Advanced Software Engineering Documentation https://github.com/alexeifigueroa/ASEPetSynthesizer

Alexei Figueroa

February 2018

Contents

1. Introduction

This documentation highlights the implementation of the techniques provided in the Advanced Software Engineering lecture. The aplication developed consists of a simple piano developed using the `Python` language with the help of the libraries `pygame` and `scipy`. The main paradigm behind the architecture of the application is the Model View Presenter, relaxations are made to cover as much as possible the requirements of the assignment. The mechanism of generation of the sound is based on the model of an analog synthesizer. Several simplifications are made to limit the complexity of the software. The synthesizer accepts only keyboard input, it requires an audio device and it has been tested on ubuntu 17.10 and Windows 10 and minimally in the trusty environment of TravisCI.

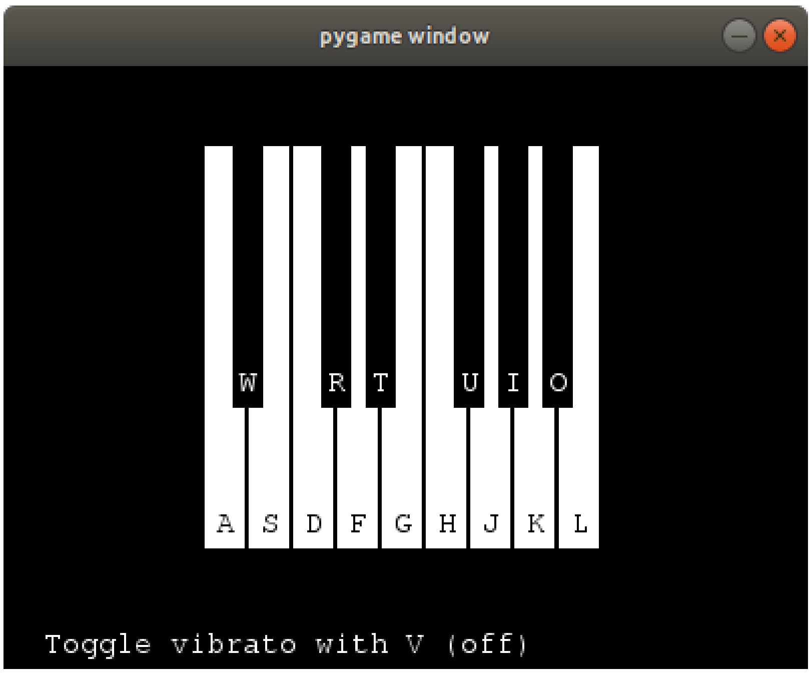


Figure 1.1: GUI of the synthesizer

2. UML

2.1 Class Diagram

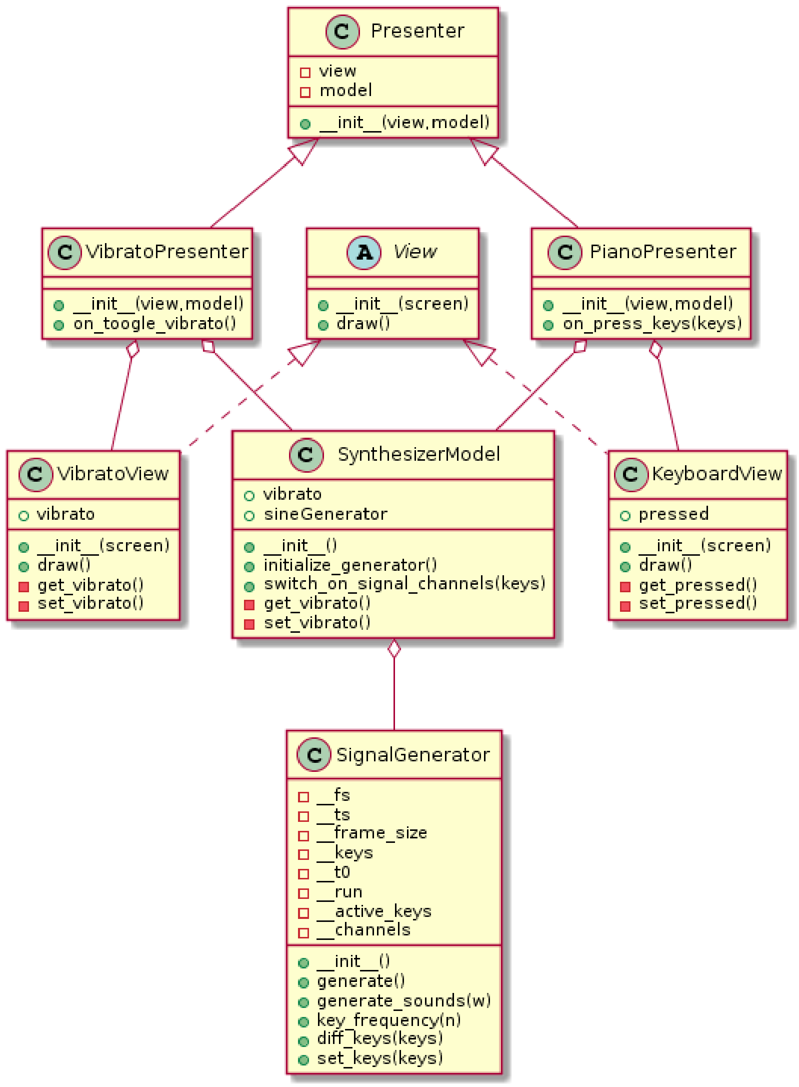


Figure 2.1: Class Diagram of the synthesizer

2.2 Sequence Diagram

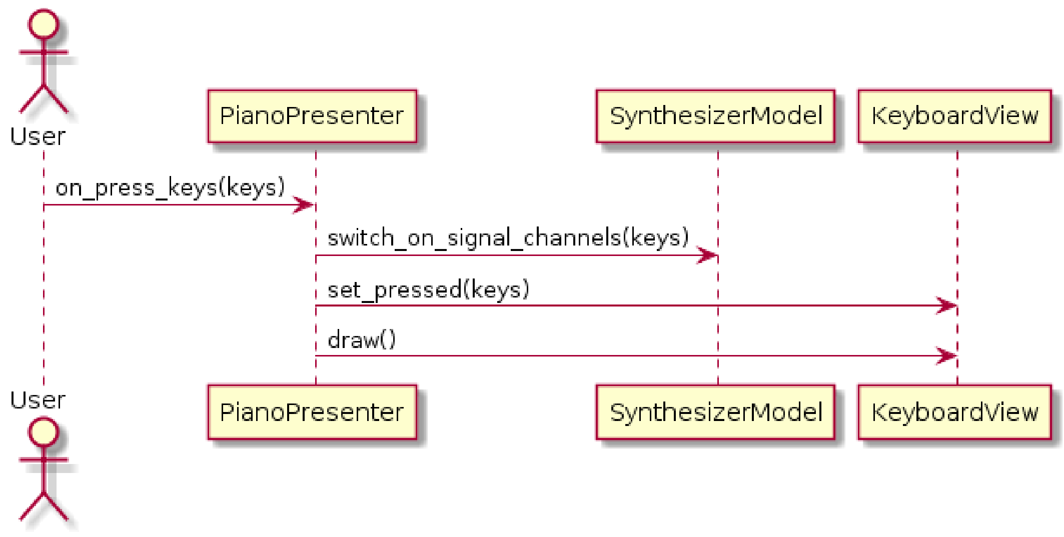


Figure 2.2: Sequence Diagram of the synthesizer describing the MVP calls given a user key event.

2.3 State Diagram

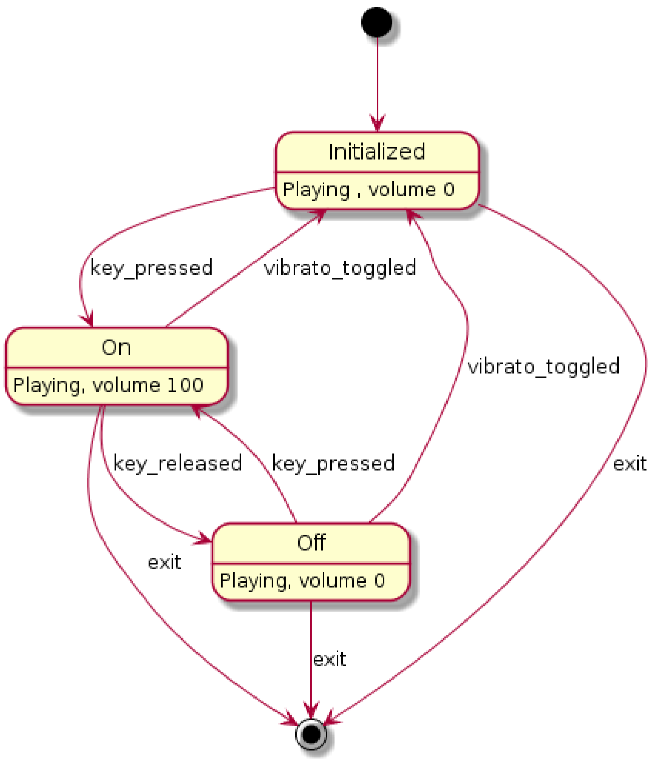


Figure 2.3: State Diagram of a mixer channel

2.4 Activity Diagram

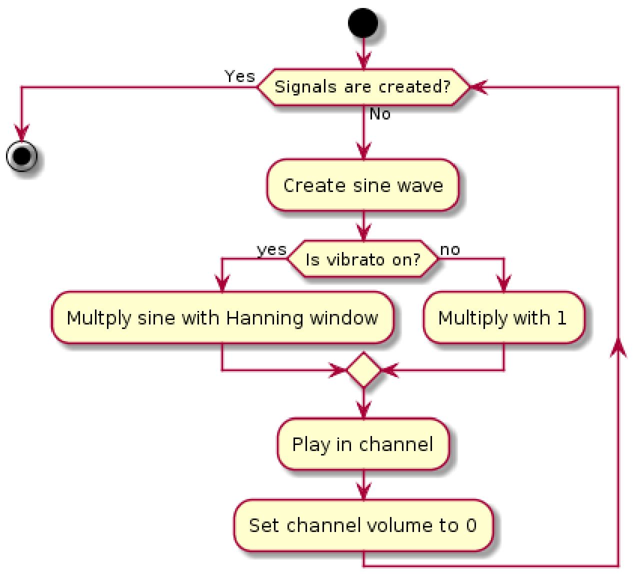


Figure 2.4: Activity Diagram describing the sound generation for all channels.

