Chapter 5. Meeting 5, History: Serialism, Loops, Tiling, and Phasing

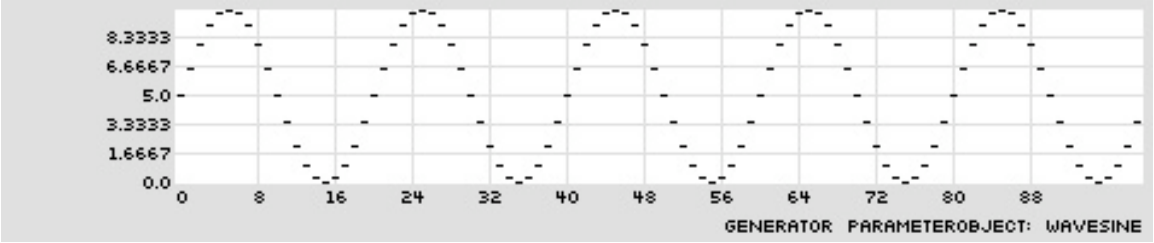
5.1. Announcements

- Musical Design Report 1 due Tuesday, 23 February
- Review readings from last class

5.2. Trigonometric Functions and Break-Point Graphs as ParameterObjects

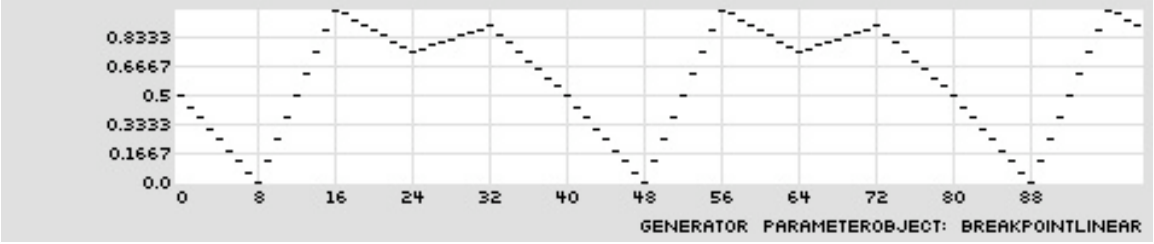
- WaveSine: A scalable sine oscillator controlled by seconds or events per cycle

```
pi{}ti{} :: tpmap 100 ws,e,20,0,0,10
waveSine, event, (constant, 20), 0, (constant, 0), (constant, 10)
TPmap display complete.
```



- BreakPointLinear: Break point segments defined by seconds or events

```
pi{}ti{} :: tpmap 100 bpl,e,l,((0,.5),(8,0),(16,1),(24,.75),(32,.9),(40,.5))
breakPointLinear, event, loop, ((0,0.5),(8,0),(16,1),(24,0.75),(32,0.9),(40,0.5))
TPmap display complete.
```



- Numerous alternative trigonometric function generators exist as ParameterObjects: WaveCosine, WavePulse, WaveSawDown, WaveSine, WaveTriangle
- Numerous alternative break-point function generators exist as ParameterObjects: BreakPointFlat, BreakPointHalfCosine, BreakPointLinear, BreakPointPower

5.3. Configuring Tempo

- The TIe command can be use to edit tempo by specifying "b" for BPM
- Tempo can be controlled by any ParameterObject

5.4. Approaches to Composing Time

- Creating overlapping repeats of the same material
- Creating overlapping repeats of transformed material
- Creating ordered material that is then transformed in ways that retain order

5.5. Canons and Tiling

- Create an initial line and repeat it with staggered entrances
- An approach to polyphony
- The initial line can be temporally shifted and temporally transformed
- Can be seen as an approach to musical tiling

5.6. Listening: Andriessen

- Louis Andriessen (1939-)
- Dutch composer notable for combining American Minimalism with (at times) more diverse harmonic language
- Andriessen: "Hout" (1991)

5.7. Building a Basic Beat

- Kick, snare, and hats
- Command sequence:
  - emo mp
  - tin a 36
  - tie r pt,(c,2),(bg,oc,(7,5,2,1,1)),(c,1)
  - tin b 37
  - tie r pt,(c,2),(bg,oc,(3,5)),(bg,oc,(0,1))
  - tin c 42
  - tie r pt,(c,2),(c,1),(bg,oc,(0,1))
  - eln; elh

5.8. A Basic Beat with More Complex Snare Part

- Continued command sequence:
  - tio b
  - tie r pt,(c,4),(bg,rp,(3,3,5,4,1)),(bg,oc,(0,1,1))
  - eln; elh

