DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Anexo código mini-collider

Teoría de lenguajes

Grupo: 11

Integrante	LU	Correo electrónico
Calderini, Nicolás	820/10	calderini.nicolas@gmail.com
Hernández, Santiago	48/11	santi-hernandez@hotmail.com
Marasca, Dardo	227/07	dmarasca@yahoo.com.ar
Saravia, Nicolás	905/04	nicolasaravia@yahoo.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

El código entregado se encuentra dividido en módulos con la siguiente estructura:

```
+--- mini_collider.py
+--- minicollider
|    \--- lexer.py
|    \--- mixer.py
|    \--- parser.py
|    \--- test
|         \--- test_mixer.py
|         \--- test_parser.py
```

1. mini_collider.py

```
import minicollider.parser
try:
    import argparse
except ImportError:
    from minicollider.external import argparse


def parsear_argumentos():
    argparser = argparse.ArgumentParser(formatter_class=
                                        argparse.ArgumentDefaultsHelpFormatter)
    argparser.add_argument('-s', '--samplerate',
                           help="The desired sample rate.",
                           default=8000,
                           type=int)
    argparser.add_argument('-b', '--beat',
                           help="The desired beat.",
                           default=8000 / 12,
                           type=int)
    argparser.add_argument('-f', '--file',
                           help="A file with a buffer to parse (optional).")
    return argparser.parse_args()


def parsear_archivo(file):
    try:
        archivo = open(args.file, 'r')
        entrada = archivo.read()
        archivo.close()
    except IOError:
        print 'Error opening the file.'
        exit(1)
    try:
        minicollider.parser.parse(entrada)
    except Exception, e:
        if e: msg = str(e)
        else: msg = 'Syntax Error'
        print "Error: %s" % msg

def prompt():
    while 1:
        try:
            entrada = raw_input('buffer > ')
        except EOFError:
            print
            break
        if entrada != '':
            try:
                minicollider.parser.parse(entrada)
            except Exception, e:
                print "Error: %s" % e



if __name__ == '__main__':
    args = parsear_argumentos()
    minicollider.parser.init(args.samplerate, args.beat)
```

```
57      if args.file is not None:
58          parsear_archivo(args.file)
59      else:
60          prompt()
```

2. lexer.py

```
# ------------------------------------------------------------------------------
# lexer.py
#
# Lexer para el mini-collider
# ------------------------------------------------------------------------------
try:
    import ply.lex as lex
except ImportError:
    import external.lex as lex
import re

tokens = (
    'NUM', 'SIN', 'LIN', 'SIL', 'NOI', 'PLAY', 'POST', 'LOOP',
    'TUNE', 'FILL', 'REDU', 'EXPA', 'CON', 'MIX', 'ADD', 'SUB', 'MUL',
    'DIV', 'LPAREN', 'RPAREN', 'LLLAVE', 'RLLAVE','PLOT', 'COMA',
)

# Tokens

t_SIN         = r'sin'
t_LIN         = r'linear|lin'
t_SIL         = r'silence|sil'
t_NOI         = r'noise|noi'
t_PLAY        = r'.play'
t_POST        = r'.post'
t_LOOP        = r'.loop'
t_TUNE        = r'.tune'
t_FILL        = r'.fill'
t_REDU        = r'.reduce'
t_EXPA        = r'.expand'
t_PLOT        = r'.plot'
t_CON         = r'con|;'
t_MIX         = r'mix|&'
t_ADD         = r'add|\+'
t_SUB         = r'sub|-'
t_MUL         = r'mul|\*'
t_DIV         = r'div|/'
t_LPAREN      = r'\('
t_RPAREN      = r'\)'
t_LLLAVE      = r'\{'
t_RLLAVE      = r'\}'
t_COMA        = r','
t_ignore_COMM = r'//.*\n'
t_ignore_WS   = r'\s|\t|\n'

def t_NUM(t):
    r'\d+(\.\d+)?'
    if t.value.find('.') == -1:
        t.value = int(t.value)
    else:
        t.value = float(t.value)

    return t

def t_error(t):
    raise SyntaxError("Caracter ilegal: '%s'" % t.value[0])
```

```
57
58 # Build the lexer
59 lexer = lex.lex()
```

3. mixer.py

```
import math
import numpy
import pylab
import pygame


NUMPY_ENCODING = numpy.int16
MIXER_ENCODING = -16

MAX_AMPLITUDE = 32000
SAMPLE_RATE = 8800
BEAT = SAMPLE_RATE / 12


def init(sample_rate=8800, beat=8800/12, init_pygame=1):
    global SAMPLE_RATE, BEAT

    SAMPLE_RATE = sample_rate
    BEAT = beat

    if init_pygame:
        pygame.mixer.pre_init(SAMPLE_RATE, MIXER_ENCODING, 1)
        pygame.init()


class Sound():
    def __init__(self, samples):
        if (len(samples)) == 0:
            raise Exception('No se puede crear un buffer vacio')

        self.samples = numpy.array(samples, numpy.float)

    def __eq__(self, other):
        return numpy.array_equal(self.samples, other.samples)

    def __iter__(self):
        return self.samples.__iter__()

    def __len__(self):
        return len(self.samples)

    def __add__(self, other):
        return self._oper(other, (lambda x, y: x + y))

    def __mul__(self, other):
        return self._oper(other, (lambda x, y: x * y))

    def __sub__(self, other):
        return self._oper(other, (lambda x, y: x - y))

    def __div__(self, other):
        return self._oper(other, (lambda x, y: x / y))

    def __floordiv__(self, other):
        return self.concat(other)

```

```
57      def __and__(self, other):
58          return self._oper(other, (lambda x, y: (x + y) / 2))
59
60      def get_samples(self):
61          return self.samples
62
63      def set_samples(self, samples):
64          self.samples = samples