2.5 Object Diagram

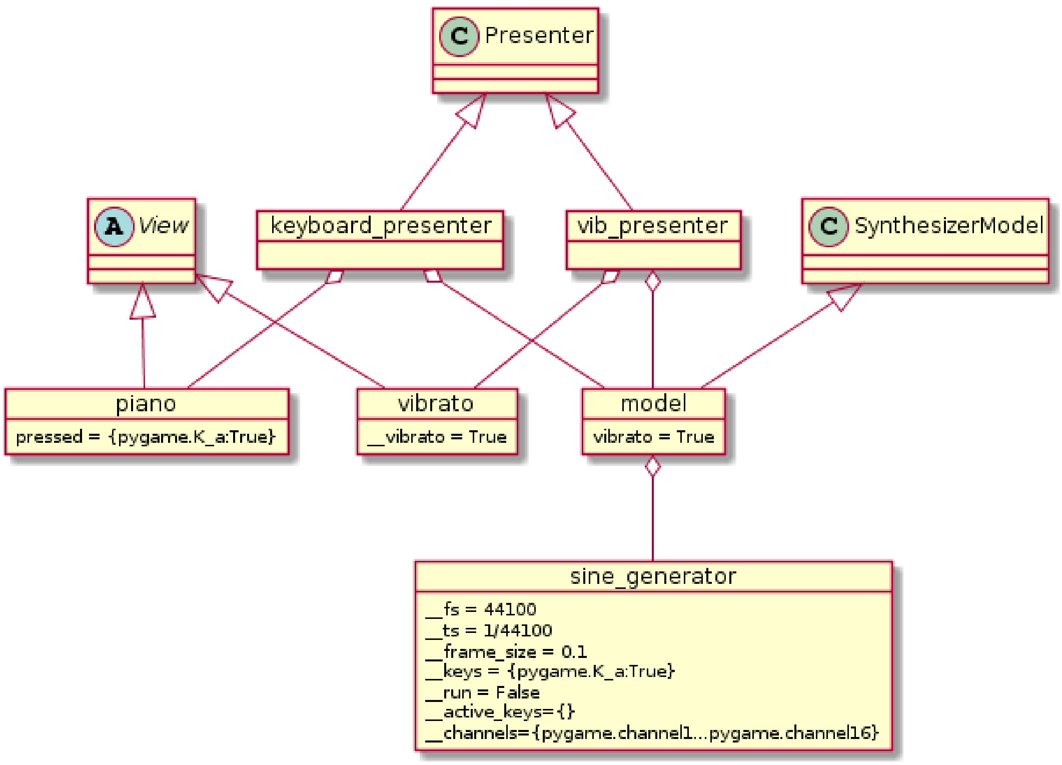
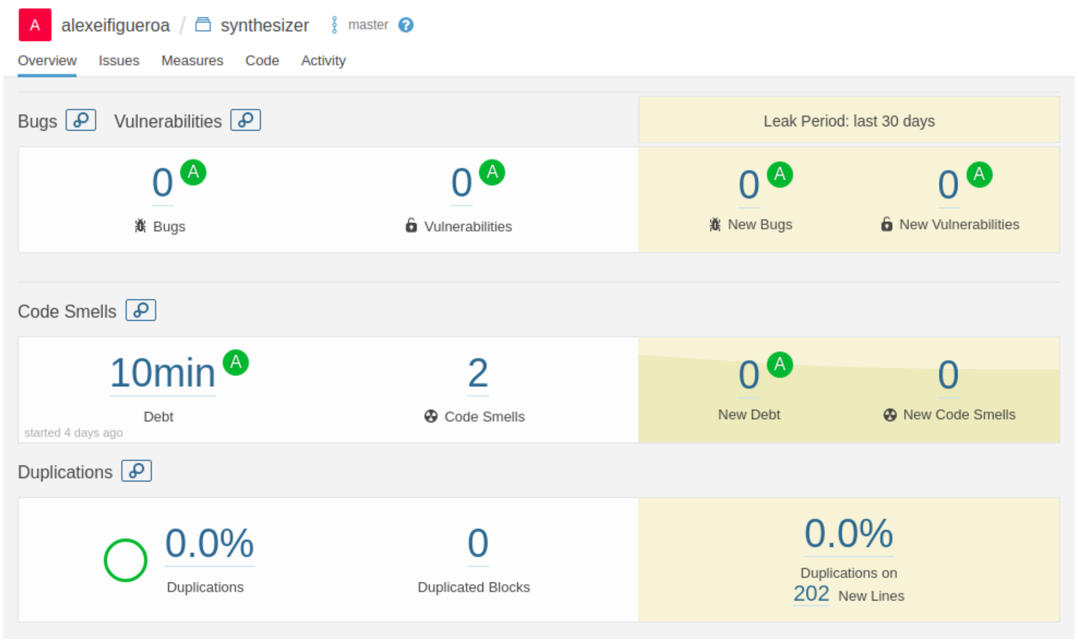


Figure 2.5: Object Diagram of key "A" pressed with the vibrato turned on, additionally the Views, Presenters and Model are indicated.

3. Metrics

The SonarCloud platform was chosen to run the code analysis on the code, The project metrics can be accessed through `https://sonarcloud.io/organizations/alexeifigueroa-github/projects`


alexeifigueroa / synthesizer master

Overview Issues Measures Code Activity

Bugs Vulnerabilities

Leak Period: last 30 days

0 A Bugs

0 A Vulnerabilities

0 A New Bugs

0 A New Vulnerabilities

Code Smells

10min A Debt

started 4 days ago

2 Code Smells

0 A New Debt

0 New Code Smells

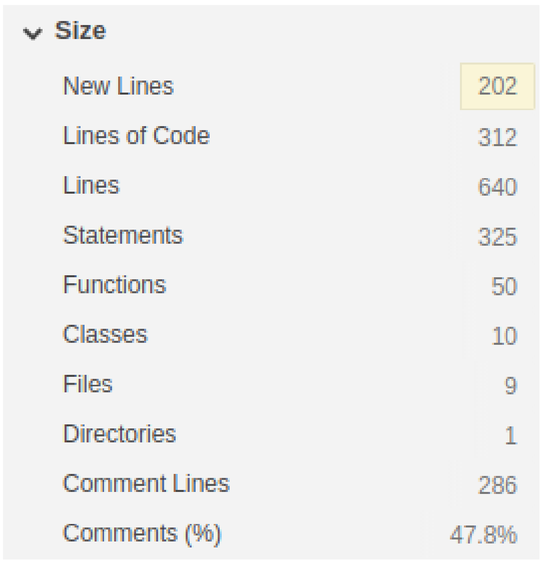
Duplications

0.0% Duplications

0 Duplicated Blocks

0.0% Duplications on 202 New Lines

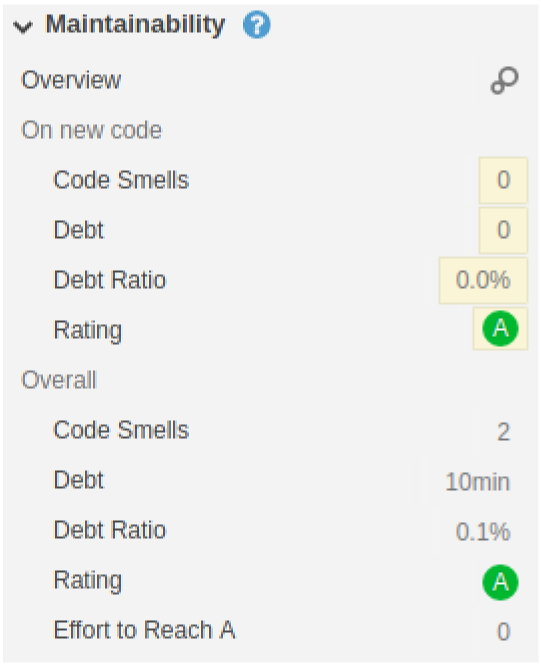
Figure 3.1: Dashboard of SonarCloud of the project



Size

New Lines	202
Lines of Code	312
Lines	640
Statements	325
Functions	50
Classes	10
Files	9
Directories	1
Comment Lines	286
Comments (%)	47.8%

(a) Size measures

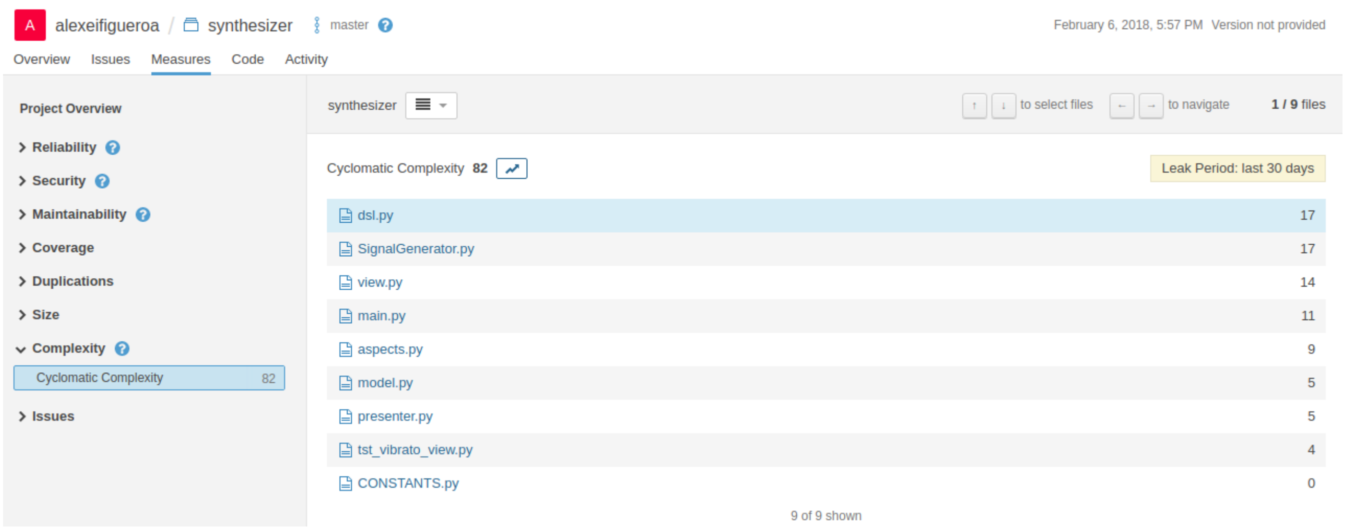


Maintainability

Overview

On new code	
Code Smells	0
Debt	0
Debt Ratio	0.0%
Rating	A
Overall	
Code Smells	2
Debt	10min
Debt Ratio	0.1%
Rating	A
Effort to Reach A	0

(b) Maintainability measures.



A alexeifigueroa / synthesizer master

February 6, 2018, 5:57 PM Version not provided

Project Overview

- Reliability
- Security
- Maintainability
- Coverage
- Duplications
- Size
- Complexity
  - Cyclomatic Complexity 82
- Issues

synthesizer

1 / 9 files

Cyclomatic Complexity 82

Leak Period: last 30 days

dsl.py	17
SignalGenerator.py	17
view.py	14
main.py	11
aspects.py	9
model.py	5
presenter.py	5
tst_vibrato_view.py	4
CONSTANTS.py	0

9 of 9 shown

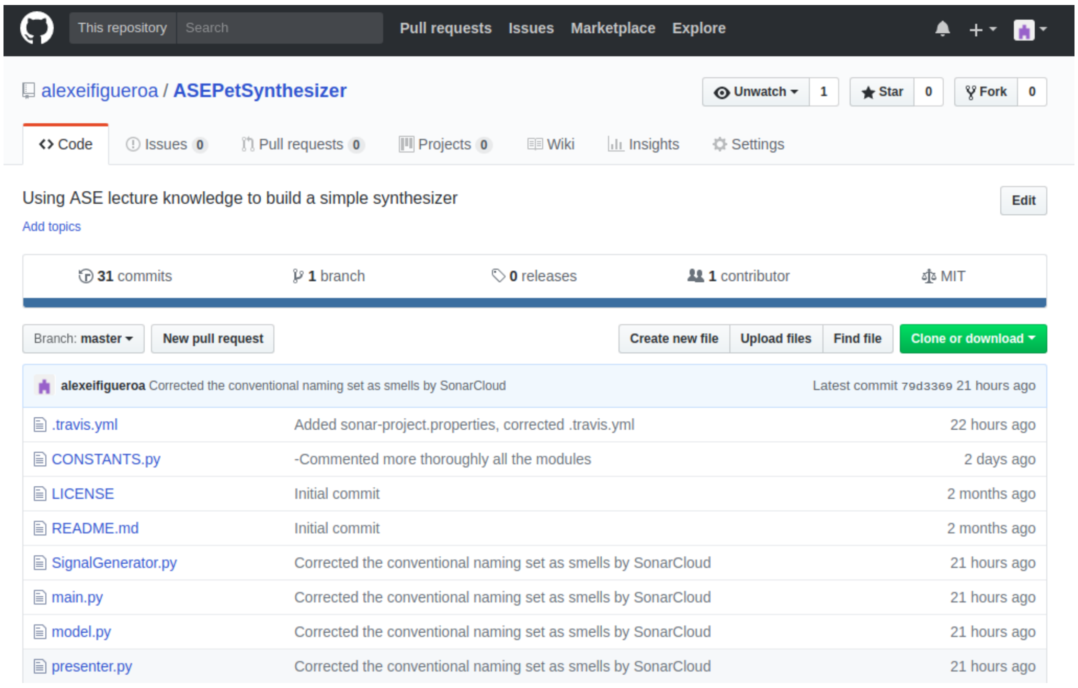
(c) Cyclomatic complexity.