5.9. Adding Canonic Snare Imitation: Texture Copying

- Copying a texture creates a new, independent, and dynamic part
- While having identically configured ParameterObjects, if randomness is employed, unique structures will be created
- Continued command sequence:
  - tio b
  - ticp b b1
  - tie t .25, 20.25

- tie i 76
- ticp b b2
- tie t .5, 20.5
- tie i 77
- eln; elh

5.10. Saving and Loading the AthenaObject

- An athenaCL XML file can be loaded in to athenaCL to restore Textures
- These XML files can be automatically created whenever an event list is created
- Continued command sequence:
  - eoo xao
  - eln

5.11. Building an Extended Rhythmic Line with Canonic Imitation

- Using different length ordered cyclic generators will create complex but non-random sequences
- Command sequence:
  - aorm confirm
  - emo mp
  - tin a 77
  - tie r pt,(c,1),(c,1),(c,1)
  - tin b 67
  - tie r pt,(bg,oc,(2,4,1)),(bg,oc,(3,5,1,7,1,3)),(c,1)
  - ticp b b1
  - tie t 0.125,20.125
  - tie i 60
  - ticp b b2

- tie t 0.25,20.25
- tie i 68
- eln; elh

5.12. Creating Mensural Canons

- Mensural canons use ratio-base time signatures for each part
- Continued command sequence:
  - tio b1
  - tie b c,90
  - tio b2
  - tie b c,180
  - eln; elh

5.13. Extensions

- We can generate complex, deterministic patterns by combining cycles at high ratios
- The same musical rhythm at different (low ratio related) rates produces interesting musical results

5.14. Tonal, Atonal, and Post-Tonal

- Tonal music employs functional harmony
  - Harmonies (chords) have a trajectory, expectation, and a resolution
  - One (or two) chords are more than others
- Atonal music does not employ functional harmony
  - The expectations and priorities of chords are removed
  - Ideally, no pitch is more important than any other
- Post-tonal refers approaches to harmony other than tonal
  - May be atonal, or may employ other approaches to pitch
  - Pitch centers may be developed and exploited

5.15. Serialism

- An approach to atonality that serialized (ordered) elements of musical parameters, developed by Arnold Schoenberg
- An alternative approach to atonality employed chords that completed the aggregate (all 12 pitches), developed by Josef Matthias Haur
- By serializing the order of all 12-tone pitches, all get equal usage
- Pitch groups smaller than 12 can be used
- A series of all 12 tones is used as a motivic origin
  - The series can be transposed to any of 12 pitch levels: prime
  - The series can be reversed: retrograde
  - The series can be inverted ((12-*n*) % 12): inversion
  - The inverted series can be reversed: retrograde inversion
  - The 12 x 4 possible rows can be presented in a matrix

    Generated with Python tools in music21: http://code.google.com/p/music21/

```
from music21 import serial

p = [8,1,7,9,0,2,3,5,4,11,6,10]
print serial.rowToMatrix(p)
```

```
  0  5 11  1  4  6  7  9  8  3 10  2
  7  0  6  8 11  1  2  4  3 10  5  9
  1  6  0  2  5  7  8 10  9  4 11  3
 11  4 10  0  3  5  6  8  7  2  9  1
  8  1  7  9  0  2  3  5  4 11  6 10
  6 11  5  7 10  0  1  3  2  9  4  8
  5 10  4  6  9 11  0  2  1  8  3  7
  3  8  2  4  7  9 10  0 11  6  1  5
  4  9  3  5  8 10 11  1  0  7  2  6
  9  2  8 10  1  3  4  6  5  0  7 11
  2  7  1  3  6  8  9 11 10  5  0  4
 10  3  9 11  2  4  5  7  6  1  8  0
```

- Milton Babbitt and Pierre Boulez extended serial techniques to new parameters and alternative organizations
- Karlheinz Stockhausen and others attempted to employ serial techniques to organize parameters in the early Electronic Music studio
- Total serialism orders amplitudes, rhythms, and other musical parameters

5.16. Listening: Boulez

- Pierre Boulez (1925-)
- Post WWII and total serialism
- Boulez: "Structures, Book I" (1952)

5.17. Extensions

- The algorithmic opportunities of serialism led many composers to generalize such techniques with the computer
- athenaCL features Paths as a way for Textures to share source Pitch data
- One Path might be shared by multiple Textures, each transposing, reversing, and inverting this Path to create serial arrangements
- While some have tried (Babbitt 1958), serial rhythm techniques have not been widely embraced

