72.667, .2, random:k(1,2)

  elseif gkT6Cue == 26 then
   event "i", "T6_Ton_CD", 0, 5, 70, .3, random:k(1,2)

  elseif gkT6Cue == 27 then
   turnoff2 "T6_Ton_CD", 0, 1

  //D
  elseif gkT6Cue == 28 then
   event "i", "T6_Ton_CD", 0, 15, 74, .8, random:k(1,2), 4

  elseif gkT6Cue == 29 then
   ;evt turnoff

  elseif gkT6Cue == 30 then
   event "i", "T6_Ton_CD", 0, 10, 72, .7, random:k(1,2), 4

  elseif gkT6Cue == 31 then
   turnoff2 "T6_Ton_CD", 0, 1

  endif

 endif

endin
maxalloc "T6", 1


instr T6_Ton
//für teil A und B

 //werte empfangen und setzen
 iPch = p4
 iPan = p5
 iIndex = p6 ;0 = erste note einer gruppe

 iCf = cpsmidinn(iPch)
 iCf limit iCf, 20, 14000

 iDbMin = -6
 iDbMax = 0
 iDbDiffLangerTon = 80
 iDb random iDbMin, iDbMax

 //LANGER LEISER TON
 if iIndex == 0 then

  p3 = 60
  iDbLang = iDb-iDbDiffLangerTon
  iNoiseDurLang = random:i(1,2)/1000
  aNoiseEnvLang transeg ampdb(iDbLang), iNoiseDurLang, -6, 0
  iQ_lang random 600, 1500
  aNoiseLang rand ampdb(iDbLang), 2, 1
  aFiltLang mode aNoiseLang, iCf, iQ_lang
  aOutLang linenr aFiltLang, random:i(1/5,1/3), random:i(.7,1), .01
  aL_lang, aR_lang pan2 aOutLang, iPan

 endif


 //KURZER TROCKENER TON

 iNoiseDur = 1/1000
 iQ random 20, 30
 aEnv transeg ampdb(iDb), iNoiseDur, -3, 0
 aNoise rand aEnv, 2, 1
 iCfHoch limit iCf*2, 20, 14000 ;oktave höher
 aFilt mode aNoise, iCfHoch, iQ
 aEnv linenr aFilt, 0, random:i(.1,.2), .01
 aL, aR pan2 aEnv, iPan

 ;output
 chnmix aL+aL_lang, "collect_text_A_L"
 chnmix aR+aR_lang, "collect_text_A_R"

endin


instr T6_Ton_CD
//für die letzten beiden strophen von yokohama

 //werte empfangen und setzen
 iPch = p4
```

```
 iPan = p5
 iVibrTim = p6/5 ;zeit in der das vibrato sich bewegt
 iFadeOutAdd = p7

 //vibrato: von groß-langsam zu klein+schnell
 kVibrFreq linseg 3, iVibrTim, 8
 kVibrPitch linseg 1, iVibrTim, 0
 kVibr poscil kVibrPitch, kVibrFreq
 kAusgleich linseg 1, iVibrTim, 0
 kPch = iPch + kVibr - kAusgleich

 iDbMin = -70
 iDbMax = -60
 iDb random iDbMin, iDbMax
 kDb linseg iDb, iVibrTim, iDb-10

 iNoiseDur = random:i(1,2)/1000
 iQ random 600, 1500
 aNoise rand ampdb(kDb), 2, 1

 aFilt mode aNoise, cpsmidinn(kPch), iQ
 aFilt8 mode aNoise, cpsmidinn(iPch+random:i(12.5,13.5)+kVibr/2), iQ/2
 aFilt12 mode aNoise, cpsmidinn(iPch+random:i(17,18.5)+kVibr/3), iQ/3

 aOut linenr aFilt+aFilt8/4+aFilt12/16, random:i(1/5,1/3), random:i(1,1.5)+iFadeOutAdd, .01
 kPan randomi iPan-.1, iPan+.1, 1, 2, iPan
 aL, aR pan2 aOut, kPan