Figure 3.2: Metrics of SonarCloud for the synthesizer project.

4. Clean Code Development

4.1 Version Control

Since it's inception this project was developed using git as the version control system with the repository being hosted on github.



alexeifigueroa / ASEPetSynthesizer

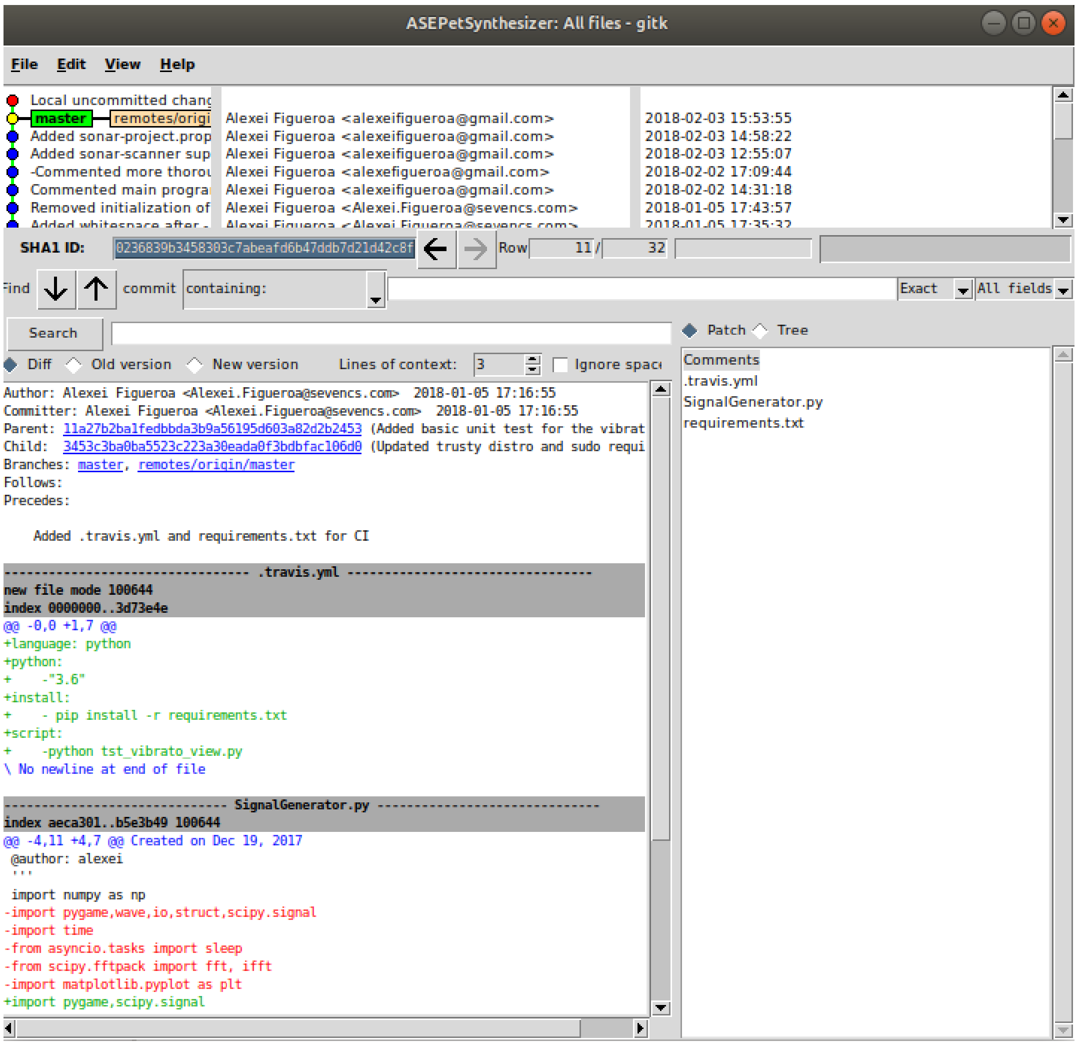
Using ASE lecture knowledge to build a simple synthesizer

31 commits 1 branch 0 releases 1 contributor MIT

alexeifigueroa Corrected the conventional naming set as smells by SonarCloud Latest commit 79d3369 21 hours ago

.travis.yml	Added sonar-project.properties, corrected .travis.yml	22 hours ago
CONSTANTS.py	-Commented more thoroughly all the modules	2 days ago
LICENSE	Initial commit	2 months ago
README.md	Initial commit	2 months ago
SignalGenerator.py	Corrected the conventional naming set as smells by SonarCloud	21 hours ago
main.py	Corrected the conventional naming set as smells by SonarCloud	21 hours ago
model.py	Corrected the conventional naming set as smells by SonarCloud	21 hours ago
presenter.py	Corrected the conventional naming set as smells by SonarCloud	21 hours ago

Figure 4.1: Github repository main page



```
Local uncommitted chang
master  remotes/origi        Alexei Figueroa <alexeifigueroa@gmail.com>      2018-02-03 15:53:55
Added sonar-project.prop     Alexei Figueroa <alexeifigueroa@gmail.com>      2018-02-03 14:58:22
Added sonar-scanner sup      Alexei Figueroa <alexeifigueroa@gmail.com>      2018-02-03 12:55:07
-Commented more thorou       Alexei Figueroa <alexefigueroa@gmail.com>       2018-02-02 17:09:44
Commented main progra        Alexei Figueroa <alexeifigueroa@gmail.com>      2018-02-02 14:31:18
Removed initialization of    Alexei Figueroa <Alexei.Figueroa@sevencs.com>   2018-01-05 17:43:57
Added whitespace after -     Alexei Figueroa <Alexei.Figueroa@sevencs.com>   2018-01-05 17:35:32

SHA1 ID: 0236839b3458303c7abeafd6b47ddb7d21d42c8f    Row 11 / 32
```

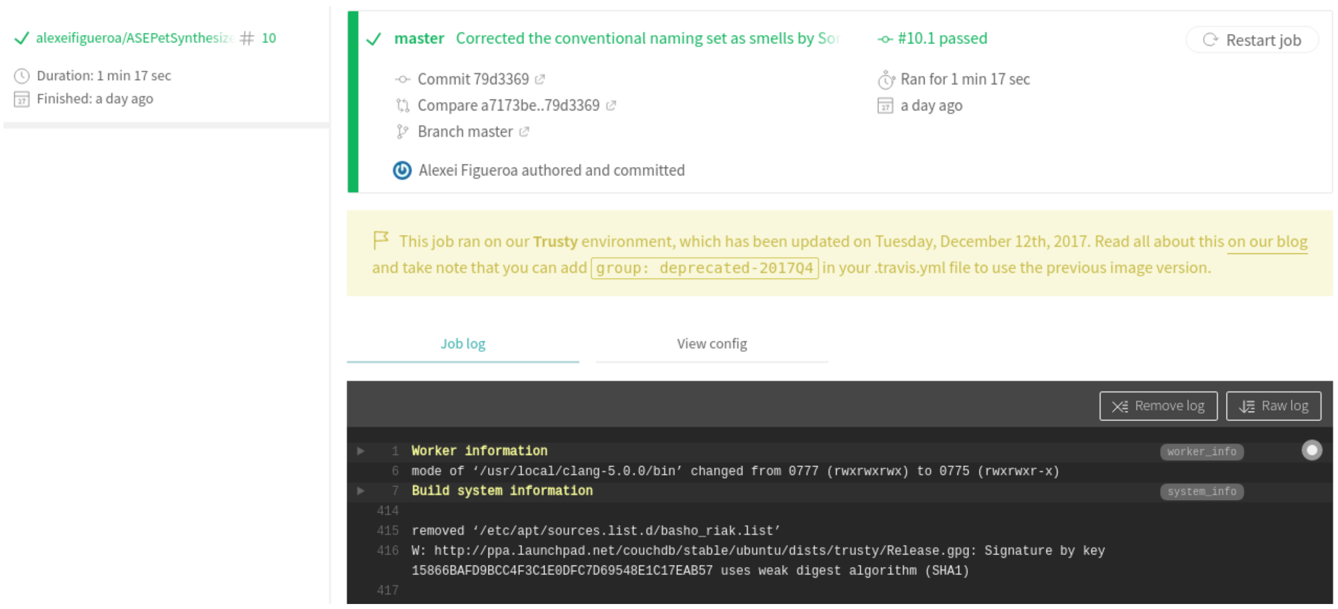
```
Author: Alexei Figueroa <Alexei.Figueroa@sevencs.com>  2018-01-05 17:16:55
Committer: Alexei Figueroa <Alexei.Figueroa@sevencs.com>  2018-01-05 17:16:55
Parent: 11a27b2ba1fedbbda3b9a56195d603a82d2b2453 (Added basic unit test for the vibrat
Child:  3453c3ba0ba5523c223a30eada0f3bdbfac106d0 (Updated trusty distro and sudo requi
Branches: master, remotes/origin/master
Follows:
Precedes:

    Added .travis.yml and requirements.txt for CI

-------------------------------- .travis.yml --------------------------------
new file mode 100644
index 0000000..3d73e4e
@@ -0,0 +1,7 @@
+language: python
+python:
+    -"3.6"
+install:
+    - pip install -r requirements.txt
+script:
+    -python tst_vibrato_view.py
\ No newline at end of file

----------------------------- SignalGenerator.py -----------------------------
index aeca301..b5e3b49 100644
@@ -4,11 +4,7 @@ Created on Dec 19, 2017
 @author: alexei
 '''
 import numpy as np
-import pygame,wave,io,struct,scipy.signal
-import time
-from asyncio.tasks import sleep
-from scipy.fftpack import fft, ifft
-import matplotlib.pyplot as plt
+import pygame,scipy.signal
```

```
Comments
.travis.yml
SignalGenerator.py
requirements.txt
```

Figure 4.2: Gitk frontend on development machine

4.2 Continuous Integration

TravisCI was the platform of choice for orchestrating the building,testing and triggering the code analysis by SonarCloud. The .travis.yml file is under the root of the repository, after providing TravisCI access to the Github account, as soon as a git push is through the whole process is run.


✓ alexeifigueroa/ASEPetSynthesiz # 10

Duration: 1 min 17 sec

Finished: a day ago

✓ master Corrected the conventional naming set as smells by So

#10.1 passed

Restart job

Commit 79d3369

Compare a7173be..79d3369

Branch master

Alexei Figueroa authored and committed

Ran for 1 min 17 sec

a day ago

This job ran on our **Trusty** environment, which has been updated on Tuesday, December 12th, 2017. Read all about this on our blog and take note that you can add `group: deprecated-2017Q4` in your .travis.yml file to use the previous image version.

Job log

View config

Remove log

Raw log

```
Worker information                                                                     worker_info
mode of '/usr/local/clang-5.0.0/bin' changed from 0777 (rwxrwxrwx) to 0775 (rwxrwxr-x)
Build system information                                                               system_info

removed '/etc/apt/sources.list.d/basho_riak.list'
W: http://ppa.launchpad.net/couchdb/stable/ubuntu/dists/trusty/Release.gpg: Signature by key
15866BAFD9BCC4F3C1E0DFC7D69548E1C17EAB57 uses weak digest algorithm (SHA1)
```

Figure 4.3: TravisCI, build log after a succesful run.

