



**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-herandez@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>

Contents

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

```
1 import minicollider.parser
2 try:
3     import argparse
4 except ImportError:
5     from minicollider.external import argparse
6
7
8 def parsear_argumentos():
9     argparser = argparse.ArgumentParser(formatter_class=
10                                         argparse.ArgumentDefaultsHelpFormatter)
11     argparser.add_argument('-s', '--samplerate',
12                           help="The desired sample rate.",
13                           default=8000,
14                           type=int)
15     argparser.add_argument('-b', '--beat',
16                           help="The desired beat.",
17                           default=8000 / 12,
18                           type=int)
19     argparser.add_argument('-f', '--file',
20                           help="A file with a buffer to parse (optional).")
21     return argparser.parse_args()
22
23
24 def parsear_archivo(file):
25     try:
26         archivo = open(args.file, 'r')
27         entrada = archivo.read()
28         archivo.close()
29     except IOError:
30         print 'Error opening the file.'
31         exit(1)
32     try:
33         minicollider.parser.parse(entrada)
34     except Exception, e:
35         if e: msg = str(e)
36         else: msg = 'Syntax Error'
37         print "Error: %s" % msg
38
39 def prompt():
40     while 1:
41         try:
42             entrada = raw_input('buffer > ')
43         except EOFError:
44             print
45             break
46         if entrada != '':
47             try:
48                 minicollider.parser.parse(entrada)
49             except Exception, e:
50                 print "Error: %s" % e
51
52
53
54 if __name__ == '__main__':
55     args = parsear_argumentos()
56     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

```

1  # -----
2  # lexer.py
3  #
4  # Lexer para el mini-collider
5  # -----
6  try:
7      import ply.lex as lex
8  except ImportError:
9      import external.lex as lex
10 import re
11
12 tokens = (
13     'NUM', 'SIN', 'LIN', 'SIL', 'NOI', 'PLAY', 'POST', 'LOOP',
14     'TUNE', 'FILL', 'REDU', 'EXPA', 'CON', 'MIX', 'ADD', 'SUB', 'MUL',
15     'DIV', 'LPAREN', 'RPAREN', 'LLAVE', 'RLLAVE', 'PLOT', 'COMA',
16 )
17
18 # Tokens
19
20 t_SIN      = r'sin'
21 t_LIN      = r'linear|lin'
22 t_SIL      = r'silence|sil'
23 t_NOI      = r'noise|noi'
24 t_PLAY     = r'.play'
25 t_POST     = r'.post'
26 t_LOOP     = r'.loop'
27 t_TUNE     = r'.tune'
28 t_FILL     = r'.fill'
29 t_REDU     = r'.reduce'
30 t_EXPA     = r'.expand'
31 t_PLOT     = r'.plot'
32 t_CON      = r'con|;'
33 t_MIX      = r'mix|&'
34 t_ADD      = r'add|\+'
35 t_SUB      = r'sub|-'
36 t_MUL      = r'mul|\*'
37 t_DIV      = r'div|/'
38 t_LPAREN   = r'\('
39 t_RPAREN   = r'\)'
40 t_LLLAVE   = r'\{'
41 t_RLLAVE   = r'\}'
42 t_COMA     = r','
43 t_ignore_COMM = r'//.*\n'
44 t_ignore_WS  = r'\s|\t|\n'
45
46 def t_NUM(t):
47     r'\d+(\.\d+)?'
48     if t.value.find('.') == -1:
49         t.value = int(t.value)
50     else:
51         t.value = float(t.value)
52
53     return t
54
55 def t_error(t):
56     raise SyntaxError("Caracter ilegal: '%s'" % t.value[0])

```

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

3 mixer.py

```
1  import math
2  import numpy
3  import pylab
4  import pygame
5
6
7  NUMPY_ENCODING = numpy.int16
8  MIXER_ENCODING = -16
9
10 MAX_AMPLITUDE = 32000
11 SAMPLE_RATE = 8800
12 BEAT = SAMPLE_RATE / 12
13
14
15 def init(sample_rate=8800, beat=8800/12, init_pygame=1):
16     global SAMPLE_RATE, BEAT
17
18     SAMPLE_RATE = sample_rate
19     BEAT = beat
20
21     if init_pygame:
22         pygame.mixer.pre_init(SAMPLE_RATE, MIXER_ENCODING, 1)
23         pygame.init()
24
25
26 class Sound():
27     def __init__(self, samples):
28         if (len(samples)) == 0:
29             raise Exception('No se puede crear un buffer vacio')
30
31         self.samples = numpy.array(samples, numpy.float)
32
33     def __eq__(self, other):
34         return numpy.array_equal(self.samples, other.samples)
35
36     def __iter__(self):
37         return self.samples.__iter__()
38
39     def __len__(self):
40         return len(self.samples)
41
42     def __add__(self, other):
43         return self._oper(other, (lambda x, y: x + y))
44
45     def __mul__(self, other):
46         return self._oper(other, (lambda x, y: x * y))
47
48     def __sub__(self, other):
49         return self._oper(other, (lambda x, y: x - y))
50
51     def __div__(self, other):
52         return self._oper(other, (lambda x, y: x / y))
53
54     def __floordiv__(self, other):
55         return self.concat(other)
56
```



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

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

4 parser.py