 chnmix aL, "collect_text_A_L"
 chnmix aR, "collect_text_A_R"

endin


instr T7

 //leertaste triggert, minus (-) geht einen zurück, plus (+) geht einen vor
 kIndx init 0
 if changed(gkKey) == 1 then

  if gkKey == 45 then
   kIndx -= 1
  elseif gkKey == 43 then
   kIndx += 1

  elseif gkKey == 32 then

   if kIndx < 3 then
    event "i", "T7_A", 0, 1, kIndx+1

   elseif kIndx == 3 then
    event "i", "T7_B", 0, 1

   endif

   kIndx += 1

  endif

 endif

endin
maxalloc "T7", 1


instr T7_A
//trocken, mehrmals kurz hintereinander

 iAnzahlToene = p4

 kRtmA[] fillarray 1, 2, 5, -1 ;die letzte zahl ist egal
 kTempo = 60

 kQ_exps[] fillarray 6, 7, 8
 kThisQ_exps[] ArrPermRnd kQ_exps

 kRtmIndx init 0
 kOffset init 0

 while kRtmIndx < iAnzahlToene do

  kIndx = 0
  kDb random -30, -20
  kPan random 1/4, 3/4

  while kIndx < lenarray(gkT7_Pitchs) do
   event "i", "T7_A_Ton", kOffset, 1, gkT7_Pitchs[kIndx], kDb, kPan, kThisQ_exps[kRtmIndx]
   kIndx += 1
  od

  kOffset += (kRtmA[kRtmIndx] / 4) * (60/kTempo)
  kRtmIndx += 1

 od

 turnoff

endin

instr T7_B

 kIndx init 0
 kDur random 5, 6
 kDb = -50
```

```
 while kIndx < lenarray(gkT7_Pitchs) do

  kThisDb = (kIndx == 0) ? kDb-20 : kDb ;erster ton ist mir zu laut
  event "i", "T7_B_Ton", 0, kDur, gkT7_Pitchs[kIndx], ampdb(kThisDb), 2
  kIndx += 1

 od

 turnoff

endin

instr T7_A_Ton

 iPch = p4
 iPch += rnd31(1/2,0)
 iDb = p5
 iPan = p6
 iQ_exp = p7

 iNoiseDur = 1/1000
 aEnv transeg ampdb(iDb), iNoiseDur, -3, 0
 aNoise rand aEnv, 2, 1
 iCf limit cpsmidinn(iPch), 20, 14000
 aFilt mode aNoise, iCf, 2^iQ_exp
 aEnv linenr aFilt, 0, random:i(.1,.2), .01
 aL, aR pan2 aEnv, iPan+rnd31:i(.1,0)

 chnmix aL, "collect_text_A_L"
 chnmix aR, "collect_text_A_R"

endin

instr T7_B_Ton

 iPch = p4
 iAmp = p5
 iVibrDepth = p6 ;1 = halbton nach oben und unten

 kEnv transeg 0, p3/3, 3, iAmp, p3*2/3, -6, 0
 kEnv *= randomi:k(1,2,2)
 aNoise rand kEnv, 2, 1
 kQ_exp randomi 5, 7, 10
 kPchDev randomh -1/5, 2/5, 15
 aTon mode aNoise, cpsmidinn(iPch+kPchDev), 2^kQ_exp

 aL, aR pan2 aTon, random:i(.1, .9)

 chnmix aL, "collect_text_A_L"
 chnmix aR, "collect_text_A_R"

endin


instr T8_1
//erste schicht: lange töne ähnlich wie T1
//und auslösen der dritten schicht

 //tonhöhen
 kPch[] = gkT8_Pitchs

 //mögliche einsatzabstände (1 = viertel im tempo)
 kEinsatz[] fillarray 1, 1.5, 2, 2.5, 3, 4, 5