4.3 Architecture and class design, separation of concerns[1]

The purpose of the program presented in this work is to simulate an analogue synthesizer. Such hardware is normally conceived as a modular device by itself, composed out of smaller very domain specific tools. The intricate domain logic makes a software simulation naturally exploit the abstraction and isolation of concerns. A fair methodology to achieve this is the Model View Presenter architecture which splits the domain logic into a **Model**, the user interactive functionality into a **View** and the presentational logic which integrates it all in a **Presenter**. In this work the **Model** logic is implemented into a single class which aggregates a SignalGenerator. Further domain specific hardware that was not implemented in this project, like for instance a voltage controlled filter VCO could be integrated here. Additionally the Views representing a piano Keyboard and a vibrato switch were implemented. Finally the Presenter module synchronizes the interaction of the user by triggering the event handlers and modifying the **Model** and the **View** accordingly.

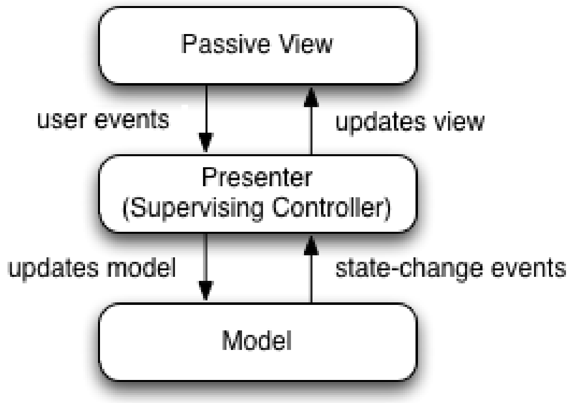
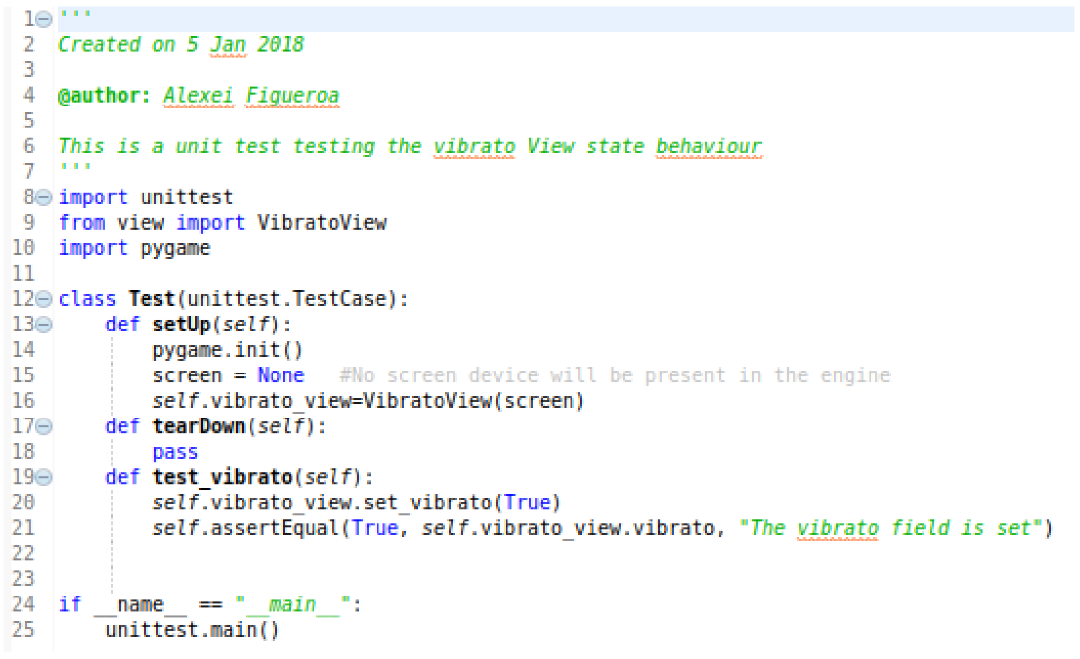


Figure 4.4: Diagram representing the MVP design pattern[2].

4.4 Automated Unit Tests

The choice in architecture of the program presented is well suited for unit testing. The library chosen for such tests is the default standard unittest library shipping with the Python interpreter. A model test for the View of the vibrato control is shown next. These unit test is executed at the end of the build process by the CI environment.

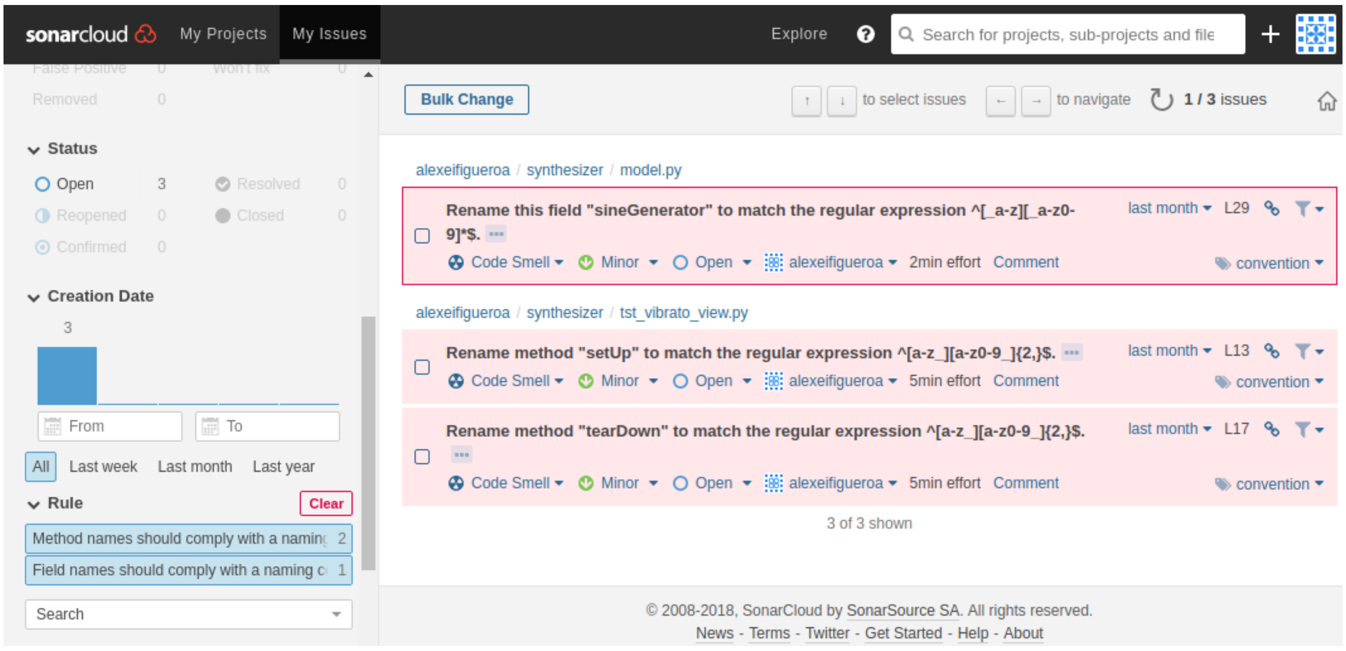


```python
'''
Created on 5 Jan 2018

@author: Alexei Figueroa

This is a unit test testing the vibrato View state behaviour
'''
import unittest
from view import VibratoView
import pygame

class Test(unittest.TestCase):
    def setUp(self):
        pygame.init()
        screen = None   #No screen device will be present in the engine
        self.vibrato_view=VibratoView(screen)
    def tearDown(self):
        pass
    def test_vibrato(self):
        self.vibrato_view.set_vibrato(True)
        self.assertEqual(True, self.vibrato_view.vibrato, "The vibrato field is set")


if __name__ == "__main__":
    unittest.main()
```

Figure 4.5: Snipet of tst_vibrato_view.py

4.5 Source Code conventions

The usage of source code conventions was checked mainly with the assistance of SonarCloud, a good example are the naming conventions. The following figure highlights 3 different examples, it's important to know that for the 2 last ones, setUp and tearDown methods, these names are part of a mandatory interface specification of the unittest Python standard library.



sonarcloud My Projects My Issues Explore Search for projects, sub-projects and files

False Positive 0 Won't fix 0
Removed 0

Status
Open 3 Resolved 0
Reopened 0 Closed 0
Confirmed 0

Creation Date
3
From To
All Last week Last month Last year

Rule Clear
Method names should comply with a naming 2
Field names should comply with a naming c 1
Search

Bulk Change ↑ ↓ to select issues ← → to navigate 1 / 3 issues

alexeifigueroa / synthesizer / model.py

Rename this field "sineGenerator" to match the regular expression ^[_a-z][_a-z0-9]*$. ··· last month L29
Code Smell Minor Open alexeifigueroa 2min effort Comment convention

alexeifigueroa / synthesizer / tst_vibrato_view.py

Rename method "setUp" to match the regular expression ^[a-z_][a-z0-9_]{2,}$. ··· last month L13
Code Smell Minor Open alexeifigueroa 5min effort Comment convention

Rename method "tearDown" to match the regular expression ^[a-z_][a-z0-9_]{2,}$. ··· last month L17
Code Smell Minor Open alexeifigueroa 5min effort Comment convention

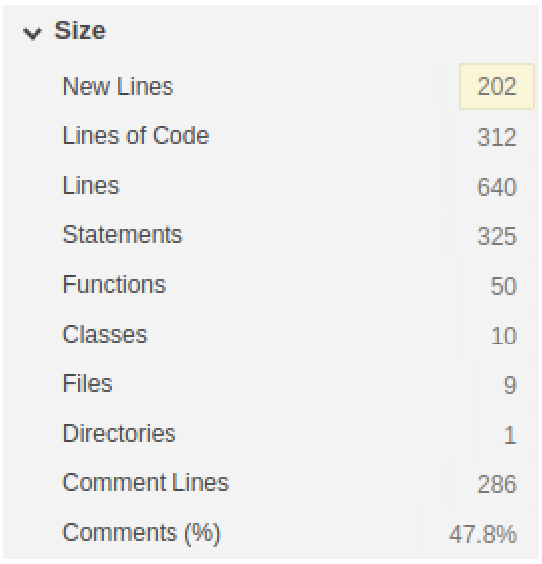
3 of 3 shown

© 2008-2018, SonarCloud by SonarSource SA. All rights reserved.
News - Terms - Twitter - Get Started - Help - About

Figure 4.6: Sonar Cloud "smells" pointing at naming convention failures

4.6 Static Code Analysis

As mentioned before the SonarCloud platform was used for the code analysis. Some of the size metrics are depicted next.



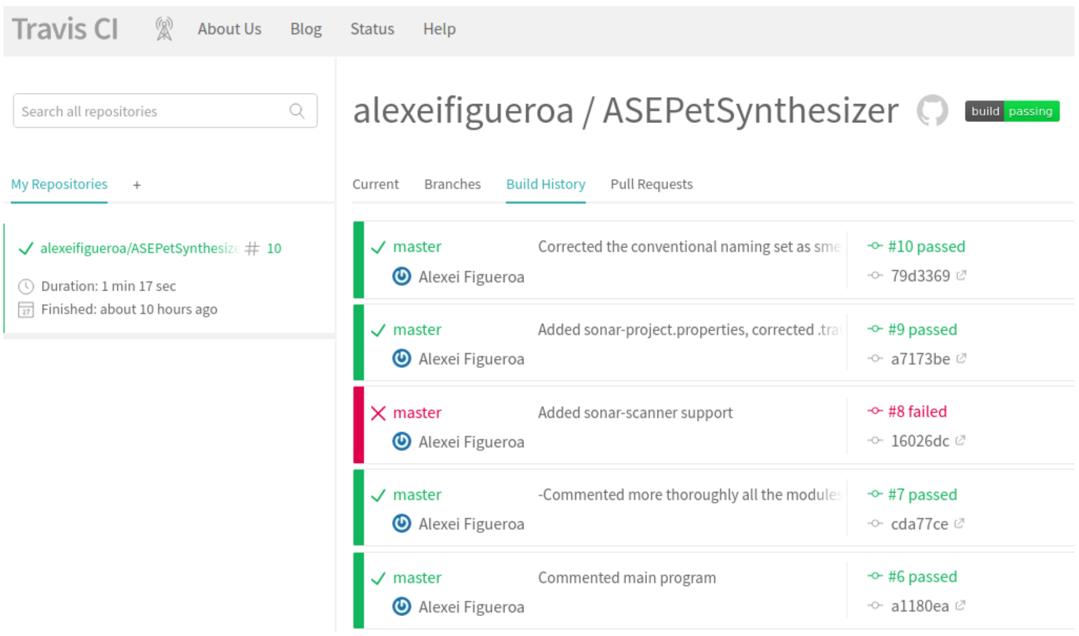
Size

New Lines	202
Lines of Code	312
Lines	640
Statements	325
Functions	50
Classes	10
Files	9
Directories	1
Comment Lines	286
Comments (%)	47.8%

Figure 4.7: Sonar Cloud Size metrics for the project.

