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

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

```python
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:
        print "Error: %s" % e

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)
    if args.file is not None:
        parsear_archivo(args.file)
```

```
57    else:
58        prompt()
```

2.   lexer.py

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

3.   mixer.py

```python
import math
import numpy
import pylab
import pygame


NUMPY_ENCODING = numpy.int16
MIXER_ENCODING = -16

AMPLITUDE_MULT = 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):
        "samples tiene que ser un array de numpy"
        if (len(samples)) == 0:
            raise Exception('No se puede crear un buffer vacio')

        for x in samples:
            if (x > 1 or x < -1):
                raise Exception('Los elementos del buffer deben ser -1 <= x <= 1: %s' % x)

        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):
```

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

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

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

4. parser.py

```
1   # ------------------------------------------------------------------------------
2   # parser.py
3   #
4   # Parser para el mini-collider
5   # ------------------------------------------------------------------------------
6   import ply.yacc as yacc
7   from lexer import tokens
8   import mixer
9
10  generator = None
11
12
13  def init(sample_rate, beat, init_pygame=1):
14      global generator
15      mixer.init(sample_rate, beat, init_pygame)
16      generator = mixer.SoundGenerator()
17
18
19  def parse(input):
20      try:
21          res = parser.parse(input)
22      except Exception, e:
23          raise SyntaxError("%s" % e)
24      return res
25
26  precedence = (
27      ('left', 'CON', 'MIX'),
28      ('left', 'ADD', 'SUB'),
29      ('left', 'MUL', 'DIV'),
30  )
31
32
33  def p_statement_expr(t):
34      'START : BUFFER'
35
36      t[0] = t[1]
37
38
39  def p_BUFFER_binop(t):
40      '''BUFFER :        BUFFER CON BUFFER
41                 |       BUFFER MIX BUFFER
42                 |       BUFFER ADD BUFFER
43                 |       BUFFER SUB BUFFER
44                 |       BUFFER MUL BUFFER
45                 |       BUFFER DIV BUFFER'''
46
47      if   t[2] in ['con', ';']: t[0] = t[1] // t[3]
48      elif t[2] in ['mix', '&']: t[0] = t[1] & t[3]
49      elif t[2] in ['add', '+']: t[0] = t[1] + t[3]
50      elif t[2] in ['sub', '-']: t[0] = t[1] - t[3]
51      elif t[2] in ['mul', '*']: t[0] = t[1] * t[3]
52      elif t[2] in ['div', '/']: t[0] = t[1] / t[3]
53
54
55  def p_BUFFER_metodo_0param(t):
56      '''BUFFER :       BUFFER PLAY ONEPARAM
```

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

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

```
173        raise SyntaxError
174
175 parser = yacc.yacc()
```

5.   test_mixer.py

```python
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          self.assertRaises(Exception, lambda : self.generator.from_list([2]))
36          self.assertRaises(Exception, lambda : self.generator.from_list([-2]))
37
38
39      def test_silence(self):
40          sound = self.generator.silence()
41          self.assertEqual([0] * self.beat, sound.tolist())
42
43
44      def test_linear(self):
45          sound = self.generator.linear(0, 0)
46          self.assertEqual([0] * self.beat, sound.tolist())
47
48          sound = self.generator.linear(1, 1)
49          self.assertEqual([1] * self.beat, sound.tolist())
50
51          sound = self.generator.linear(0.5, 0.5)
52          self.assertEqual([0.5] * self.beat, sound.tolist())
53
54          sound = self.generator.linear(-1, -1)
55          self.assertEqual([-1] * self.beat, sound.tolist())
56
```