 //mögliche pausen (lasse ich erstmal weg, scheint nicht nötig)
 kPaus[] fillarray 1, 1.5, 2

 //mögliche dauern bezogen auf die einsatzabstände
 //die erste zeile bezieht sich auf den ersten index in iRtm[] usw
 kDauern[] fillarray 2, 2, 2,
                     2, 3, 2,
                     2, 3, 2,
                     2, 3, 4,
                     3, 4, 6,
                     3, 4, 6,
                     4, 6, 4

 //tempo als metronomzahl
 iTempo = 40

 //panning langsame bewegung
 kPan randomi .1, .9, 1/10, 3

 //lautstärkenbereich
 kDb linseg -44, 10, -34
 kDbMaxDev linseg 4, 10, 8

 //für den schluss
 kLetzteToene[] fillarray 58, 67

 //auf gehts
 gkT8PchIndx init 0
 kTimSpan_k init 0

 //immer wenn die zeit da ist das unterinstrument rufen
 if kTimSpan_k == 0 then

  //einsatzabstand für den nächsten ton
  kOffsetIndx random 0, lenarray(kEinsatz)-0.001
  kOffset = kEinsatz[kOffsetIndx]

  //je nachdem eine dauer
  kThisTimUnit ArrRndEl kDauern, kOffsetIndx*3, (kOffsetIndx+1)*3-1
  kThisDur = kThisTimUnit*(60/iTempo)
```

```
  //tonhöhe holen
  if gkT8PchIndx >= lenarray(kPch) then
   kThisPitch ArrRndEl kLetzteToene
   kOffset += 2
  else
   kThisPitch = kPch[gkT8PchIndx]
  endif

  //laustärke
  kThisDb = kDb + rnd31:k(kDbMaxDev,0)

  //instrument rufen und gkT8_pch setzen
  event "i", "T8_1_Ton", 0, kThisDur, kThisPitch, kPan, kThisDb

  //ggf schicht 3 auslösen
  ;if gkT8PchIndx == 5 || gkT8PchIndx == 10 || gkT8PchIndx == 14 || gkT8PchIndx == 19 then
  if gkT8PchIndx % 5 == 0 && gkT8PchIndx > 0 then
   event "i", "T8_3", 0, 1
  endif

  //werte setzen für nächsten durchlauf
  kOffsetSec = kOffset*(60/iTempo)
  kTimSpan_k = round(kOffsetSec*kr)
  gkT8PchIndx += 1

 endif

 kTimSpan_k -= 1


endin
maxalloc "T8_1", 1



instr T8_2
//T8 zweite schicht

 //mögliche abweichungen von der tonhöhe
 kPitchDev[] fillarray 0, 2/3, 4/3, -2/3, -4/3
 kTransp = 0 ;(gkT8_pch < 70) ? 12 : 0

 //mögliche q faktoren
 kQVorrat[] fillarray 5, 10, 20, 40

 //grundlautstärke
 iBasDb = -12

 //tim intv grenzen für knuth
 kKnuthTimIntv[] fillarray 1/3, 1, 3, 1/3

 //panning langsame eigenständige bewegung
 kPan randomi .1, .9, 1/10, 3

 //wenn knuth triggert
 if gkKnuthTrigImp == 1 then

  //nimm entweder den gegenwärtigen oder den vorigen oder den nächsten ton von gkT8_Pitchs
  kPchIndx = int(random:k(gkT8PchIndx-1, gkT8PchIndx+1))
  kPchIndx limit kPchIndx, 0, lenarray(gkT8_Pitchs)-1

  //mit abweichungen
  kPchDev ArrRndEl kPitchDev
  kPitch = gkT8_Pitchs[kPchIndx] + kPchDev + kTransp

  //anderes
  kNoiseDurMs = 10
  kQ ArrRndEl kQVorrat
  kDb = iBasDb - kQ/20 //ungefährer ausgleich der q-faktoren
  kDb += rand:k(6,2)
  event "i", "EinfachArpegg_B", 0, 5, kDb, kNoiseDurMs, kPitch, kQ, kPan