5.18. Phasing

- Musical material shifting in and out of time, or moving at different rates
- Developed out of manipulations to recording reels: flanging and phasing

- Can be used as a canon-like technique

5.19. Listening: Reich

- Steve Reich (1936-)
- Influenced by techniques of minimalism based in part on music of Terry Riley, La Monte Young, and others
- Reich: "It's gonna rain" (1965)
- "Scorification" of a technological process for acoustic instruments
- Reich: "Piano Phase" (1967)

5.20. Phasing with athenaCL Python Libraries

- pianoPhase.py

```
import os
from athenaCL.libATH import midiTools
from athenaCL.libATH import osTools
from athenaCL.libATH import pitchTools
from athenaCL.libATH import rhythm
from athenaCL.libATH.libOrc import generalMidi
from athenaCL.libATH.libPmtr import parameter

OUTDIR = '/Volumes/xdisc/_scratch'
BEATDUR = rhythm.bpmToBeatTime(225) # provide bpm value

def getInstName(nameMatch):
   for name, pgm in generalMidi.gmProgramNames.items():
      if name.lower().startswith(nameMatch.lower()):
         return pgm # an integer
   return None

def getSource(repeat):
   """get source melody and rhythm"""

   pitchSequence = ['E4','F#4','B4','C#5','D5','F#4',
                    'E4','C#5','B4','F#4','D5','C#5']
   rhythmSequence = [.5, .5, .5, .5, .5]
   ampGen = parameter.factory(['ws','e',14,0,90,120]) # sine osc b/n 90 and 120
```

```
    score = []
    tStart = 0.0
    for i in range(len(pitchSequence) * repeat):
       ps = pitchTools.psNameToPs(pitchSequence[i%len(pitchSequence)])
       pitch = pitchTools.psToMidi(ps)
       dur = BEATDUR * rhythmSequence[i%len(rhythmSequence)]
       amp = int(round(ampGen(0)))
       pan = 30
       event = [tStart, dur, amp, pitch, pan]
       score.append(event)
       tStart = tStart + dur

    return score, len(pitchSequence)

 def transformSource(score, srcLength):
    """transform source, srcLength is size of each melodic unit
    """
    post = []
    octaveShift = -1
    panShift = 60
    shiftUnit = BEATDUR / 16.

    eCount = 0
    repCount = 0 # starting at zero means first cycle will be in phase
    for event in score:
       if eCount % srcLength == 0:
          shift = shiftUnit * repCount
          repCount = repCount + 1 # increment after using

       newEvent = [event[0]+shift, event[1], event[2],
                   event[3]+(octaveShift*12), (event[4]+panShift)%128]
       post.append(newEvent)
       eCount = eCount + 1 # increment for each event
    return post

 def main():
    repeat = 33
    partA, seqLen = getSource(repeat)
    partB = transformSource(partA, seqLen)

    trackList = [('part-a', getInstName('piano'), None, partA),
                 ('part-b', getInstName('piano'), None, partB),]
    path = os.path.join(OUTDIR, 'test.midi')
    mObj = midiTools.MidiScore(trackList)
    mObj.write(path)
    osTools.openMedia(path)

 if __name__ == '__main__':
    main()
```