65          return self
66
67      def play(self, speed=1):
68          if not(0 < speed):
69              raise Exception("[PLAY] Se esperaba un numero positivo: %s" % speed)
70          if speed != 1:
71              target = self.resample(int((1.0 / speed) * len(self)))
72          else:
73              target = self
74
75          amp_mul = MAX_AMPLITUDE / numpy.amax(target.samples)
76          samples = numpy.array(target.get_samples() * amp_mul, NUMPY_ENCODING)
77          channel = pygame.sndarray.make_sound(samples).play()
78          while channel.get_busy(): pass
79          return self
80
81      def plot(self):
82          pylab.plot(numpy.arange(int(len(self.samples))), self.samples)
83          pylab.show()
84          return self
85
86      def post(self):
87          print self
88          return self
89
90      def loop(self, count):
91          if not(0 < count):
92              raise Exception("[LOOP] Se esperaba un numero positivo: %s" % count)
93          return self.resize(int(count * len(self.samples)))
94
95      def resize(self, new_len):
96          if not(isinstance(new_len, int) and 0 < new_len):
97              raise Exception("[RESIZE] Se esperaba un entero positivo: %s" % new_len)
98          new_samples = numpy.zeros(new_len)
99          for i in xrange(new_len):
100             new_samples[i] = self.samples[i % len(self.samples)]
101         return Sound(new_samples)
102
103     def resample(self, new_len):
104         if not(isinstance(new_len, int) and 0 < new_len):
105             raise Exception("[RESAMPLE] Se esperaba un entero positivo: %s" % new_len)
106         new_samples = numpy.zeros(new_len)
107         for i in xrange(new_len):
108             new_samples[i] = self.samples[int(i * len(self) / new_len)]
109         return Sound(new_samples)
110
111     def copy(self):
112         return Sound(self.samples)
113
114     def concat(self, other):
```

```
115         new_samples = numpy.concatenate((self.samples, other.samples))
116         return Sound(new_samples)
117 
118     def tune(self, pitch):
119         return self.resample(int(
120                 len(self)
121                 * ( (2**(1.0/12))**(-pitch) )
122         ))
123 
124     def fill(self, count):
125         if not(0 < count):
126             raise Exception("[FILL] Se esperaba un numero positivo: %s" % count)
127         new_len = int(BEAT * count)
128 
129         if (len(self) >= new_len): return self.copy()
130 
131         new_samples = numpy.zeros(new_len)
132         for i in xrange(len(self)):
133             new_samples[i] = self.samples[i]
134         return Sound(new_samples)
135 
136     def reduce(self, count=1):
137         if not(0 < count):
138             raise Exception("[REDUCE] Se esperaba un numero positivo: %s" % count)
139         new_len = int(count * BEAT)
140         if (len(self) > new_len):
141             return self.resample(new_len)
142         else:
143             return self.copy()
144 
145     def _oper(self, other, op):
146         if (len(self) < len(other)):
147             a = self.resize(len(other))
148             b = other
149         else:
150             a = self
151             b = other.resize(len(self))
152 
153         new_samples = numpy.zeros(len(a))
154         for i in xrange(len(a)):
155             new_samples[i] = op(a.samples[i], b.samples[i])
156         return Sound(new_samples)
157 
158     def expand(self, count=1):
159         if not(0 < count):
160             raise Exception("[EXPAND] Se esperaba un numero positivo: %s" % count)
161         new_len = int(count * BEAT)
162         if (len(self) < new_len):
163             return self.resample(new_len)
164         else:
165             return self.copy()
166 
167     def __str__(self):
168         return str(self.samples)
169 
170     def tolist(self):
171         return self.samples.tolist()
172 
```

```
173 class SoundGenerator():
174     def __init__(self):
175         pass
176 
177     def get_sample_rate(self):
178         return SAMPLE_RATE
179 
180     def get_beat(self):
181         return BEAT
182 
183     def get_beats_per_second(self):
184         return SAMPLE_RATE / BEAT
185 
186     def from_list(self, samples):
187         return Sound(numpy.array(samples))
188 
189     def sine(self, cicles, amp):
190         if not(0<= amp <=1):
191             raise Exception("[SINE] Amplitud incorrecta: %s" % amp)
192         if not(0 < cicles and isinstance(cicles, int)):
193             raise Exception("[SINE] Valor de ciclos incorrecto: %s" % cicles)
194         omega = (cicles * numpy.pi * 2) / BEAT
195         xvalues = numpy.arange(BEAT) * omega
196         return Sound(amp * numpy.sin(xvalues))
197 
198     def silence(self):
199         return Sound(numpy.zeros(BEAT))
200 
201     def linear(self, start, end):
202         if not(-1<= start <=1 and -1<= end <=1):
203             raise Exception("[LINEAR]  Rango incorrecto: %s, %s" % (start, end))
204         return Sound(numpy.linspace(start, end, BEAT))
205 
206     def noise(self, amp):
207         if not(0<= amp <=1):
208             raise Exception("[NOISE] Amplitud incorrecta: %s" % amp)
209         return Sound(numpy.random.random(BEAT) * amp)
210 
211     def note(self, note, amp, octave=1):
212         freqencies = {
213             'C' : 261.63,
214             'D' : 293.66,
215             'E' : 329.63,