```

1  # -----
2  # parser.py
3  #
4  # Parser para el mini-collider
5  # -----
6  try:
7      import ply.yacc as yacc
8  except ImportError:
9      import external.yacc as yacc
10 from lexer import tokens
11 import mixer
12
13 generator = None
14
15
16 def init(sample_rate, beat, init_pygame=1):
17     global generator
18     mixer.init(sample_rate, beat, init_pygame)
19     generator = mixer.SoundGenerator()
20
21
22 def parse(input):
23     try:
24         res = parser.parse(input)
25     except Exception, e:
26         raise SyntaxError("%s" % e)
27     return res
28
29 precedence = (
30     ('left', 'CON', 'MIX'),
31     ('nonassoc', 'LOOP', 'POST', 'TUNE', 'EXPA', 'REDU', 'FILL', 'PLAY', 'PLOT'),
32     ('left', 'ADD', 'SUB'),
33     ('left', 'MUL', 'DIV'),
34 )
35
36
37 def p_statement_expr(t):
38     'START : BUFFER'
39
40     t[0] = t[1]
41
42
43 def p_BUFFER_binop(t):
44     '''BUFFER :      BUFFER CON BUFFER
45                  /      BUFFER MIX BUFFER
46                  /      BUFFER ADD BUFFER
47                  /      BUFFER SUB BUFFER
48                  /      BUFFER MUL BUFFER
49                  /      BUFFER DIV BUFFER'''
50
51     if t[2] in ['con', ';']: t[0] = t[1] // t[3]
52     elif t[2] in ['mix', '&']: t[0] = t[1] & t[3]
53     elif t[2] in ['add', '+']: t[0] = t[1] + t[3]
54     elif t[2] in ['sub', '-']: t[0] = t[1] - t[3]
55     elif t[2] in ['mul', '*']: t[0] = t[1] * t[3]
56     elif t[2] in ['div', '/']: t[0] = t[1] / t[3]

```

```

57
58
59 def p_BUFFER_metodo_Oparam(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_Oparam(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

```

1  import minicollider.mixer as mixer
2  import numpy
3  import unittest
4
5  class MixerTestCase(unittest.TestCase):
6
7
8      def assertElementsInRange(self, list, min, max):
9          for item in list:
10             self.assertTrue(min <= item <= max)
11
12      def setUp(self):
13          self.sample_rate = 4800
14          self.beat = self.sample_rate / 12
15
16          mixer.init(self.sample_rate, self.beat, 0)
17          self.generator = mixer.SoundGenerator()
18
19
20  class TestSoundGeneratorCases(MixerTestCase):
21
22
23      def test_from_list(self):
24
25          sound = self.generator.from_list([1])
26          self.assertEqual([1], sound.tolist())
27
28          sound = self.generator.from_list([0.5])
29          self.assertEqual([0.5], sound.tolist())
30
31          sound = self.generator.from_list([0, 0.1, -1])
32          self.assertEqual([0, 0.1, -1], sound.tolist())
33
34          self.assertRaises(Exception, lambda : self.generator.from_list([]))
35
36
37      def test_silence(self):
38          sound = self.generator.silence()
39          self.assertEqual([0] * self.beat, sound.tolist())
40
41
42      def test_linear(self):
43          sound = self.generator.linear(0, 0)
44          self.assertEqual([0] * self.beat, sound.tolist())
45
46          sound = self.generator.linear(1, 1)
47          self.assertEqual([1] * self.beat, sound.tolist())
48
49          sound = self.generator.linear(0.5, 0.5)
50          self.assertEqual([0.5] * self.beat, sound.tolist())
51
52          sound = self.generator.linear(-1, -1)
53          self.assertEqual([-1] * self.beat, sound.tolist())
54
55          sound = self.generator.linear(0, -1);
56          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

```

1  import minicollider.parser as parser
2  import unittest
3
4  class ParserTestCase(unittest.TestCase):
5
6
7      def setUp(self):
8
9          self.beat = 8
10         self.sample_rate = self.beat * 12
11
12         parser.init(self.sample_rate, self.beat, 0)
13         self.generator = parser.generator
14
15
16     def assertParseEqualList(self, list, input):
17         self.assertParseEqual(self.generator.from_list(list), input)
18
19
20     def assertParseFail(self, input):
21         self.assertRaises(Exception, lambda : parser.parse(input),
22             'Se esperaba un error para el input: %s' % input)
23
24
25     def assertParseAllFail(self, input_list):
26         for input in input_list:
27             self.assertParseFail(input)
28
29
30     def assertParseEqual(self, sound, input):
31         parsed_sound = parser.parse(input)
32         self.assertEqual(sound, parsed_sound, '%s != %s' % (sound, parsed_sound))
33
34
35     def assertParseAllEqual(self, sound, input_list):
36         for input in input_list:
37             self.assertParseEqual(sound, input)
38
39
40     def assertElementsInRange(self, list, min, max):
41         for item in list:
42             self.assertTrue(min <= item <= max)
43
44
45     class TestGeneratorsCases(ParserTestCase):
46
47
48     def test_manual(self):
49         self.assertParseAllEqual(
50             self.generator.from_list([0]),
51             ['0', '{0}', '{ 0}', '{0 }', '{ 0 }'])
52     )
53
54     self.assertParseAllEqual(
55         self.generator.from_list([1]),
56         ['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] * (self.beat / 2), '{1;2}.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()
```