  //neue zeit für knuth
  gkKnuthMinTimIntv ArrRndEl kKnuthTimIntv

 endif

endin
maxalloc "T8_2", 1


instr T8_3
//glaston

 //mögliche stufen für teiltöne 2 und 3 (tt1 = 84)
 iTT2[] fillarray 96.667, 97.333, 98, 98.667, 99.333, 100, 100.667, 101.333
 iTT3[] fillarray 109, 109.4, 109.8, 110.2, 110.6, 111, 111.4

 //normalerweise ist fis (90) tonhöhe
 iTransp = -6

 //davon zufallsabweichung
 iPchRndDev rnd31 1, 0

 //dauer
 iDur random 7, 13

 //lautstärke
 iDb random -50, -40

 //vibratotiefe
 iVibrDepth random 1/10, 1/2

 //3 teiltöne auslösen
 schedule "T8_3_Ton", 0, iDur+rnd31:i(iDur/5,0), 90+iTransp+iPchRndDev, iDb+rnd31:i(4,0), iVibrDepth
```

```
 schedule "T8_3_Ton", 0, iDur+rnd31:i(iDur/5,0), ArrRndEl:i(iTT2)+iTransp+iPchRndDev, iDb+rnd31:i(4,0), iVibrDepth
 schedule "T8_3_Ton", 0, iDur+rnd31:i(iDur/5,0), ArrRndEl:i(iTT3)+iTransp+iPchRndDev, iDb+rnd31:i(4,0), iVibrDepth

 turnoff

endin
maxalloc "T8_3", 1


instr T8_1_Ton

 ;tonhöhe (midi noten) und lautstärke empfangen
 iMidiPitch = p4
 idB = p6


 ;grain density (hz)
 iGrainDens random 30, 90
 kGrainDens init iGrainDens

 ;grain dauer (sec)
 kGrainDurMin = 60/1000
 kGrainDurMax = 80/1000

 ;rauschdauer
 iNoiseDur = p3

 ;zentralfrequenz
 iCf = cpsmidinn(iMidiPitch)

 ;panning
 iPan = p5
 iPanMaxDev = 0.05

 ;filter q
 kQ_min = 10
 kQ_max = 50

 ;mögliches grain offset (ratio)
 kGrainMaxOffset = 1

 ;verhältnis einblende zur gesamtdauer
 iFadeRatio random 1/10, 1/4

 ;daraus reale zeiten
 iFadeIn = iFadeRatio * iNoiseDur
 iFadeOut = iNoiseDur - iFadeIn

 ;hüllkurve
 iTypeIn = 3
 iTypeOut = 3
 kGrainAmp transeg 0, iFadeIn, iTypeIn, ampdb(idB), iFadeOut, -iTypeOut, 0
 kGrainAmp *= ampdb(gkArpeggVolDbIntern)

 ;grains rufen
 kTrig metro kGrainDens

 if kTrig == 1 then

  kGrainDur random kGrainDurMin, kGrainDurMax
  kQ random kQ_min, kQ_max
  kPan random iPan-iPanMaxDev, iPan+iPanMaxDev
  event "i", "GrainArpegg", random:k(0, kGrainDurMin*kGrainMaxOffset), kGrainDur, kGrainAmp, iCf, kQ, kPan
  event "i", "GrainArpegg", random:k(0, kGrainDurMin*kGrainMaxOffset), kGrainDur, kGrainAmp/3, iCf*3, kQ, kPan
  event "i", "GrainArpegg", random:k(0, kGrainDurMin*kGrainMaxOffset), kGrainDur, kGrainAmp/4, iCf*4, kQ, kPan

 endif

endin


instr T8_3_Ton

 iPch = p4
 iDb = p5
 iVibrDepth = p6 ;1 = halbton nach oben und unten

 iRatio random 1/10, 1/5

 aEnv transeg 0, p3*iRatio, 3, ampdb(iDb), p3*(1-iRatio), -3, 0
 kVibr poscil iVibrDepth, iPch/4
 aTon poscil aEnv, cpsmidinn(iPch+kVibr)