216             'F' : 349.23,
217             'G' : 392,
218             'A' : 440,
219             'B' : 493.88
220         }
221         return self.sine(freqencies[note] * octave / self.get_beats_per_second(), amp)
```

4. parser.py

```
# ------------------------------------------------------------------------------
# parser.py
#
# Parser para el mini-collider
# ------------------------------------------------------------------------------
try:
    import ply.yacc as yacc
except ImportError:
    import external.yacc as yacc
from lexer import tokens
import mixer

generator = None


def init(sample_rate, beat, init_pygame=1):
    global generator
    mixer.init(sample_rate, beat, init_pygame)
    generator = mixer.SoundGenerator()


def parse(input):
    try:
        res = parser.parse(input)
    except Exception, e:
        raise SyntaxError("%s" % e)
    return res

precedence = (
    ('left', 'CON', 'MIX'),
    ('nonassoc', 'LOOP', 'POST', 'TUNE', 'EXPA', 'REDU','FILL', 'PLAY', 'PLOT'),
    ('left', 'ADD', 'SUB'),
    ('left', 'MUL', 'DIV'),
)


def p_statement_expr(t):
    'START : BUFFER'

    t[0] = t[1]


def p_BUFFER_binop(t):
    '''BUFFER :       BUFFER CON BUFFER
                |     BUFFER MIX BUFFER
                |     BUFFER ADD BUFFER
                |     BUFFER SUB BUFFER
                |     BUFFER MUL BUFFER
                |     BUFFER DIV BUFFER'''

    if   t[2] in ['con', ';']: t[0] = t[1] // t[3]
    elif t[2] in ['mix', '&']: t[0] = t[1] & t[3]
    elif t[2] in ['add', '+']: t[0] = t[1] + t[3]
    elif t[2] in ['sub', '-']: t[0] = t[1] - t[3]
    elif t[2] in ['mul', '*']: t[0] = t[1] * t[3]
    elif t[2] in ['div', '/']: t[0] = t[1] / t[3]
```

```
57
58
59  def p_BUFFER_metodo_0param(t):
60      '''BUFFER :       BUFFER PLAY ONEPARAM
61                  |     BUFFER POST ONEPARAM
62                  |     BUFFER LOOP ONEPARAM
63                  |     BUFFER TUNE ONEPARAM
64                  |     BUFFER FILL ONEPARAM
65                  |     BUFFER REDU ONEPARAM
66                  |     BUFFER PLOT ONEPARAM
67                  |     BUFFER EXPA ONEPARAM '''
68
69      try:
70          if   t[2] == '.play': t[0] = t[1].play(1)
71          elif t[2] == '.post': t[0] = t[1].post()
72          elif t[2] == '.loop': t[0] = t[1].loop(1)
73          elif t[2] == '.tune': t[0] = t[1].tune(1)
74          elif t[2] == '.fill': t[0] = t[1].fill(1)
75          elif t[2] == '.plot': t[0] = t[1].plot()
76          elif t[2] == '.reduce': t[0] = t[1].reduce(1)
77          elif t[2] == '.expand': t[0] = t[1].expand(1)
78      except Exception, e:
79          print "Syntax error: %s" % e
80          raise SyntaxError
81
82
83  def p_ONEPARAM(t):
84      '''ONEPARAM :       LPAREN RPAREN
85                  | '''
86
87
88  def p_BUFFER_metodo_1param(t):
89      '''BUFFER :       BUFFER PLAY LPAREN NUM RPAREN
90                  |     BUFFER LOOP LPAREN NUM RPAREN
91                  |     BUFFER FILL LPAREN NUM RPAREN
92                  |     BUFFER REDU LPAREN NUM RPAREN
93                  |     BUFFER EXPA LPAREN NUM RPAREN '''
94
95      if   t[2] == '.play': t[0] = t[1].play(t[4])
96      elif t[2] == '.loop': t[0] = t[1].loop(t[4])
97      elif t[2] == '.fill': t[0] = t[1].fill(t[4])
98      elif t[2] == '.reduce': t[0] = t[1].reduce(t[4])
99      elif t[2] == '.expand': t[0] = t[1].expand(t[4])
100
101
102 def p_BUFFER_metodo_1param_tune_pos(t):
103     '''BUFFER :    BUFFER TUNE LPAREN NUM RPAREN'''
104     t[0] = t[1].tune(t[4])
105
106
107 def p_BUFFER_metodo_1param_tune_neg(t):
108     '''BUFFER :    BUFFER TUNE LPAREN SUB NUM RPAREN'''
109     t[0] = t[1].tune(-t[5])
110
111
112 def p_BUFFER_generador_0param(t):
113     '''BUFFER :     SIL ONEPARAM
114                 |   NOI ONEPARAM'''
```

```
115
116     if   t[1] in ['silence','sil']: t[0] = generator.silence()
117     elif t[1] in ['noise','noi']: t[0]   = generator.noise(1)
118
119
120 def p_BUFFER_generator_1param(t):
121     '''BUFFER :       SIN LPAREN NUM RPAREN
122                 |     NOI LPAREN NUM RPAREN '''
123
124     if   t[1] == 'sin': t[0] = generator.sine(t[3], 1)
125     elif t[1] in ['noise','noi']: t[0] = generator.noise(t[3])
126
127
128 def p_BUFFER_generator_2param_sin(t):
129     'BUFFER : SIN LPAREN NUM COMA NUM RPAREN '
130
131     t[0] = generator.sine(t[3], t[5])
132
133
134 def p_BUFFER_generator_2param_lin_pos_pos(t):
135     'BUFFER : LIN LPAREN NUM COMA NUM RPAREN'
136
137     t[0] = generator.linear(t[3], t[5])
138
139
140 def p_BUFFER_generator_2param_lin_neg_pos(t):
141     'BUFFER : LIN LPAREN SUB NUM COMA NUM RPAREN'
142
143     t[0] = generator.linear(-t[4], t[6])
144
145
146 def p_BUFFER_generator_2param_lin_pos_neg(t):
147     'BUFFER : LIN LPAREN NUM COMA SUB NUM RPAREN'
148
149     t[0] = generator.linear(t[3], -t[6])
150
151
152 def p_BUFFER_generator_2param_lin_neg_neg(t):
153     'BUFFER : LIN LPAREN SUB NUM COMA SUB NUM RPAREN'
154
155     t[0] = generator.linear(-t[4], -t[7])
156
157
158 def p_BUFFER_llaves(t):
159     'BUFFER : LLLAVE BUFFER RLLAVE '
160
161     t[0] = t[2]
162
163
164 def p_minus_number(t):
165     'BUFFER : SUB NUM'
166
167     t[0] = generator.from_list([- t[2]])
168
169
170 def p_expression_number(t):
171     'BUFFER : NUM'
172
```

```
173     t[0] = generator.from_list([t[1]])
174
175
176 def p_error(t):
177     raise SyntaxError("Syntax error in input!")