5. Continuous Delivery

As mentioned before, the system for Continuous Integration chosen was TravisCI. The build history and contents of the .travis.yml file can be seen next.

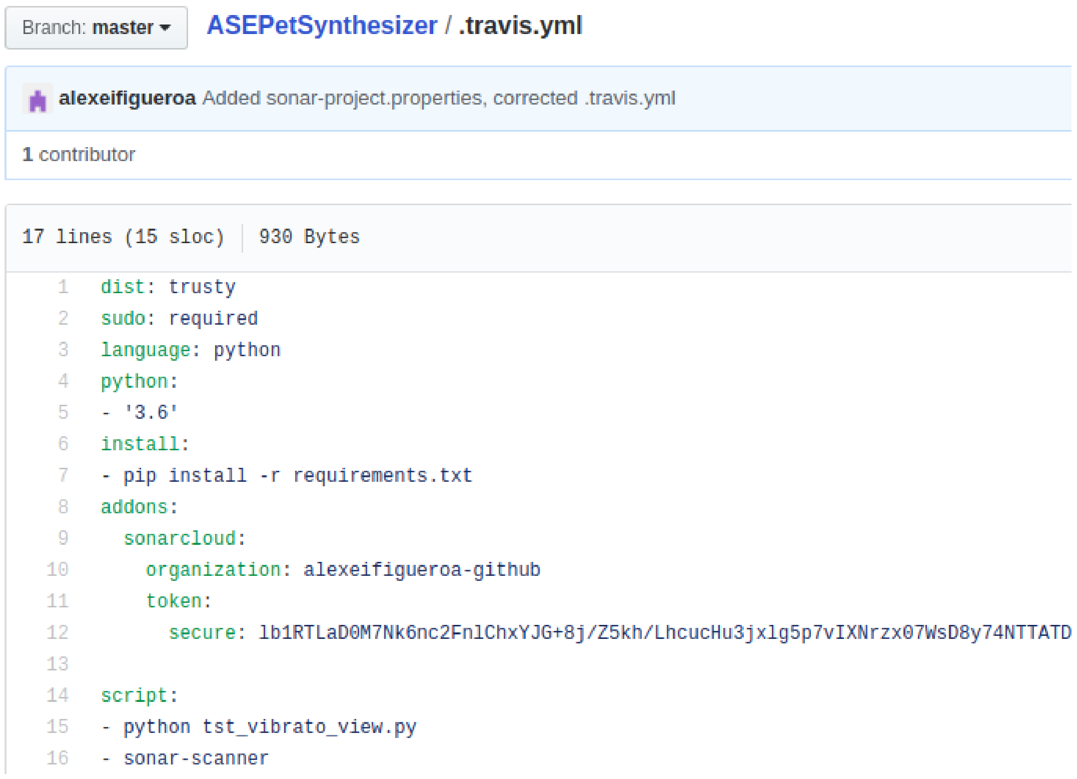


alexeifigueroa / ASEPetSynthesizer

✓ alexeifigueroa/ASEPetSynthesiz # 10
Duration: 1 min 17 sec
Finished: about 10 hours ago

✓ master	Alexei Figueroa	Corrected the conventional naming set as sme	#10 passed, 79d3369
✓ master	Alexei Figueroa	Added sonar-project.properties, corrected .tra	#9 passed, a7173be
✗ master	Alexei Figueroa	Added sonar-scanner support	#8 failed, 16026dc
✓ master	Alexei Figueroa	-Commented more thoroughly all the module	#7 passed, cda77ce
✓ master	Alexei Figueroa	Commented main program	#6 passed, a1180ea

Figure 5.1: Build history in Travis-CI



Branch: **master** ▾ **ASEPetSynthesizer / .travis.yml**

alexeifigueroa Added sonar-project.properties, corrected .travis.yml

1 contributor

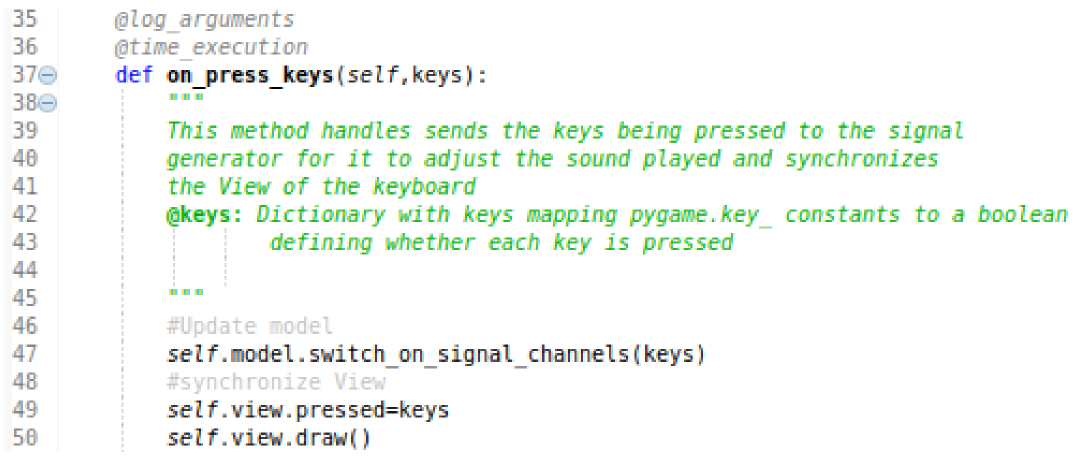
17 lines (15 sloc) | 930 Bytes

```yaml
dist: trusty
sudo: required
language: python
python:
- '3.6'
install:
- pip install -r requirements.txt
addons:
  sonarcloud:
    organization: alexeifigueroa-github
    token:
      secure: lb1RTLaD0M7Nk6nc2FnlChxYJG+8j/Z5kh/LhcucHu3jxlg5p7vIXNrzx07WsD8y74NTTATD

script:
- python tst_vibrato_view.py
- sonar-scanner
```

Figure 5.2: Contents of .travis.yml

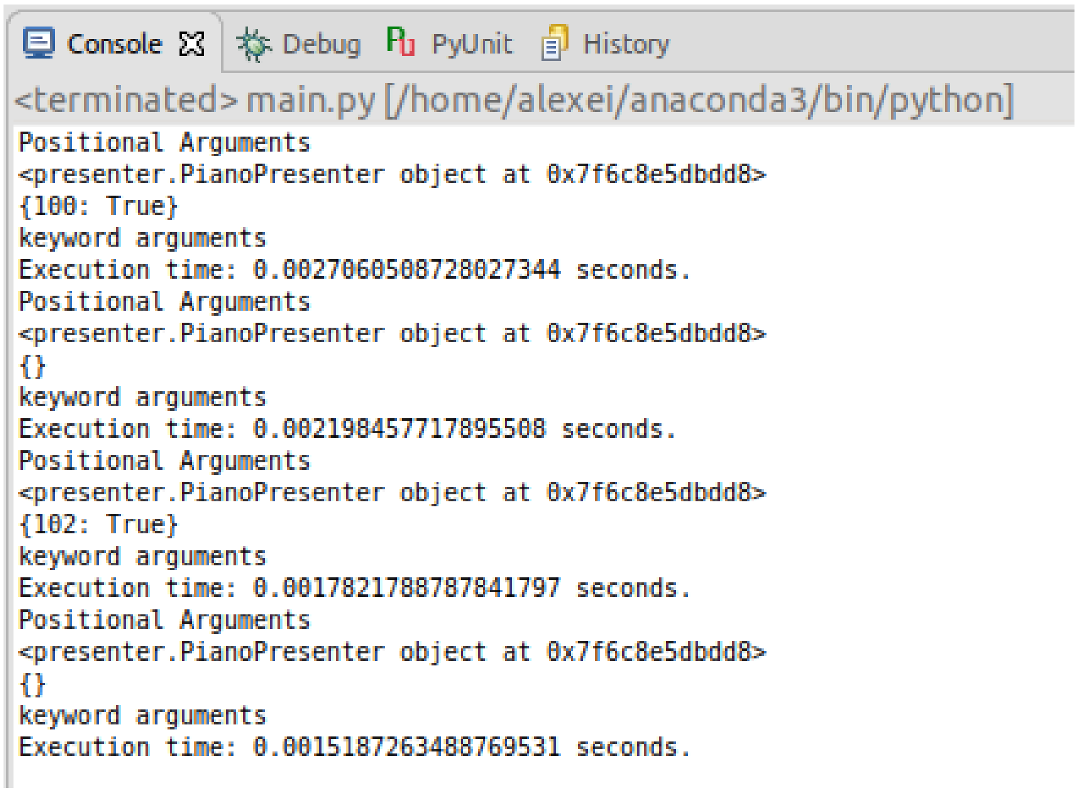
6. AOP

Aspect Oriented Programming in the context of this work is applied by the means of python decorators, these are high order functions that wrap their arguments with code placed before and after the function of interest. This function can be optionally executed. For the purposes of the synthesizer simulation a critical operation is the responsiveness of the program to react to the user pressing a key. Following this ideas the event handler for any key press is logged to expose the keys that are pressed (in case of multiplicity), and timed. This is implemented in the PianoPresenter class with non invasive code by the means of the `@decorator` notation. The implementation of both wrappers is under the aspects module.



