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)
    minicollider.parser.parse(entrada)


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)
    else:
        prompt()
```

## 2. lexer.py

```
# ------------------------------------------------------------------------------
# lexer.py
#
# Lexer para el mini-collider
# ------------------------------------------------------------------------------
import ply.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_WS   = r'\s|\t|\n'
t_ignore_COMM = r'//.*$'

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])

# Build the lexer
lexer = lex.lex()
```

## 3. mixer.py

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

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

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

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

## 4. parser.py

```
# ------------------------------------------------------------------------------
# parser.py
#
# Parser para el mini-collider
# ------------------------------------------------------------------------------
import ply.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'),
    ('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]


def p_BUFFER_metodo_0param(t):
    '''BUFFER :      BUFFER PLAY ONEPARAM
```

```
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
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([]))
        self.assertRaises(Exception, lambda : self.generator.from_list([2]))
        self.assertRaises(Exception, lambda : self.generator.from_list([-2]))


    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())

```

```
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])
```

```
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

```
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()
```