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

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

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

16

```
231
232            self.assertEqual([0.1, 0.2, 0.3], (sound1 // sound2).tolist())
233
234
235        def test_loop(self):
236            sound1 = self.generator.from_list([1])
237            sound2 = self.generator.from_list([1, 1, 1])
238            self.assertEqual(sound2, sound1.loop(3))
239
240            sound1 = self.generator.from_list([1, 0.5])
241            sound2 = self.generator.from_list([1, 0.5, 1, 0.5, 1, 0.5])
242            self.assertEqual(sound2, sound1.loop(3))
243
244            self.assertRaises(Exception, lambda : sound1.loop(0))
245            self.assertRaises(Exception, lambda : sound1.loop(-1))
246            self.assertRaises(Exception, lambda : sound1.loop(0.5))
247            self.assertRaises(Exception, lambda : sound1.loop(1.5))
248
249
250        def test_resize(self):
251            sound1 = self.generator.from_list([0, 0.1, 0.2])
252
253            self.assertEqual(self.generator.from_list([0]), sound1.resize(1))
254            self.assertEqual(self.generator.from_list([0, 0.1]), sound1.resize(2))
255            self.assertEqual(self.generator.from_list([0, 0.1, 0.2]), sound1.resize(3))
256
257            self.assertEqual(
258                self.generator.from_list([0, 0.1, 0.2, 0.0]),
259                sound1.resize(4))
260
261            self.assertEqual(
262                self.generator.from_list([0, 0.1, 0.2, 0.0, 0.1]),
263                sound1.resize(5))
264
265            self.assertEqual(
266                self.generator.from_list([0, 0.1, 0.2, 0.0, 0.1, 0.2]),
267                sound1.resize(6))
268
269            self.assertRaises(Exception, lambda : sound1.resize(0))
270            self.assertRaises(Exception, lambda : sound1.resize(-1))
271            self.assertRaises(Exception, lambda : sound1.resize(0.5))
272            self.assertRaises(Exception, lambda : sound1.resize(1.5))
273
274
275        def test_resample(self):
276            sound1 = self.generator.from_list([0, 0.1, 0.2])
277
278            self.assertRaises(Exception, lambda : sound1.resample(0))
279            self.assertRaises(Exception, lambda : sound1.resample(-1))
280            self.assertRaises(Exception, lambda : sound1.resample(0.5))
281            self.assertRaises(Exception, lambda : sound1.resample(1.5))
282
283
284        def test_copy(self):
285            sound1 = self.generator.from_list([0, 0.1, 0.2])
286            copy = sound1.copy()
287
288            self.assertEqual(sound1, copy)
```

```
289             self.assertFalse(sound1 is copy)
290
291
292     def test_concat(self):
293             sound1 = self.generator.from_list([0.1])
294             sound2 = self.generator.from_list([0.2])
295
296             self.assertEqual(self.generator.from_list([0.1, 0.2]), sound1.concat(sound2))
297
298
299     def test_fill(self):
300             sound1 = self.generator.from_list([0, 0.1, 0.2])
301
302             self.assertEqual(
303                 self.generator.from_list([0, 0.1, 0.2] + [0] * (self.beat - 3)),
304                 sound1.fill(1))
305             self.assertEqual(
306                 self.generator.from_list([0, 0.1, 0.2] + [0] * (self.beat * 2- 3)),
307                 sound1.fill(2))
308
309             sound2 = self.generator.from_list([0.1] * self.beat)
310             self.assertEqual(
311                 sound2,
312                 sound2.fill(1))
313
314             self.assertRaises(Exception, lambda : sound1.fill(0))
315             self.assertRaises(Exception, lambda : sound1.fill(-1))
316             self.assertRaises(Exception, lambda : sound1.fill(0.5))
317             self.assertRaises(Exception, lambda : sound1.fill(1.5))
318
319
320     def test_reduce(self):
321             sound1 = self.generator.from_list([0, 0.1] * (self.beat - 1))
322             self.assertEqual(self.beat, len(sound1.reduce(1)))
323
324             sound1 = self.generator.from_list([0.1] * (self.beat - 1))
325             self.assertEqual(len(sound1), len(sound1.reduce(1)))
326
327             sound1 = self.generator.from_list([0, 0.1] * (self.beat - 1))
328             self.assertEqual(len(sound1), len(sound1.reduce(2)))
329
330             self.assertRaises(Exception, lambda : sound1.reduce(0))
331             self.assertRaises(Exception, lambda : sound1.reduce(-1))
332             self.assertRaises(Exception, lambda : sound1.reduce(0.5))
333             self.assertRaises(Exception, lambda : sound1.reduce(1.5))
334
335
336     def test_expand(self):
337             sound1 = self.generator.from_list([0, 0.1, 0.2])
338             self.assertEqual(self.beat, len(sound1.expand(1)))
339             self.assertEqual(self.beat * 2, len(sound1.expand(2)))
340
341             sound1 = self.generator.from_list([0, 0.1] * self.beat)
342             self.assertEqual(len(sound1), len(sound1.expand(1)))
343
344             self.assertRaises(Exception, lambda : sound1.expand(0))
345             self.assertRaises(Exception, lambda : sound1.expand(-1))
346             self.assertRaises(Exception, lambda : sound1.expand(0.5))
```

```
347             self.assertRaises(Exception, lambda : sound1.expand(1.5))
348
349
350     def test_tolist(self):
351             sound1 = self.generator.from_list([0, 0.1, 0.2])
352             self.assertEqual([0, 0.1, 0.2], sound1.tolist())
353
354
355     def test_resample(self):
356             sound1 = self.generator.from_list([0, 0.1, 0.2, 0.3])
357
358             self.assertEqual(
359                 self.generator.from_list([0, 0.1, 0.2, 0.3]),
360                 sound1.resample(4))
361
362             self.assertEqual(1, len(sound1.resample(1)))
363             self.assertEqual(3, len(sound1.resample(3)))
364             self.assertEqual(6, len(sound1.resample(6)))
365             self.assertEqual(10, len(sound1.resample(10)))
366
367             self.assertRaises(Exception, lambda : sound1.expand(0))
368             self.assertRaises(Exception, lambda : sound1.expand(-1))
369             self.assertRaises(Exception, lambda : sound1.expand(0.5))
370             self.assertRaises(Exception, lambda : sound1.expand(1.5))
371
372
373 if __name__ == '__main__':
374     unittest.main()
```