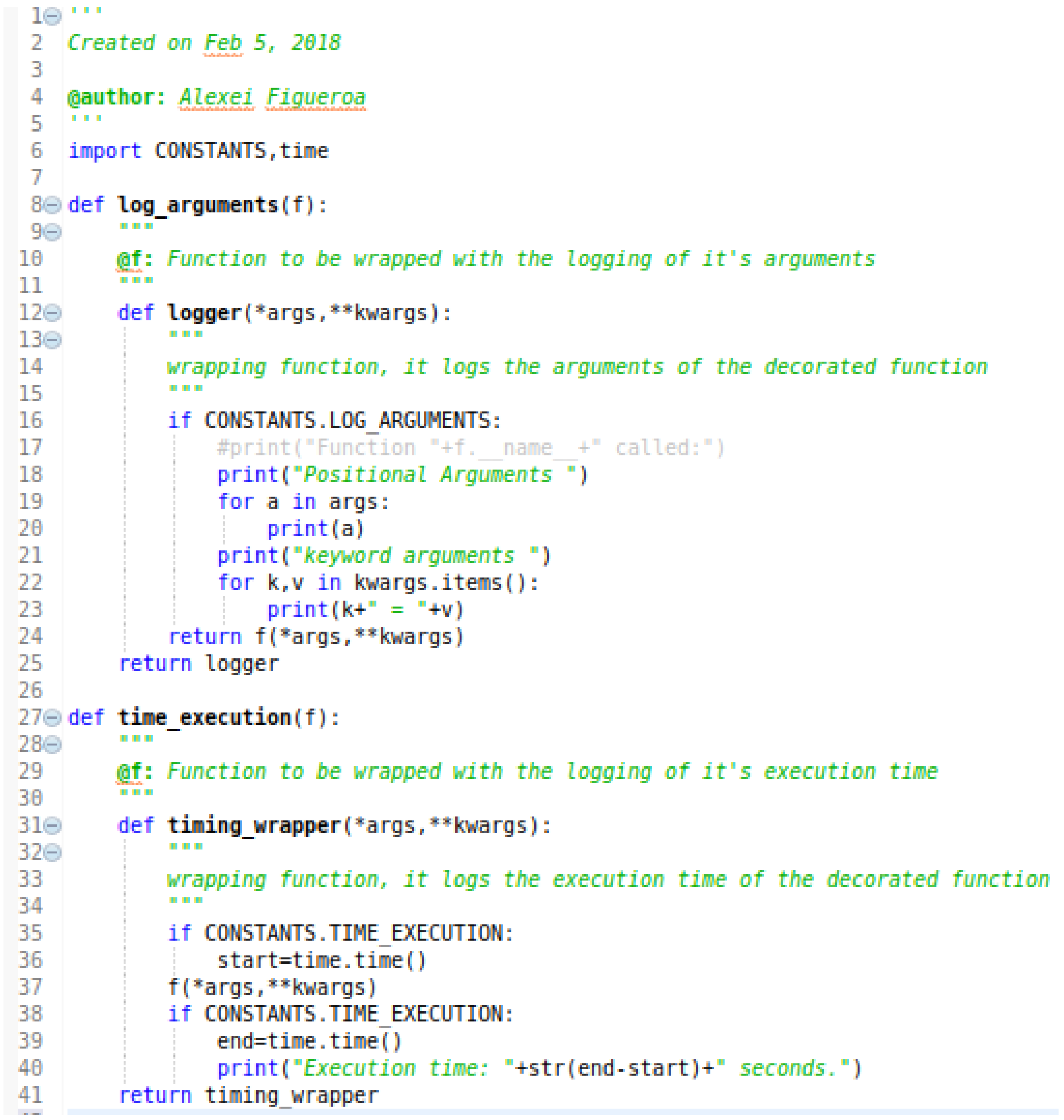
```python
    @log_arguments
    @time_execution
    def on_press_keys(self,keys):
        """
        This method handles sends the keys being pressed to the signal
        generator for it to adjust the sound played and synchronizes
        the View of the keyboard
        @keys: Dictionary with keys mapping pygame.key_ constants to a boolean
               defining whether each key is pressed

        """
        #Update model
        self.model.switch_on_signal_channels(keys)
        #synchronize View
        self.view.pressed=keys
        self.view.draw()
```

Figure 6.1: Decorators placed around the key pressing event handler.



```
<terminated> main.py [/home/alexei/anaconda3/bin/python]
Positional Arguments
<presenter.PianoPresenter object at 0x7f6c8e5dbdd8>
{100: True}
keyword arguments
Execution time: 0.0027060508728027344 seconds.
Positional Arguments
<presenter.PianoPresenter object at 0x7f6c8e5dbdd8>
{}
keyword arguments
Execution time: 0.002198457717895508 seconds.
Positional Arguments
<presenter.PianoPresenter object at 0x7f6c8e5dbdd8>
{102: True}
keyword arguments
Execution time: 0.0017821788787841797 seconds.
Positional Arguments
<presenter.PianoPresenter object at 0x7f6c8e5dbdd8>
{}
keyword arguments
Execution time: 0.0015187263488769531 seconds.
```

Figure 6.2: Logging of keys and execution times at runtime for both key pressing and releasing of two distinct keys.

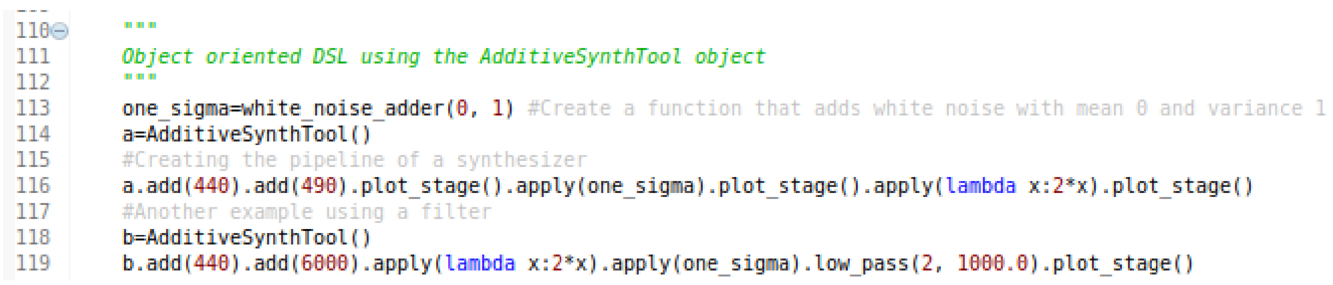


```python
'''
Created on Feb 5, 2018

@author: Alexei Figueroa
'''
import CONSTANTS,time

def log_arguments(f):
    """
    @f: Function to be wrapped with the logging of it's arguments
    """
    def logger(*args,**kwargs):
        """
        wrapping function, it logs the arguments of the decorated function
        """
        if CONSTANTS.LOG_ARGUMENTS:
            #print("Function "+f.__name__+" called:")
            print("Positional Arguments ")
            for a in args:
                print(a)
            print("keyword arguments ")
            for k,v in kwargs.items():
                print(k+" = "+v)
        return f(*args,**kwargs)
    return logger

def time_execution(f):
    """
    @f: Function to be wrapped with the logging of it's execution time
    """
    def timing_wrapper(*args,**kwargs):
        """
        wrapping function, it logs the execution time of the decorated function
        """
        if CONSTANTS.TIME_EXECUTION:
            start=time.time()
        f(*args,**kwargs)
        if CONSTANTS.TIME_EXECUTION:
            end=time.time()
            print("Execution time: "+str(end-start)+" seconds.")
    return timing_wrapper
```

Figure 6.3: Higher order functions for execution timing and logging arguments in the aspects.py module

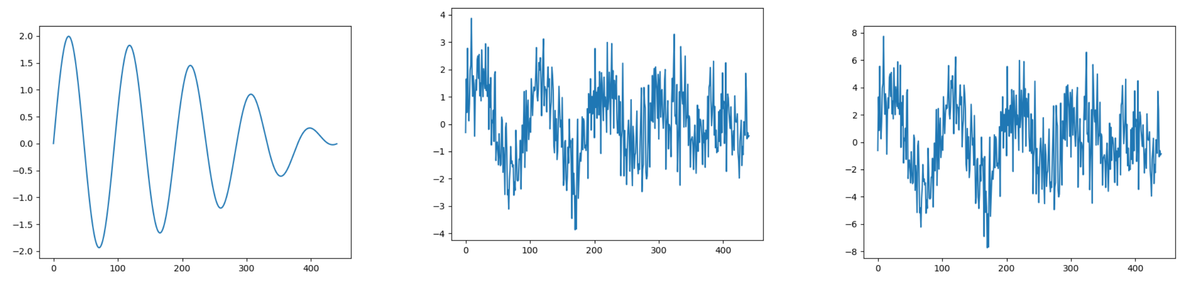
7. DSL

An analogue synthesizer comprehends a signal processing pipeline. Signals from a signal generator are added, and subsequently transformed to an output that is converted to audio. The DSL proposed with this project is for purposes of describing the pipeline of audio synthesizer. Deliberately 2 different approaches where implemented to describe this tipe of pipeline: an object oriented class implementing chaining of the transformation of it's sole attribute (a signal), and a functional approach describing a composition of functions. These are located under the `dsl` module and demo applications are presented next.



```
110    """
111    Object oriented DSL using the AdditiveSynthTool object
112    """
113    one_sigma=white_noise_adder(0, 1) #Create a function that adds white noise with mean 0 and variance 1
114    a=AdditiveSynthTool()
115    #Creating the pipeline of a synthesizer
116    a.add(440).add(490).plot_stage().apply(one_sigma).plot_stage().apply(lambda x:2*x).plot_stage()
117    #Another example using a filter
118    b=AdditiveSynthTool()
119    b.add(440).add(6000).apply(lambda x:2*x).apply(one_sigma).low_pass(2, 1000.0).plot_stage()
```

Figure 7.1: Object oriented functionality of building a signal transformation pipeline.



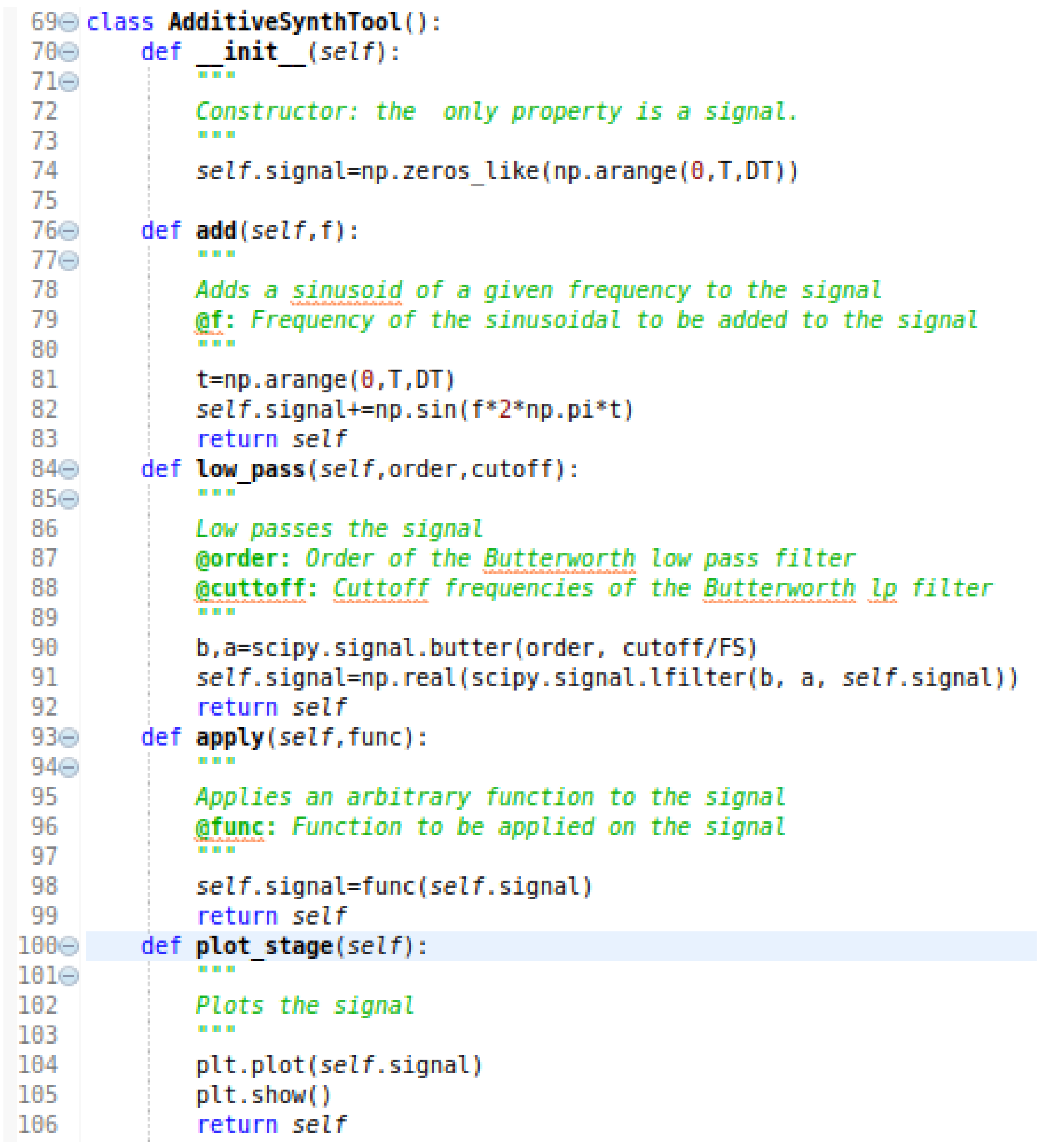
(a) First plot_stage call, only 2 sine waves in the pipeline