 aL, aR pan2 aTon, random:i(.2, .8)

 chnmix aL, "collect_text_C_L"
 chnmix aR, "collect_text_C_R"

endin



/*****************************************************************/
/********************** KLANGPRODUKTION **************************/
/*****************************************************************/

instr GrainRasch

 ;gewünschte graindauer
 iGrainDur = p3

 ;länger machen um filter ausklang zu ermöglichen
 p3 *= 2
```

```
 ;lautstärke mit zufälliger abweichung +- 3db
 iAmp = p4
 iMaxDbDev = 3
 iAmp *= ampdb(random:i(-iMaxDbDev,iMaxDbDev))

 ;grain hüllkurve
 aEnvPointer linseg 0, iGrainDur, 1
 aGrainEnv tablei aEnvPointer, giHalfSine, 1

 ;rauschen (kann vielleicht global sein)
 aNoise rand aGrainEnv*iAmp, 2, 1

 ;filtern
 iCf = p5
 iQ = p6
 iCf limit iCf, 20, 14000
 aFilt mode aNoise/10, iCf, iQ

 ;panning
 iPan = p7
 aL, aR pan2 aFilt, iPan

 ;output
 chnmix aL, "collect_rasch_L"
 chnmix aR, "collect_rasch_R"

endin

instr GrainArpegg

 ;gewünschte graindauer
 iGrainDur = p3

 ;länger machen um filter ausklang zu ermöglichen
 p3 *= 2

 ;lautstärke mit zufälliger abweichung +- 3db
 iAmp = p4
 iMaxDbDev = 3
 iAmp *= ampdb(random:i(-iMaxDbDev,iMaxDbDev))

 ;grain hüllkurve
 aEnvPointer linseg 0, iGrainDur, 1
 aGrainEnv tablei aEnvPointer, giHalfSine, 1

 ;rauschen (kann vielleicht global sein)
 aNoise rand aGrainEnv*iAmp, 2, 1

 ;filtern
 iCf = p5
 iQ = p6
 iCf limit iCf, 20, 14000
 aFilt mode aNoise/10, iCf, iQ

 ;panning
 iPan = p7
 aL, aR pan2 aFilt, iPan

 ;output
 chnmix aL, "collect_text_A_L"
 chnmix aR, "collect_text_A_R"

endin


instr EinfachArpegg

 ;laustärke
 idB = p4

 ;rauschdauer (kommt an in ms)
 iNoiseDur = p5/1000

 ;pitch und daraus zentralfrequenz
 iPch = p6
 iCf = cpsmidinn(iPch)
 iCf limit iCf, 20, 14000

 ;qualität
 iQ = p7

 ;panning
 iPan = p8

 ;hüllkurve
 aEnv transeg ampdb(idB), iNoiseDur, -6, 0

 ;rauschen
 aNoise rand aEnv, 2, 1

 ;bandpass
 aFilt mode aNoise, iCf, iQ

 ;zusätzliche ausblende
 aEnv linen aFilt, 0, p3, p3/4

 ;zufälliges panning
 aL, aR pan2 aEnv, iPan

 ;output
 chnmix aL, "collect_text_A_L"
 chnmix aR, "collect_text_A_R"

endin
```

```
instr EinfachArpegg_B
//output schicht B statt A

 ;laustärke
 idB = p4

 ;rauschdauer (kommt an in ms)
 iNoiseDur = p5/1000

 ;pitch und daraus zentralfrequenz
 iPch = p6
 iCf = cpsmidinn(iPch)
 iCf limit iCf, 20, 14000