178
179 parser = yacc.yacc()
```

5. test_mixer.py

```
import minicollider.mixer as mixer
import numpy
import unittest

class MixerTestCase(unittest.TestCase):


    def assertElementsInRange(self, list, min, max):
        for item in list:
            self.assertTrue(min <= item <= max)

    def setUp(self):
        self.sample_rate = 4800
        self.beat = self.sample_rate / 12

        mixer.init(self.sample_rate, self.beat, 0)
        self.generator = mixer.SoundGenerator()


class TestSoundGeneratorCases(MixerTestCase):


    def test_from_list(self):

        sound = self.generator.from_list([1])
        self.assertEqual([1], sound.tolist())

        sound = self.generator.from_list([0.5])
        self.assertEqual([0.5], sound.tolist())

        sound = self.generator.from_list([0, 0.1, -1])
        self.assertEqual([0, 0.1, -1], sound.tolist())

        self.assertRaises(Exception, lambda : self.generator.from_list([]))


    def test_silence(self):
        sound = self.generator.silence()
        self.assertEqual([0] * self.beat, sound.tolist())


    def test_linear(self):
        sound = self.generator.linear(0, 0)
        self.assertEqual([0] * self.beat, sound.tolist())

        sound = self.generator.linear(1, 1)
        self.assertEqual([1] * self.beat, sound.tolist())

        sound = self.generator.linear(0.5, 0.5)
        self.assertEqual([0.5] * self.beat, sound.tolist())

        sound = self.generator.linear(-1, -1)
        self.assertEqual([-1] * self.beat, sound.tolist())

        sound = self.generator.linear(0, -1);
        self.assertTrue(numpy.array_equal(numpy.linspace(0, -1, self.beat),
```

```
57                                          sound.get_samples()))
58          self.assertEqual(self.beat, len(sound))
59
60          sound = self.generator.linear(1, -1);
61          self.assertTrue(numpy.array_equal(numpy.linspace(1, -1, self.beat),
62                                          sound.get_samples()))
63          self.assertEqual(self.beat, len(sound))
64
65          sound = self.generator.linear(0, 1);
66          self.assertTrue(numpy.array_equal(numpy.linspace(0, 1, self.beat),
67                                          sound.get_samples()))
68          self.assertEqual(self.beat, len(sound))
69
70          sound = self.generator.linear(-1, 1);
71          self.assertTrue(numpy.array_equal(numpy.linspace(-1, 1, self.beat),
72                                          sound.get_samples()))
73          self.assertEqual(self.beat, len(sound))
74
75          self.assertRaises(Exception, lambda : self.generator.linear(2, 0))
76          self.assertRaises(Exception, lambda : self.generator.linear(0, 2))
77
78
79      def test_noise(self):
80          sound = self.generator.noise(0)
81          self.assertEqual([0] * self.beat, sound.tolist())
82
83          sound = self.generator.noise(1)
84          self.assertElementsInRange(sound, -1, 1)
85          self.assertEqual(len(numpy.unique(sound.get_samples())), len(sound))
86          self.assertEqual(self.beat, len(sound))
87
88          sound = self.generator.noise(0.5)
89          self.assertElementsInRange(sound, -0.5, 0.5)
90          self.assertEqual(len(numpy.unique(sound.get_samples())), len(sound))
91          self.assertEqual(self.beat, len(sound))
92
93          sound = self.generator.noise(0.1)
94          self.assertElementsInRange(sound, -0.1, 0.1)
95          self.assertEqual(len(numpy.unique(sound.get_samples())), len(sound))
96          self.assertEqual(self.beat, len(sound))
97
98
99      def test_sine(self):
100         sound = self.generator.sine(1, 0)
101         self.assertEqual([0] * self.beat, sound.tolist())
102
103         sound = self.generator.sine(1, 1)
104         self.assertEqual(self.beat, len(sound))
105         self.assertElementsInRange(sound, -1, 1)
106
107         sound = self.generator.sine(4, 0.5)
108         self.assertEqual(self.beat, len(sound))
109         self.assertElementsInRange(sound, -0.5, 0.5)
110
111         self.assertRaises(Exception, lambda : self.generator.sine(0, 1))
112         self.assertRaises(Exception, lambda : self.generator.sine(-1, 1))
113         self.assertRaises(Exception, lambda : self.generator.sine(0.5, 1))
114         self.assertRaises(Exception, lambda : self.generator.sine(1, -1))
```

```
115         self.assertRaises(Exception, lambda : self.generator.sine(0, 2))
116
117
118 class TestSoundCases(MixerTestCase):
119
120
121     def test_add(self):
122
123         sound1 = self.generator.from_list([0, 1, -0.5])
124         sound2 = self.generator.from_list([0.5, -0.2, 0.5])
125         sound3 = sound1 + sound2
126         self.assertEqual([0.5, 0.8, 0], sound3.tolist())
127
128         sound1 = self.generator.from_list([0.1])
129         sound2 = self.generator.from_list([0, 0.4, 0.5])
130         sound3 = sound1 + sound2
131         self.assertEqual([0.1, 0.5, 0.6], sound3.tolist())
132         sound4 = sound2 + sound1
133         self.assertEqual(sound3, sound4)
134
135         sound1 = self.generator.from_list([0.1, -0.1])
136         sound2 = self.generator.from_list([0, 0.5, 0.5, 0.6])
137         sound3 = sound1 + sound2
138         self.assertEqual([0.1, 0.4, 0.6, 0.5], sound3.tolist())
139         sound4 = sound2 + sound1
140         self.assertEqual(sound3, sound4)
141
142
143     def test_sub(self):
144
145         sound1 = self.generator.from_list([0, 0, -0.5])
146         sound2 = self.generator.from_list([0.5, -0.2, 0.5])
147         sound3 = sound1 - sound2
148         self.assertEqual([-0.5, 0.2, -1], sound3.tolist())
149
150         sound1 = self.generator.from_list([0.1])
151         sound2 = self.generator.from_list([0, 0.2, 0.5])
152         sound3 = sound1 - sound2
153         self.assertEqual([0.1, -0.1, -0.4], sound3.tolist())
154
155         sound1 = self.generator.from_list([0.1])