(b) Second plot_stage call, white noise is added to the pipeline

(c) Third plot_stage call,the signal is doubled

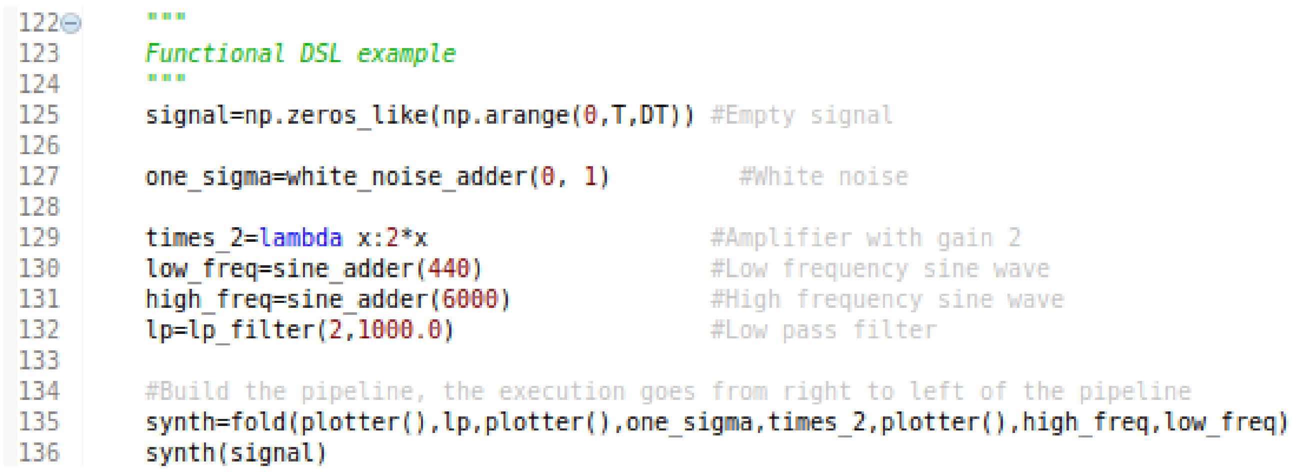
Figure 7.2: Execution of line 116 plotting the signal at different stages of the transformation pipeline

The class that makes the chaining of the pipeline possible is presented next, followed by the functions and Closures for the functional programming approach.



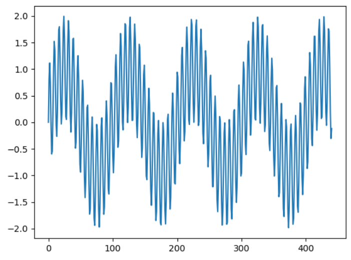
```python
class AdditiveSynthTool():
    def __init__(self):
        """
        Constructor: the  only property is a signal.
        """
        self.signal=np.zeros_like(np.arange(0,T,DT))

    def add(self,f):
        """
        Adds a sinusoid of a given frequency to the signal
        @f: Frequency of the sinusoidal to be added to the signal
        """
        t=np.arange(0,T,DT)
        self.signal+=np.sin(f*2*np.pi*t)
        return self
    def low_pass(self,order,cutoff):
        """
        Low passes the signal
        @order: Order of the Butterworth low pass filter
        @cuttoff: Cuttoff frequencies of the Butterworth lp filter
        """
        b,a=scipy.signal.butter(order, cutoff/FS)
        self.signal=np.real(scipy.signal.lfilter(b, a, self.signal))
        return self
    def apply(self,func):
        """
        Applies an arbitrary function to the signal
        @func: Function to be applied on the signal
        """
        self.signal=func(self.signal)
        return self
    def plot_stage(self):
        """
        Plots the signal
        """
        plt.plot(self.signal)
        plt.show()
        return self
```

Figure 7.3: Class to generate signals by providing methods that can be chained in the object oriented DSL.

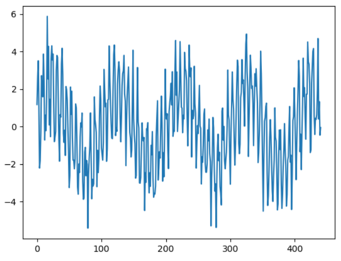


```python
    """
    Functional DSL example
    """
    signal=np.zeros_like(np.arange(0,T,DT)) #Empty signal

    one_sigma=white_noise_adder(0, 1)         #White noise

    times_2=lambda x:2*x                    #Amplifier with gain 2
    low_freq=sine_adder(440)                #Low frequency sine wave
    high_freq=sine_adder(6000)              #High frequency sine wave
    lp=lp_filter(2,1000.0)                  #Low pass filter

    #Build the pipeline, the execution goes from right to left of the pipeline
    synth=fold(plotter(),lp,plotter(),one_sigma,times_2,plotter(),high_freq,low_freq)
    synth(signal)
```

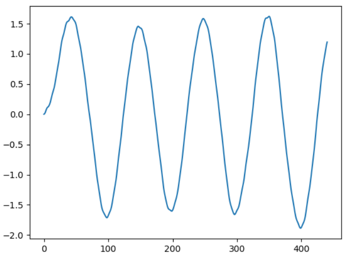
Figure 7.4: Functional approach of building a signal transformation pipeline.



(a) First plotter() closure , only 2 sine waves in the pipeline

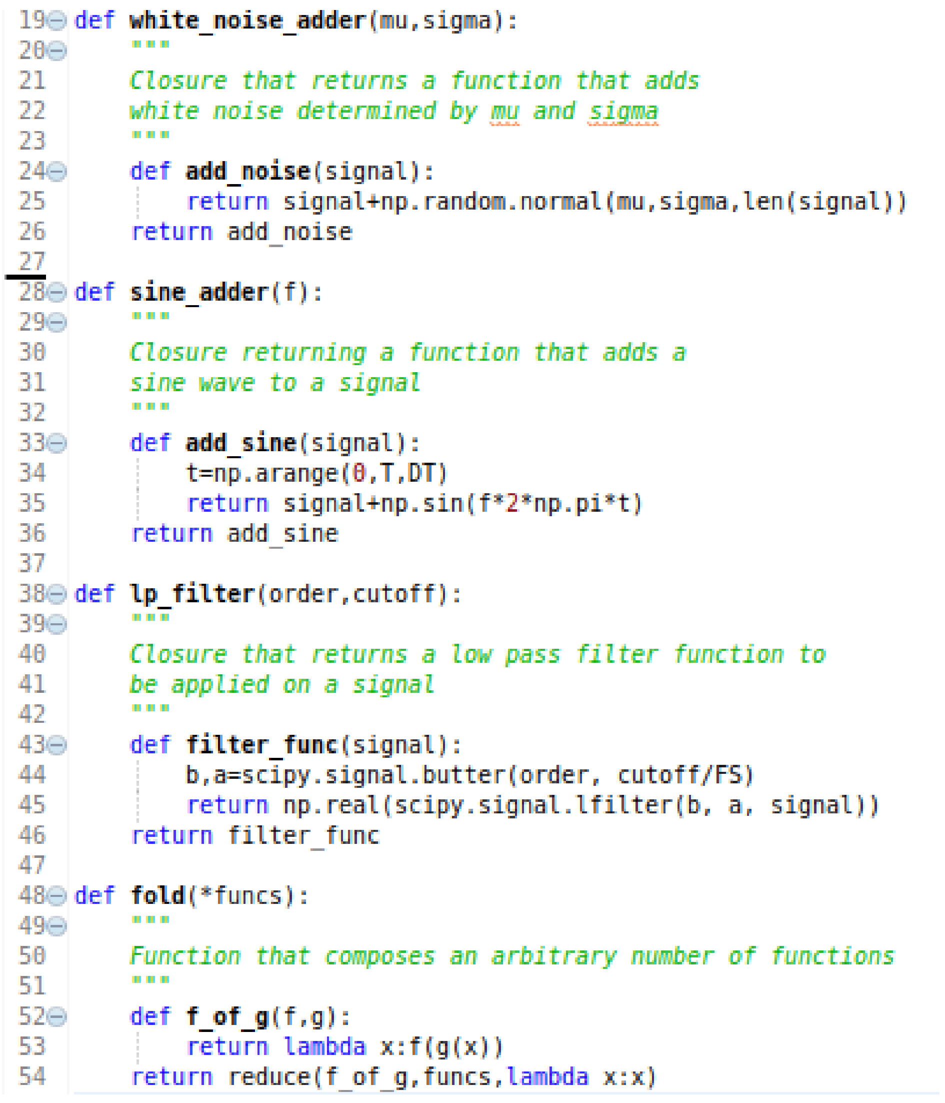


(b) Second plotter() closure , white noise is added to the pipeline



(c) Third plotter() closure ,the signal is filtered for one sinewave and the noise

Figure 7.5: Execution of line 135,136 plotting the signal at different stages of the transformation pipeline using functional programming only.



```python
def white_noise_adder(mu,sigma):
    """
    Closure that returns a function that adds
    white noise determined by mu and sigma
    """
    def add_noise(signal):
        return signal+np.random.normal(mu,sigma,len(signal))
    return add_noise

def sine_adder(f):
    """
    Closure returning a function that adds a
    sine wave to a signal
    """
    def add_sine(signal):
        t=np.arange(0,T,DT)
        return signal+np.sin(f*2*np.pi*t)
    return add_sine

def lp_filter(order,cutoff):
    """
    Closure that returns a low pass filter function to
    be applied on a signal
    """
    def filter_func(signal):
        b,a=scipy.signal.butter(order, cutoff/FS)
        return np.real(scipy.signal.lfilter(b, a, signal))
    return filter_func

def fold(*funcs):
    """
    Function that composes an arbitrary number of functions
    """
    def f_of_g(f,g):
        return lambda x:f(g(x))
    return reduce(f_of_g,funcs,lambda x:x)
```

Figure 7.6: Closures and functions needed to transform an input signal in the functional programming DSL.

8. Functional Programming

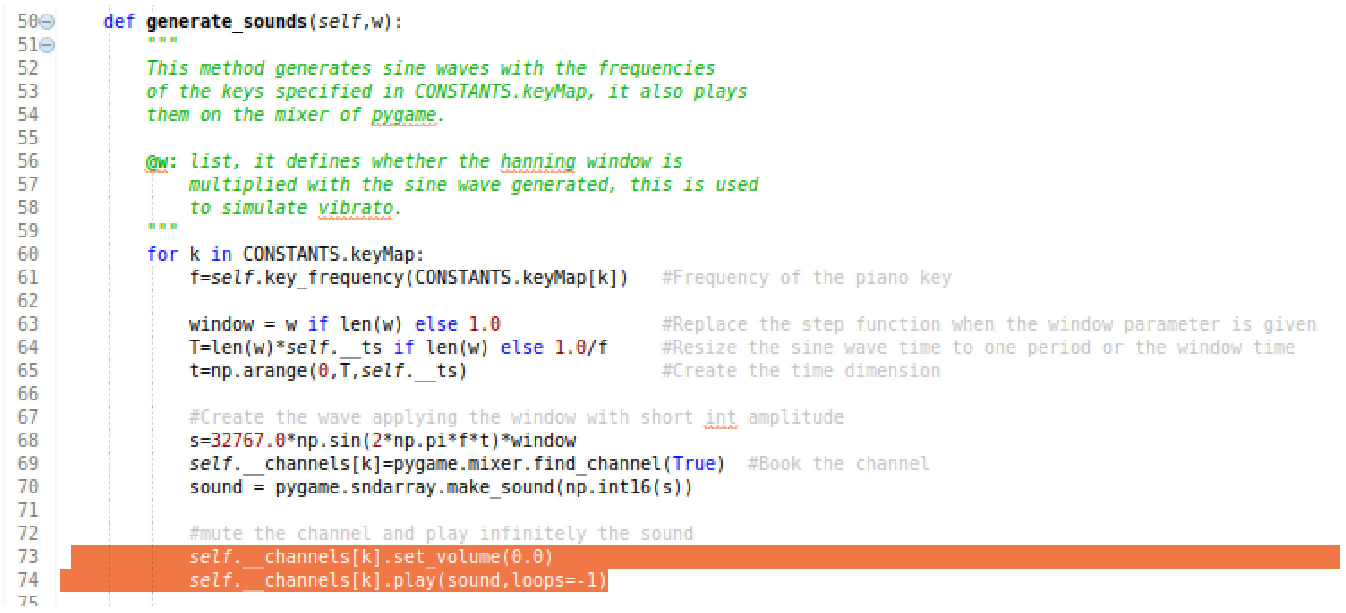
Although the architecture for the development of this project was chosen to be MVP, hence object oriented. The following concepts of functional programming were applied as strictly as possible.