 ;qualität
 iQ = p7

 ;panning
 iPan = p8

 ;hüllkurve
 aEnv transeg ampdb(idB), iNoiseDur, -6, 0

 ;rauschen
 aNoise rand aEnv, 2, 1

 ;bandpass
 aFilt mode aNoise, iCf, iQ

 ;zusätzliche ausblende
 aEnv linen aFilt, 0, p3, p3/4

 ;zufälliges panning
 aL, aR pan2 aEnv, iPan

 ;output
 chnmix aL, "collect_text_B_L"
 chnmix aR, "collect_text_B_R"

endin


instr EinfachArpeggVibr

 ;laustärke
 idB = p4

 ;rauschdauer (kommt an in ms)
 iNoiseDur = p5/1000

 ;pitch
 iPch = p6

 ;qualität
 iQ = p7

 ;panning
 iPan = p8

 ;vibrato maximale abweichung (1=halbton)
 iVibMaxDev = p9

 ;frequenz für randomi
 iVibFreq = p10

 ;fade in
 iFadIn = p11

 ;freq mit bewegung
 kPch randomi iPch-iVibMaxDev, iPch+iVibMaxDev, iVibFreq, 3
 kFreq limit cpsmidinn(kPch), 20, 14000

 ;hüllkurve
 aEnv transeg ampdb(idB), iNoiseDur, -6, 0

 ;rauschen
 aNoise rand aEnv, 2, 1

 ;bandpass
 aFilt mode aNoise, kFreq, iQ

 ;zusätzliche ausblende
 aEnv linen aFilt, iFadIn, p3, p3/4

 ;panning
 aL, aR pan2 aEnv, iPan

 ;output
 chnmix aL, "collect_text_A_L"
 chnmix aR, "collect_text_A_R"

endin


/*****************************************************************/
/************************** OUTPUT ******************************/
/*****************************************************************/

instr Collect

 /* R */
 //empfangen
 aRaschDry_L chnget "collect_rasch_L"
 aRaschDry_R chnget "collect_rasch_R"
```

```
 //wd mix
 aRaschWet_L, aRaschWet_R reverbsc aRaschDry_L, aRaschDry_R, gkRaschHallRoomSiz, 20000
 aRasch_L ntrpol aRaschDry_L, aRaschWet_L, gkRaschHallWdMix
 aRasch_R ntrpol aRaschDry_R, aRaschWet_R, gkRaschHallWdMix

 //lautstärke und zeigen
 aRasch_L *= ampdb(gk_R_VolDbLive) * ampdb(gkGlobVolDb)
 aRasch_R *= ampdb(gk_R_VolDbLive) * ampdb(gkGlobVolDb)
 Meter "R_outL", "R_outL_over", aRasch_L, gkTrigDisp
 Meter "R_outR", "R_outR_over", aRasch_R, gkTrigDisp


 /* A */
 //empfangen
 a_A_dry_L chnget "collect_text_A_L"
 a_A_dry_R chnget "collect_text_A_R"

 //wd mix
 a_A_wet_L, a_A_wet_R reverbsc a_A_dry_L, a_A_dry_R, gkArpeggHallRoomSiz, 20000
 a_A_L ntrpol a_A_dry_L, a_A_wet_L, gkArpeggHallWdMix
 a_A_R ntrpol a_A_dry_R, a_A_wet_R, gkArpeggHallWdMix

 //lautstärke und zeigen
 a_A_L *= ampdb(gk_A_VolDbLive) * ampdb(gkGlobVolDb)
 a_A_R *= ampdb(gk_A_VolDbLive) * ampdb(gkGlobVolDb)
 Meter "A_outL", "A_outL_over", a_A_L, gkTrigDisp
 Meter "A_outR", "A_outR_over", a_A_R, gkTrigDisp


 /* B */
 //empfangen
 a_B_L chnget "collect_text_B_L"
 a_B_R chnget "collect_text_B_R"