156         sound2 = self.generator.from_list([0, 0.2, 0.5])
157         sound3 = sound2 - sound1
158         self.assertEqual([-0.1, 0.1, 0.4], sound3.tolist())
159
160         sound1 = self.generator.from_list([0.1, -0.1])
161         sound2 = self.generator.from_list([0, 0.5, 0.5, 0.6])
162         sound3 = sound1 - sound2
163         self.assertEqual([0.1, -0.6, -0.4, -0.7], sound3.tolist())
164
165         sound1 = self.generator.from_list([0.1, -0.1])
166         sound2 = self.generator.from_list([0, 0.5, 0.5, 0.6])
167         sound3 = sound2 - sound1
168         self.assertEqual([-0.1, 0.6, 0.4, 0.7], sound3.tolist())
169
170
171     def test_mul(self):
172
```

```
173         sound1 = self.generator.from_list([1, -1, 0.5])
174         sound2 = self.generator.from_list([0.5, 0.2, 0.5])
175         sound3 = sound1 * sound2
176         self.assertEqual([0.5, -0.2, 0.25], sound3.tolist())
177 
178         sound1 = self.generator.from_list([0.1])
179         sound2 = self.generator.from_list([0, 0.5, -1])
180         sound3 = sound1 * sound2
181         self.assertEqual([0, 0.05, -0.1], sound3.tolist())
182         sound4 = sound2 * sound1
183         self.assertEqual(sound3, sound4)
184 
185         sound1 = self.generator.from_list([0.1, -0.1])
186         sound2 = self.generator.from_list([0, 0.5, 0.5, 1])
187         sound3 = sound1 * sound2
188         self.assertEqual([0, -0.05, 0.05, -0.1], sound3.tolist())
189         sound4 = sound2 * sound1
190         self.assertEqual(sound3, sound4)
191 
192 
193     def test_div(self):
194 
195         sound1 = self.generator.from_list([0.5, 0.1, 0.8])
196         sound2 = self.generator.from_list([0.5, -0.2, 1])
197         sound3 = sound1 / sound2
198         self.assertEqual([1, -0.5, 0.8], sound3.tolist())
199 
200         sound1 = self.generator.from_list([0.1])
201         sound2 = self.generator.from_list([0.5, -0.2, 0.25])
202         sound3 = sound1 / sound2
203         self.assertEqual([0.2, -0.5, 0.4], sound3.tolist())
204 
205         sound1 = self.generator.from_list([0.5])
206         sound2 = self.generator.from_list([0, -0.2, 0.5])
207         sound3 = sound2 / sound1
208         self.assertEqual([0, -0.4, 1], sound3.tolist())
209 
210 
211     def test_mix(self):
212 
213         sound1 = self.generator.from_list([0.5, 0.6, 0])
214         sound2 = self.generator.from_list([0.5, 0.2, 1])
215         sound3 = sound1 & sound2
216         self.assertEqual([0.5, 0.4, 0.5], sound3.tolist())
217 
218         sound1 = self.generator.from_list([0.2])
219         sound2 = self.generator.from_list([0.5, -0.2, 0.6])
220         sound3 = sound1 & sound2
221         self.assertEqual([0.35, 0, 0.4], sound3.tolist())
222         sound4 = sound2 & sound1
223         self.assertEqual(sound3, sound4)
224 
225 
226     def test_concat(self):
227         sound1 = self.generator.from_list([0.1])
228         sound2 = self.generator.from_list([0.2, 0.3])
229 
230         self.assertEqual([0.1, 0.2, 0.3], (sound1 // sound2).tolist())
```

```
231
232
233     def test_loop(self):
234         sound1 = self.generator.from_list([1])
235         sound2 = self.generator.from_list([1, 1, 1])
236         self.assertEqual(sound2, sound1.loop(3))
237
238         sound1 = self.generator.from_list([1, 0.5])
239         sound2 = self.generator.from_list([1, 0.5, 1, 0.5, 1, 0.5])
240         self.assertEqual(sound2, sound1.loop(3))
241
242         self.assertRaises(Exception, lambda : sound1.loop(0))
243         self.assertRaises(Exception, lambda : sound1.loop(-1))
244
245     def test_resize(self):
246         sound1 = self.generator.from_list([0, 0.1, 0.2])
247
248         self.assertEqual(self.generator.from_list([0]), sound1.resize(1))
249         self.assertEqual(self.generator.from_list([0, 0.1]), sound1.resize(2))
250         self.assertEqual(self.generator.from_list([0, 0.1, 0.2]), sound1.resize(3))
251
252         self.assertEqual(
253             self.generator.from_list([0, 0.1, 0.2, 0.0]),
254             sound1.resize(4))
255
256         self.assertEqual(
257             self.generator.from_list([0, 0.1, 0.2, 0.0, 0.1]),
258             sound1.resize(5))
259
260         self.assertEqual(
261             self.generator.from_list([0, 0.1, 0.2, 0.0, 0.1, 0.2]),
262             sound1.resize(6))
263
264         self.assertRaises(Exception, lambda : sound1.resize(0))
265         self.assertRaises(Exception, lambda : sound1.resize(-1))
266         self.assertRaises(Exception, lambda : sound1.resize(0.5))
267         self.assertRaises(Exception, lambda : sound1.resize(1.5))
268
269
270     def test_resample(self):
271         sound1 = self.generator.from_list([0, 0.1, 0.2])
272
273         self.assertRaises(Exception, lambda : sound1.resample(0))
274         self.assertRaises(Exception, lambda : sound1.resample(-1))
275         self.assertRaises(Exception, lambda : sound1.resample(0.5))
276         self.assertRaises(Exception, lambda : sound1.resample(1.5))
277
278
279     def test_copy(self):
280         sound1 = self.generator.from_list([0, 0.1, 0.2])
281         copy = sound1.copy()
282
283         self.assertEqual(sound1, copy)
284         self.assertFalse(sound1 is copy)
285
286
287     def test_concat(self):
288         sound1 = self.generator.from_list([0.1])
```

```
289         sound2 = self.generator.from_list([0.2])
290 
291         self.assertEqual(self.generator.from_list([0.1, 0.2]), sound1.concat(sound2))
292 
293 
294     def test_fill(self):
295         sound1 = self.generator.from_list([0, 0.1, 0.2])
296 
297         self.assertEqual(
298             self.generator.from_list([0, 0.1, 0.2] + [0] * (self.beat - 3)),
299             sound1.fill(1))
300         self.assertEqual(
301             self.generator.from_list([0, 0.1, 0.2] + [0] * (self.beat * 2- 3)),
302             sound1.fill(2))
303 
304         sound2 = self.generator.from_list([0.1] * self.beat)