6.   test_parser.py

```python
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 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):
        self.assertEqual(sound, parser.parse(input))



    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 }']
        )

        self.assertParseAllEqual(
            self.generator.from_list([-1]),
```

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

```
115                    ['linear(-0.5, 0.5)', 'lin(-0.5, 0.5)'])
116
117            self.assertParseAllEqual(self.generator.linear(0.5, -0.5),
118                    ['linear(0.5, -0.5)', 'lin(0.5, -0.5)'])
119
120            self.assertParseAllEqual(self.generator.linear(-0.5, -0.5),
121                    ['linear(-0.5, -0.5)', 'lin(-0.5, -0.5)'])
122
123            self.assertParseAllFail(
124                    ['linear', 'linear()', 'lin', 'lin()', 'lin(0)', 'lin(0, 0, 0)'])
125
126            self.assertParseAllFail(
127                    ['lin(2, 0)', 'lin(0, 2)', 'lin(-2, 0)', 'lin(0, -2)',])
128
129        def test_noise(self):
130            sound = parser.parse('noi')
131            self.assertEqual(self.beat, len(sound))
132
133            sound = parser.parse('noi()')
134            self.assertEqual(self.beat, len(sound))
135
136            sound = parser.parse('noise')
137            self.assertEqual(self.beat, len(sound))
138
139            sound = parser.parse('noise()')
140            self.assertEqual(self.beat, len(sound))
141
142            sound = parser.parse('noi(0)')
143            self.assertEqual(self.generator.silence(), sound)
144
145            sound = parser.parse('noi(0.5)')
146            self.assertEqual(self.beat, len(sound))
147            self.assertElementsInRange(sound, -0.5, 0.5)
148
149            self.assertParseAllFail(['noi(2)', 'noi(-2)'])
150
151
152
153
154
155
156
157 if __name__ == '__main__':
158     unittest.main()
```