 //lautstärke und zeigen
 a_B_L *= ampdb(gk_B_VolDbLive) * ampdb(gkGlobVolDb)
 a_B_R *= ampdb(gk_B_VolDbLive) * ampdb(gkGlobVolDb)
 Meter "B_outL", "B_outL_over", a_B_L, gkTrigDisp
 Meter "B_outR", "B_outR_over", a_B_R, gkTrigDisp


 /* C */
 //empfangen
 a_C_L chnget "collect_text_C_L"
 a_C_R chnget "collect_text_C_R"

 //lautstärke und zeigen
 a_C_L *= ampdb(gk_C_VolDbLive) * ampdb(gkGlobVolDb)
 a_C_R *= ampdb(gk_C_VolDbLive) * ampdb(gkGlobVolDb)
 Meter "C_outL", "C_outL_over", a_C_L, gkTrigDisp
 Meter "C_outR", "C_outR_over", a_C_R, gkTrigDisp


 /* OUTPUT UND ANZEIGE*/
 aOutL = aRasch_L + a_A_L + a_B_L + a_C_L
 aOutR = aRasch_R + a_A_R + a_B_R + a_C_R

 outch giOutChnL, aOutL, giOutChnR, aOutR

 Meter "outL", "outL_over", aOutL, gkTrigDisp
 Meter "outR", "outR_over", aOutR, gkTrigDisp


 //KANÄLE NULLEN
 chnclear "collect_rasch_L"
 chnclear "collect_rasch_R"
 chnclear "collect_text_A_L"
 chnclear "collect_text_A_R"
 chnclear "collect_text_B_L"
 chnclear "collect_text_B_R"
 chnclear "collect_text_C_L"
 chnclear "collect_text_C_R"


  // record
 itim     date
 Stim     dates     itim
 Syear    strsub    Stim, 20, 24
 Smonth   strsub    Stim, 4, 7
 Sday     strsub    Stim, 8, 10
 iday     strtod    Sday
 Shor     strsub    Stim, 11, 13
 Smin     strsub    Stim, 14, 16
 Ssec     strsub    Stim, 17, 19
 S_outfile  sprintf  "%s%s%02d__%s_%s_%s.wav", Syear, Smonth, iday, Shor,Smin, Ssec
 fout S_outfile, 18, aRasch_L, aRasch_R, a_A_L, a_A_R, a_B_L, a_B_R, a_C_L, a_C_R, gaLiveIn

endin
schedule "Collect", 0.1, 99999



/*****************************************************************/
/************************ TOOLS ******************************/
/*****************************************************************/

instr ChangeRaschVolDbIntern
//ändert die (interne) lautstärke des raschelns über eine zeit (p3) um eine db differenz

 //neues db ziel
 iDbTarget = p4

 //db differenz positiv (lauter) oder negativ (leiser)
 iCurrentDb = i(gkRaschVolDbIntern)
```

```
 iDbDiff = iDbTarget - iCurrentDb

 //änderungen der db zahl pro k-zyklus
 iDbDiff_k = iDbDiff / (p3*kr)

 //anwenden
 kRaschVolDb init iCurrentDb
 gkRaschVolDbIntern += iDbDiff_k

endin

instr ChangeRaschRtmAuslassungen

 //neuer wert für die auslassungen
 iAuslassTarget = p4

 //differenz
 iCurrentAuslass = i(gkRaschRtmAuslassungen)
 iAuslassDiff = iAuslassTarget - iCurrentAuslass

 //änderungen pro k-zyklus über p3
 iAuslassDiff_k = iAuslassDiff / (p3*kr)

 //ausführen
 gkRaschRtmAuslassungen += iAuslassDiff_k

endin

instr ChangeRaschRtmTempo

 //neuer wert für das tempo
 iTarget = p4

 //differenz
 iCurrent = i(gkRaschRtmTempo)
 iDiff = iTarget - iCurrent

 //änderungen pro k-zyklus über p3
 iDiff_k = iDiff / (p3*kr)