305         self.assertEqual(
306             sound2,
307             sound2.fill(1))
308 
309         self.assertRaises(Exception, lambda : sound1.fill(0))
310         self.assertRaises(Exception, lambda : sound1.fill(-1))
311 
312 
313     def test_reduce(self):
314         sound1 = self.generator.from_list([0, 0.1] * (self.beat - 1))
315         self.assertEqual(self.beat, len(sound1.reduce(1)))
316 
317         sound1 = self.generator.from_list([0.1] * (self.beat - 1))
318         self.assertEqual(len(sound1), len(sound1.reduce(1)))
319 
320         sound1 = self.generator.from_list([0, 0.1] * (self.beat - 1))
321         self.assertEqual(len(sound1), len(sound1.reduce(2)))
322 
323         self.assertRaises(Exception, lambda : sound1.reduce(0))
324         self.assertRaises(Exception, lambda : sound1.reduce(-1))
325 
326 
327     def test_expand(self):
328         sound1 = self.generator.from_list([0, 0.1, 0.2])
329         self.assertEqual(self.beat, len(sound1.expand(1)))
330         self.assertEqual(self.beat * 2, len(sound1.expand(2)))
331 
332         sound1 = self.generator.from_list([0, 0.1] * self.beat)
333         self.assertEqual(len(sound1), len(sound1.expand(1)))
334 
335         self.assertRaises(Exception, lambda : sound1.expand(0))
336         self.assertRaises(Exception, lambda : sound1.expand(-1))
337 
338 
339     def test_tolist(self):
340         sound1 = self.generator.from_list([0, 0.1, 0.2])
341         self.assertEqual([0, 0.1, 0.2], sound1.tolist())
342 
343 
344     def test_resample(self):
345         sound1 = self.generator.from_list([0, 0.1, 0.2, 0.3])
346 
```

```
347         self.assertEqual(
348             self.generator.from_list([0, 0.1, 0.2, 0.3]),
349             sound1.resample(4))
350 
351         self.assertEqual(1, len(sound1.resample(1)))
352         self.assertEqual(3, len(sound1.resample(3)))
353         self.assertEqual(6, len(sound1.resample(6)))
354         self.assertEqual(10, len(sound1.resample(10)))
355 
356         self.assertRaises(Exception, lambda : sound1.resample(0))
357         self.assertRaises(Exception, lambda : sound1.resample(-1))
358         self.assertRaises(Exception, lambda : sound1.resample(0.5))
359         self.assertRaises(Exception, lambda : sound1.resample(1.5))
360 
361 class TestCustomCases(MixerTestCase):
362     pass
363 
364 if __name__ == '__main__':
365     unittest.main()
```

6. test_parser.py

```
import minicollider.parser as parser
import unittest

class ParserTestCase(unittest.TestCase):


    def setUp(self):

        self.beat = 8
        self.sample_rate = self.beat * 12

        parser.init(self.sample_rate, self.beat, 0)
        self.generator = parser.generator


    def assertParseEqualList(self, list, input):
        self.assertParseEqual(self.generator.from_list(list), input)


    def assertParseFail(self, input):
        self.assertRaises(Exception, lambda : parser.parse(input),
            'Se esperaba un error para el input: %s' % input)


    def assertParseAllFail(self, input_list):
        for input in input_list:
            self.assertParseFail(input)


    def assertParseEqual(self, sound, input):
        parsed_sound = parser.parse(input)
        self.assertEqual(sound, parsed_sound, '%s != %s' % (sound, parsed_sound))


    def assertParseAllEqual(self, sound, input_list):
        for input in input_list:
            self.assertParseEqual(sound, input)


    def assertElementsInRange(self, list, min, max):
        for item in list:
            self.assertTrue(min <= item <= max)


class TestGeneratorsCases(ParserTestCase):


    def test_manual(self):
        self.assertParseAllEqual(
            self.generator.from_list([0]),
            ['0', '{0}', '{ 0}', '{0 }', '{ 0 }']
        )

        self.assertParseAllEqual(
            self.generator.from_list([1]),
            ['1', '{1}', '{ 1}', '{1 }', '{ 1 }']
```

```
57          )
58
59          self.assertParseAllEqual(
60              self.generator.from_list([-1]),
61              ['-1', '{-1}', '{ -1}', '{-1 }', '{ -1 }']
62          )
63
64          self.assertParseAllEqual(
65              self.generator.from_list([1]),
66              ['1.0', '{1.0}', '{ 1.0}', '{1.0 }', '{ 1.0 }']
67          )
68
69          self.assertParseAllEqual(
70              self.generator.from_list([-0.5]),
71              ['-0.5', '{-0.5}', '{ -0.5}', '{-0.5 }', '{ -0.5 }']
72          )
73
74          self.assertParseAllFail(
75              ['{', '}', '{ }', '}{', 'a', '-a', 'asd1']
76          )
77
78          self.assertParseAllFail(
79              ['2', '2.0', '1.1', '-2', '-2.1', '0.1.0']
80          )
81
82          self.assertParseAllFail(
83              ['{2}', '{2.0}', '{1.1}', '{-2}', '{-2.1}']
84          )
85
86
87      def test_silence(self):
88          self.assertParseAllEqual(self.generator.silence(),
89              ['sil', 'sil()', '{sil}', '{sil()}']
90          )
91
92          self.assertParseAllEqual(self.generator.silence(),
93              ['silence', 'silence()', '{silence}', '{silence()}']
94          )
95
96          self.assertParseAllFail(['Sil', 'sile()', 'Silence', 'sil ence'])
97
98          self.assertParseAllFail(['sil(1)', 'sil(1,2)'])
99
100
101     def test_sine(self):
102         self.assertParseEqual(self.generator.sine(1, 1), 'sin(1)')
103         self.assertParseEqual(self.generator.sine(5, 1), 'sin(5)')
104         self.assertParseEqual(self.generator.sine(11, 1), 'sin(11)')
105
106         self.assertParseEqual(self.generator.sine(1, 0.5), 'sin(1, 0.5)')
107         self.assertParseEqual(self.generator.sine(1, 0), 'sin(1, 0)')
108         self.assertParseEqual(self.generator.sine(10, 0.2), 'sin(10, 0.2)')
109
110         self.assertParseAllFail(['sin', 'sin()', 'sin(0)','sin(1.0)', 'sin(-1)'])
111         self.assertParseAllFail(['sin(1, 0)', 'sin(1, 2)', 'sin(1, -1)'])
112
113
114     def test_linear(self):