5.21. Beats with athenaCL Python Libraries

- basicBeat.py

```
import os, random
from athenaCL.libATH import midiTools
from athenaCL.libATH import osTools
from athenaCL.libATH import pitchTools
from athenaCL.libATH import rhythm
from athenaCL.libATH.libOrc import generalMidi
from athenaCL.libATH.libPmtr import parameter
```

```
OUTDIR = '/Volumes/xdisc/_scratch' # provide output directory
BEATDUR = rhythm.bpmToBeatTime(160) # provide bpm value

def getInstPitch(nameMatch):
   for name, pgm in generalMidi.gmPercussionNames.items():
      if name.lower().startswith(nameMatch.lower()):
         return pgm # an integer
   raise NameError('bad pitch name')

def getKickSnare(repeat):
   rhythmA = [1, 1.5, .5, 1]
   rhythmB = [1.5, .5, 1.5, .5]
   rhythmC = [1.75, .25,  1.5, .125, .125, .125, .125]
   instA   = ['acousticBassDrum','sideStick']
   instB   = ['sideStick']

   ampGen = parameter.factory(['rb',.2,.2,110,127])

   score = []
   tStart = 0.0
   for q in range(repeat):
      if q % 3 == 0:
         rhythmSequence = rhythmB
         instSequence = instA
      elif q % 11 == 10:
         rhythmSequence = rhythmC
         instSequence = instB
         random.shuffle(rhythmSequence)
      else:
         rhythmSequence = rhythmA
         instSequence = instA

      for i in range(len(rhythmSequence)):
         inst = instSequence[i % len(instSequence)]
         pitch = getInstPitch(inst)
         dur = BEATDUR * rhythmSequence[i % len(rhythmSequence)]
         amp = int(round(ampGen(0)))
         pan = 63
         event = [tStart, dur, amp, pitch, pan]
         score.append(event)
         tStart = tStart + dur
   return score, len(rhythmSequence)

def getHats(repeat):
   rhythmSequence = [.5, .5, .25, .25, .5, .5, .5, .5]
   instSequence   = ['closedHiHat','closedHiHat',
                     'closedHiHat','closedHiHat',
                     'closedHiHat','openHiHat']
   ampGen = parameter.factory(['rb',.2,.2,50,80])

   score = []
   tStart = 0.0
   for q in range(repeat):
      for i in range(len(rhythmSequence)):
         inst = instSequence[i % len(instSequence)]
         pitch = getInstPitch(inst)
         dur = BEATDUR * rhythmSequence[i % len(rhythmSequence)]
         amp = int(round(ampGen(0)))
         pan = 63
         event = [tStart, dur, amp, pitch, pan]
         score.append(event)
         tStart = tStart + dur

   return score, len(rhythmSequence)
```

```
def main():
   repeat = 33
   partA, seqLen = getKickSnare(repeat)
   partB, seqLen = getHats(repeat)

   trackList = [('part-a', 0, 10, partA),
                ('part-b', 0, 10, partB),]

   path = os.path.join(OUTDIR, 'test.midi')
   mObj = midiTools.MidiScore(trackList)
   mObj.write(path) # writes in cwd
   osTools.openMedia(path)

if __name__ == '__main__':
   main()
```

5.22. Building an Extended Rhythmic Line with Fixed Tempo Phasing

- Using different tempi will create shifting rhythmic patterns
- Command sequence:
  - aorm confirm
  - emo mp
  - tin a 70
  - tie r pt,(bg,oc,(2,4,4)),(bg,oc,(4,1,1,2,1)),(c,1)
  - tie t 0,60
  - ticp a a1
  - tie b c,124
  - ticp a a2
  - tie b c,128
  - eln; elh

5.23. Building an Extended Rhythmic Line with Dynamic Tempo Phasing

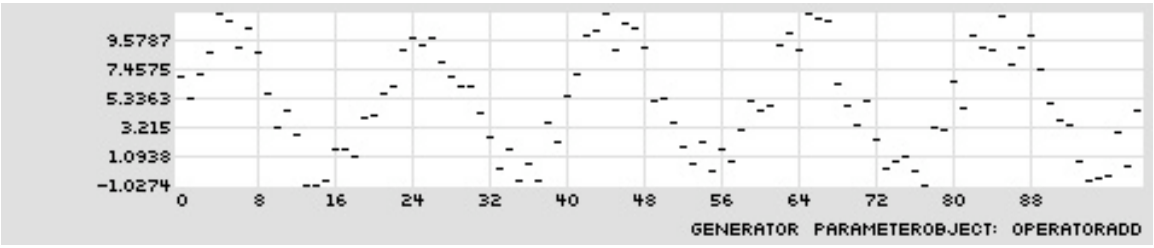
- Oscillating the tempo at different rates will create dynamic changes
- Command sequence:
  - aorm confirm

- emo mp
- tin a 64
- tie r pt,(bg,oc,(2,4,4)),(bg,oc,(4,1,1,2,1)),(c,1)
- tie t 0,60
- ticp a a1
- tie i 60
- tie b ws,t,20,0,115,125
- ticp a a2
- tie i 69
- tie b ws,t,30,0,100,140
- eln; elh

5.24. Extensions

- Many works have been built with slow and gradual tempo changes
- Tempos might slowly deviate with a BreakPointLinear or similar generator
- Tempos might be randomly perturbed by adding in randomness: PO OperatorAdd can sum two ParameterObjects

```
pi{}ti{} :: tpmap 100 oa,(ws,e,20,0,0,10),(ru,-2,2)
operatorAdd, (waveSine, event, (constant, 20), 0, (constant, 0), (constant, 10)),
(randomUniform,
(constant, -2), (constant, 2))
TPmap display complete.
```



21M.380 Music and Technology: Algorithmic and Generative Music
Spring 2010