 //ausführen
 gkRaschRtmTempo += iDiff_k

endin

instr KnuthShowTrigImp

 chnset k(1), "knuth_trig_imp"
 if release:k() == 1 then
  chnset k(0), "knuth_trig_imp"
 endif

endin

instr ShowActive

 gS_welche[] fillarray  "Z1", "Z2", "Z3", "Z4", "Z5", "Z6", "Z7", "Z8", "T1", "T2", "T3",
                                                "T4", "T5", "T6", "T6_Cues", "T7", "T7_A", "T7_B", "T8_1", "T8_2", "T8_3"

 gkActiveStatus[] init lenarray(gS_welche)

 //einmal so viele instanzen erzeugen wie der array will
 iIndx = 0
 while iIndx < lenarray(gS_welche) do
  schedule "WatchActive", 0, 99999, iIndx
  iIndx += 1
 od

 //immer dann rausschicken wenn sich was geändert hat, und am anfang
 if timeinstk() == 2 || changed2(gkActiveStatus) == 1 then

  S_print strcpyk ""
  kIndx = 0
  while kIndx < lenarray(gS_welche) do

   if gkActiveStatus[kIndx] == 1 then
    S_neu sprintfk "%s\n\n", gS_welche[kIndx]
   else
    S_neu strcpyk ""
   endif

   S_print strcatk S_print, S_neu

   kIndx += 1
  od

  chnset S_print, "aktive_instrumente"

 endif

endin
maxalloc "ShowActive", 1

instr ShowActive_2

 gS_welchenoch[] fillarray      "Knuth", "Knuth_key", "T1_dummy", "T2_dummy", "T3_dummy", "T4_dummy", "T5_dummy",
"T6_dummy",
                                                "T7_dummy", "T8_dummy", "Z6_seq", "Z7_seq", "Z7_tief", "RaschSeq",
"RaschSeq_Z2", "HakSeq",
                                                "EinRaschler", "EinRaschlerZ5", "T1_Ton", "T2_tief", "T3_Welle", "T4_Ton",
"T5_Ton",
                                                "T6_Ton", "T6_Ton_CD", "T7_A_Ton", "T7_B_Ton", "T8_1_Ton", "T8_3_Ton",
"ChangeRaschVolDbIntern",
                                                "ChangeRaschRtmAuslassungen", "ChangeRaschRtmTempo"
```

```
 gkActiveStatus_2[] init lenarray(gS_welchenoch)

 //einmal so viele instanzen erzeugen wie der array will
 iIndx = 0
 while iIndx < lenarray(gS_welchenoch) do
  schedule "WatchActive_2", 0, 99999, iIndx
  iIndx += 1
 od

 //rausgeben bei änderung oder am anfang
 if timeinstk() == 2 || changed2(gkActiveStatus_2) == 1 then

  S_print strcpyk ""
  kIndx = 0
  while kIndx < lenarray(gS_welchenoch) do

   if gkActiveStatus_2[kIndx] == 1 then
    S_neu sprintfk "%s\n\n", gS_welchenoch[kIndx]
   elseif gkActiveStatus_2[kIndx] > 1 then
    S_neu sprintfk "%s (%d)\n\n", gS_welchenoch[kIndx], gkActiveStatus_2[kIndx]
   else
    S_neu strcpyk ""
   endif

   S_print strcatk S_print, S_neu

   kIndx += 1
  od

  chnset S_print, "aktive_instrumente_2"

 endif

endin
maxalloc "ShowActive_2", 1


instr WatchActive

 iInstrIndx = p4
 S_instr = gS_welche[iInstrIndx]
 kActive active S_instr
 if changed(kActive) == 1 then
  gkActiveStatus[iInstrIndx] = kActive
 endif

endin


instr WatchActive_2

 iInstrIndx = p4
 S_instr = gS_welchenoch[iInstrIndx]
 kActive active S_instr
 if changed(kActive) == 1 then
  gkActiveStatus_2[iInstrIndx] = kActive
 endif

endin


</CsInstruments>
<CsScore>

</CsScore>
</CsoundSynthesizer>
```