```

```
115         self.assertParseAllEqual(self.generator.linear(0, 1),
116             ['linear(0, 1)', 'lin(0, 1)'])
117 
118         self.assertParseAllEqual(self.generator.linear(0.5, 0.5),
119             ['linear(0.5, 0.5)', 'lin(0.5, 0.5)'])
120 
121         self.assertParseAllEqual(self.generator.linear(-0.5, 0.5),
122             ['linear(-0.5, 0.5)', 'lin(-0.5, 0.5)'])
123 
124         self.assertParseAllEqual(self.generator.linear(0.5, -0.5),
125             ['linear(0.5, -0.5)', 'lin(0.5, -0.5)'])
126 
127         self.assertParseAllEqual(self.generator.linear(-0.5, -0.5),
128             ['linear(-0.5, -0.5)', 'lin(-0.5, -0.5)'])
129 
130         self.assertParseAllFail(
131             ['linear', 'linear()', 'lin', 'lin()', 'lin(0)', 'lin(0, 0, 0)'])
132 
133         self.assertParseAllFail(
134             ['lin(2, 0)', 'lin(0, 2)', 'lin(-2, 0)', 'lin(0, -2)',])
135 
136 
137     def test_noise(self):
138         sound = parser.parse('noi')
139         self.assertEqual(self.beat, len(sound))
140 
141         sound = parser.parse('noi()')
142         self.assertEqual(self.beat, len(sound))
143 
144         sound = parser.parse('noise')
145         self.assertEqual(self.beat, len(sound))
146 
147         sound = parser.parse('noise()')
148         self.assertEqual(self.beat, len(sound))
149 
150         sound = parser.parse('noi(0)')
151         self.assertEqual(self.generator.silence(), sound)
152 
153         sound = parser.parse('noi(0.5)')
154         self.assertEqual(self.beat, len(sound))
155         self.assertElementsInRange(sound, -0.5, 0.5)
156 
157         self.assertParseAllFail(['noi(2)', 'noi(-2)'])
158 
159 
160 class TestOperatorCases(ParserTestCase):
161 
162 
163     def test_add(self):
164 
165         self.assertParseEqualList([3], '1 + 2')
166         self.assertParseEqualList([6], '1 + 2 + 3')
167         self.assertParseEqualList([-1], '1 + 2 + -4')
168         self.assertParseEqualList([2, 5, -1], '{1; 2; 3} + {1; 3; -4}')
169         self.assertParseEqualList([2, 5, -1], '{1; 4; -2} + {1}')
170         self.assertParseEqualList([2, 5, -1], '{1; 4; -2} + 1')
171         self.assertParseEqualList([2, 5, -1, 0], '{1; 3; -2; -2} + {1; 2}')
172         self.assertParseEqualList([2, 5, -1, 0], '{1; 2} + {1; 3; -2; -2}')
```

```
173
174
175     def test_sub(self):
176
177         self.assertParseEqualList([-1], '1 - 2')
178         self.assertParseEqualList([-4], '1 - 2 - 3')
179         self.assertParseEqualList([3], '1 - 2 - -4')
180         self.assertParseEqualList([0, -1, 7], '{1; 2; 3} - {1; 3; -4}')
181         self.assertParseEqualList([0, 3, -3], '{1; 4; -2} - {1}')
182         self.assertParseEqualList([0, 3, -3], '{1; 4; -2} - 1')
183         self.assertParseEqualList([0, 1, -3, -4], '{1; 3; -2; -2} - {1; 2}')
184         self.assertParseEqualList([0, -1, 3, 4], '{1; 2} - {1; 3; -2; -2}')
185
186
187     def test_mul(self):
188
189         self.assertParseEqualList([2], '1 * 2')
190         self.assertParseEqualList([6], '1 * 2 * 3')
191         self.assertParseEqualList([-8], '1 * 2 * -4')
192         self.assertParseEqualList([1, 6, -12], '{1; 2; 3} * {1; 3; -4}')
193         self.assertParseEqualList([2, 8, -4], '{1; 4; -2} * {2}')
194         self.assertParseEqualList([2, 8, -4], '{1; 4; -2} * 2')
195         self.assertParseEqualList([1, 6, -2, -4], '{1; 3; -2; -2} * {1; 2}')
196         self.assertParseEqualList([1, 6, -2, -4], '{1; 2} * {1; 3; -2; -2}')
197
198
199     def test_div(self):
200
201         self.assertParseEqualList([5], '10 / 2')
202         self.assertParseEqualList([2], '12 / 2 / 3')
203         self.assertParseEqualList([-2], '16 / 2 / -4')
204         self.assertParseEqualList([3, 0.5, -0.25], '{3; 2; 1} / {1; 4; -4}')
205         self.assertParseEqualList([5, 2, -0.5], '{10; 4; -1} / {2}')
206         self.assertParseEqualList([5, 2, -0.5], '{10; 4; -1} / 2')
207         self.assertParseEqualList([1, 1.5, -2, -1], '{1; 3; -2; -2} / {1; 2}')
208         self.assertParseEqualList([1, 1, -0.5, -1.5], '{1; 3} / {1; 3; -2; -2}')
209
210
211     def test_div(self):
212
213         self.assertParseEqualList([5], '10 / 2')
214         self.assertParseEqualList([2], '12 / 2 / 3')
215         self.assertParseEqualList([-2], '16 / 2 / -4')
216         self.assertParseEqualList([3, 0.5, -0.25], '{3; 2; 1} / {1; 4; -4}')
217         self.assertParseEqualList([5, 2, -0.5], '{10; 4; -1} / {2}')
218         self.assertParseEqualList([5, 2, -0.5], '{10; 4; -1} / 2')
219         self.assertParseEqualList([1, 1.5, -2, -1], '{1; 3; -2; -2} / {1; 2}')
220         self.assertParseEqualList([1, 1, -0.5, -1.5], '{1; 3} / {1; 3; -2; -2}')
221
222
223     def test_mix(self):
224
225         self.assertParseEqualList([6], '10 & 2')
226         self.assertParseEqualList([5], '12 & 2 & 3')
227         self.assertParseEqualList([2.5], '16 & 2 & -4')
228         self.assertParseEqualList([2, 3, -1.5], '{3; 2; 1} & {1; 4; -4}')
229         self.assertParseEqualList([6, 3, 0.5], '{10; 4; -1} & {2}')
230         self.assertParseEqualList([6, 3, 0.5], '{10; 4; -1} & 2')
```

```
231         self.assertParseEqualList([1, 2.5, -0.5, 0], '{1; 3; -2; -2} & {1; 2}')
232         self.assertParseEqualList([1, 2.5, -0.5, 0], '{1; 2} & {1; 3; -2; -2}')
233 
234 
235     def test_concat(self):
236 
237         self.assertParseEqualList([10, 2], '10 ; 2')
238         self.assertParseEqualList([10, 2, -3], '10 ; 2 ; -3')
239 
240         self.assertParseEqualList([10, 2], '{10} ; {2}')
241         self.assertParseEqualList([10, 1, 2], '{10; 1} ; {2}')
242         self.assertParseEqualList([10, 2, 1], '{10} ; {2; 1}')
243         self.assertParseEqualList([1, 2, 3, 4, 5, 6], '{1; 2} ; {3; 4} ; {5; 6}')
244 
245 
246 class TestMethodsCases(ParserTestCase):
247 
248 
249     def test_loop(self):
250 
251         self.assertParseEqualList([5], '5.loop()')
252         self.assertParseEqualList([5], '5.loop')
253 
254         self.assertParseEqualList([1, 1], '1.loop(2)')
255         self.assertParseEqualList([1, 1], '{1}.loop(2)')
256 
257         self.assertParseEqualList([1, 2], '{1 ; 2}.loop(1)')
258         self.assertParseEqualList([1, 2] * 2, '{1 ; 2}.loop(2)')
259         self.assertParseEqualList([1, 2] * 10, '{1 ; 2}.loop(10)')
260 
261         self.assertParseEqualList([1], '{1 ; 2}.loop(0.5)')
262         self.assertParseEqualList([1, 2, 1], '{1 ; 2}.loop(1.5)')
263 
264         self.assertParseFail('1.loop(0)')
265         self.assertParseFail('1.loop(-1)')
266 
267 
268     def test_fill(self):
269 
270         self.assertParseEqualList([1] + [0] * (self.beat - 1), '1.fill(1)')
271         self.assertParseEqualList([1] + [0] * (self.beat - 1), '1.fill()')
272         self.assertParseEqualList([1] + [0] * (self.beat - 1), '1.fill')
273 
274         self.assertParseEqualList([1, 2, 3] + [0] * (self.beat - 3), '{1;2;3}.fill')
275         self.assertParseEqualList([1, 2, 3] + [0] * (self.beat * 2 - 3), '{1;2;3}.fill(2)')
276 
277         self.assertParseEqualList([1 ] + [0] * (self.beat / 2 - 1), '1.fill(0.5)')
278         self.assertParseEqualList([1, 2, 3] * 1000, '{1;2;3}.loop(1000).fill')
279 
280         self.assertParseFail('1.fill(0)')
281         self.assertParseFail('1.fill(-1)')
282 
283 
284     def test_reduce(self):
285 
286         self.assertParseEqualList([1], '1.reduce')
287         self.assertParseEqualList([1], '1.reduce()')
288         self.assertParseEqualList([1, 2] * (self.beat / 2), '{1; 2}.loop(100).reduce')
```

```
289         self.assertParseEqualList([2] * self.beat, '2.loop(1000).reduce()')
290         self.assertParseEqualList([2] * (self.beat * 2), '2.loop(1000).reduce(2)')
291 
292         self.assertParseFail('1.reduce(0)')
293         self.assertParseFail('1.reduce(-1)')
294 
295 
296     def test_expand(self):
297 
298         self.assertParseEqualList([1] * self.beat, '1.expand')
299         self.assertParseEqualList([1] * self.beat, '1.expand()')
300         self.assertParseEqualList([1] * (self.beat / 2) + [2] * (self.beat / 2), '{1;2}.expand()')
301         self.assertParseEqualList([1] * self.beat + [2] * self.beat , '{1;2}.expand(2)')
302         self.assertParseEqualList([1] * 1000, '1.loop(1000).expand()')
303 
304         self.assertParseFail('1.expand(0)')
305         self.assertParseFail('1.expand(-1)')
306 
307 
308 class TestPrecedenceCases(ParserTestCase):
309 
310 
311     def test_add_mul(self):
312         self.assertParseEqualList([7], '1 + 2 * 3')
313         self.assertParseEqualList([7], '2 * 3 + 1')
314         self.assertParseEqualList([8], '2 * {3 + 1}')
315         self.assertParseEqualList([8], '{3 + 1} * 2')
316 
317 
318     def test_add_div(self):
319 
320         self.assertParseEqualList([4], '1 + 9 / 3')
321         self.assertParseEqualList([3], '6 / 3 + 1')
322         self.assertParseEqualList([2], '8 / {3 + 1}')
323         self.assertParseEqualList([2], '{3 + 1} / 2')
324 
325 
326     def test_add_sub(self):
327 
328         self.assertParseEqualList([0], '1 + 2 - 3')
329         self.assertParseEqualList([2], '1 - 2 + 3')
330         self.assertParseEqualList([-4], '1 - 2 + -3')
331         self.assertParseEqualList([6], '1 + 2 - -3')
332 
333 
334     def test_add_concat(self):
335 
336         self.assertParseEqualList([6, 3], '1 + 5 ; 3')
337         self.assertParseEqualList([6, 3, 1.5], '1 + 5 ; 3 ; 0.5 + 1')
338 
339 
340     def test_sub_concat(self):
341 
342         self.assertParseEqualList([-4, 3], '1 - 5 ; 3')
343         self.assertParseEqualList([-4, 3, -0.5], '1 - 5 ; 3 ; 0.5 - 1')
344 
345 
346     def test_mul_concat(self):
```

```
347
348         self.assertParseEqualList([10, 3], '2 * 5 ; 3')
349         self.assertParseEqualList([10, 3, 1], '2 * 5 ; 3 ; 0.5 * 2')
350
351
352     def test_div_concat(self):
353
354         self.assertParseEqualList([2, 3], '10 / 5 ; 3')
355         self.assertParseEqualList([2, 3, 0.25], '10 / 5 ; 3 ; 0.5 / 2')
356
357
358     def test_loop_concat(self):
359
360         self.assertParseEqualList([1, 1, 1, 3], '1.loop(3) ; 3')
361         self.assertParseEqualList([3, 1, 1, 1], '3 ; 1.loop(3)')
362
363
364     def test_expand_concat(self):
365
366         self.assertParseEqualList([1] * self.beat + [3], '1.expand ; 3')
367         self.assertParseEqualList([3] + [1] * self.beat, '3 ; 1.expand')
368
369
370     def test_reduce_concat(self):
371
372         self.assertParseEqualList([1, 2, 3], '{1;2}.reduce ; 3')
373         self.assertParseEqualList([3, 1, 2], '3 ; {1;2}.reduce')
374
375
376     def test_fill_concat(self):
377
378         self.assertParseEqualList([1, 2] + [0] * (self.beat - 2) + [3], '{1;2}.fill ; 3')
379         self.assertParseEqualList([3] + [1, 2] + [0] * (self.beat - 2), '3 ; {1;2}.fill')
380
381
382     def test_add_mix(self):
383
384         self.assertParseEqualList([4.5], '1 + 5 & 3')
385         self.assertParseEqualList([4], '3 & 4 + 1')
386
387     def test_loop_mix(self):
388
389         self.assertParseEqualList([2, 2, 2], '1.loop(3) & 3')
390         self.assertParseEqualList([2, 2, 2], '3 & 1.loop(3)')
391
392
393 class TestCustomCases(ParserTestCase):
394
395
396     def test_enunciado(self):
397
398         self.assertParseEqualList([1, 2], '1; 2')
399         self.assertParseEqualList([1, 2], '{1; 2}')
400         self.assertParseEqualList([7], '{1 + 2*3}')
401         self.assertParseEqualList([6, 3, 6], '{{2;1} mul {3;3;3}}')
402         self.assertParseEqualList([0, 1], '{1-1;2-1}')
403         self.assertParseEqualList([2, 3, 4], '{4+1*-2 & {2;4;6}}')
404         self.assertParseEqualList([0, 1, 1], '{0;1.loop(2)}')
```

```
405         self.assertParseEqualList([0, 1], '{0;1;2;3}.loop(0.5)')
406
407
408 if __name__ == '__main__':
409     unittest.main()
```