8.1 Final data structures

Python is not a purely functional programming language ,nevertheless the practice of only having final data structures can be achieved by keeping them unmodified : never at the left of the assignment operator[3].

8.2 Side effect free functions

Since the application is purported for sound device output, these side effects are relaxed and thus allowed in the functions. Next figure depicts the extent of what is probably the most complex function in the project. The side effects: only related ot the audio stream are highlighted.

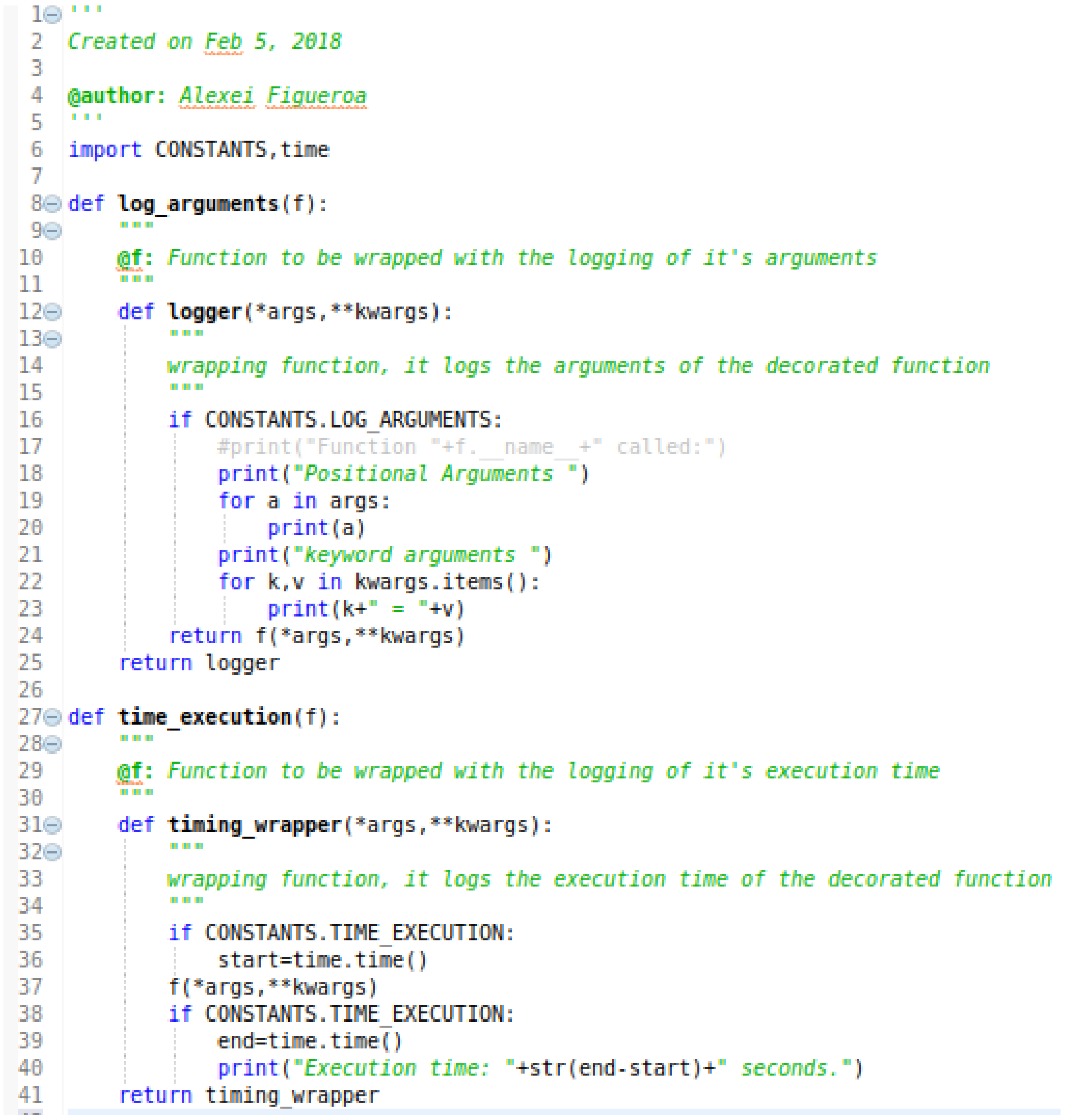


```
    def generate_sounds(self,w):
        """
        This method generates sine waves with the frequencies
        of the keys specified in CONSTANTS.keyMap, it also plays
        them on the mixer of pygame.

        @w: list, it defines whether the hanning window is
            multiplied with the sine wave generated, this is used
            to simulate vibrato.
        """
        for k in CONSTANTS.keyMap:
            f=self.key_frequency(CONSTANTS.keyMap[k])   #Frequency of the piano key

            window = w if len(w) else 1.0               #Replace the step function when the window parameter is given
            T=len(w)*self.__ts if len(w) else 1.0/f     #Resize the sine wave time to one period or the window time
            t=np.arange(0,T,self.__ts)                  #Create the time dimension

            #Create the wave applying the window with short int amplitude
            s=32767.0*np.sin(2*np.pi*f*t)*window
            self.__channels[k]=pygame.mixer.find_channel(True)  #Book the channel
            sound = pygame.sndarray.make_sound(np.int16(s))

            #mute the channel and play infinitely the sound
            self.__channels[k].set_volume(0.0)
            self.__channels[k].play(sound,loops=-1)
```

Figure 8.1: method generate_sounds from the class SignalGenerator, the side effects are highlighted.

8.3 Higher order functions, functions as parameters and return values

As explained above the Aspect Oriented functionality was implemented by the use of decorators, which are indeed higher order functions. Furthermore they receive functions as parameters and also return functions. Additionally the closures developed for the DSL presented in the next session also present this behaviour.

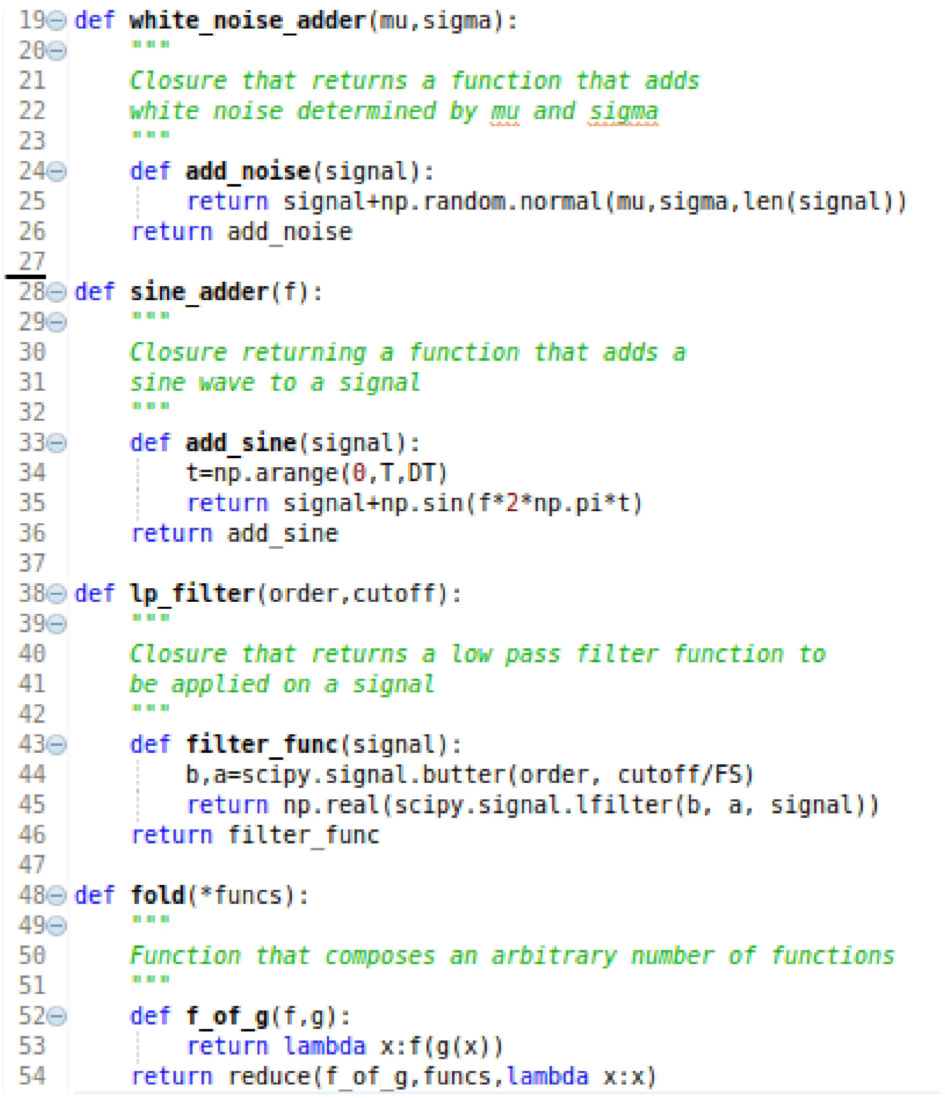


```python
'''
Created on Feb 5, 2018

@author: Alexei Figueroa
'''
import CONSTANTS,time

def log_arguments(f):
    """
    @f: Function to be wrapped with the logging of it's arguments
    """
    def logger(*args,**kwargs):
        """
        wrapping function, it logs the arguments of the decorated function
        """
        if CONSTANTS.LOG_ARGUMENTS:
            #print("Function "+f.__name__+" called:")
            print("Positional Arguments ")
            for a in args:
                print(a)
            print("keyword arguments ")
            for k,v in kwargs.items():
                print(k+" = "+v)
        return f(*args,**kwargs)
    return logger

def time_execution(f):
    """
    @f: Function to be wrapped with the logging of it's execution time
    """
    def timing_wrapper(*args,**kwargs):
        """
        wrapping function, it logs the execution time of the decorated function
        """
        if CONSTANTS.TIME_EXECUTION:
            start=time.time()
        f(*args,**kwargs)
        if CONSTANTS.TIME_EXECUTION:
            end=time.time()
            print("Execution time: "+str(end-start)+" seconds.")
    return timing_wrapper
```

Figure 8.2: Higher order functions for execution timing and logging arguments in the aspects.py module

8.4 Closures / anonymous functions

As mentioned in the DSL chapter, several closuers were developed to achieve a functionally composed pipeline of sound generation, aditionally lambdas (anonymous functions) were used for both the purposes of composition of functions as well as creating signal transformations (`lambda x:2*x` doubles a signal amplitude).

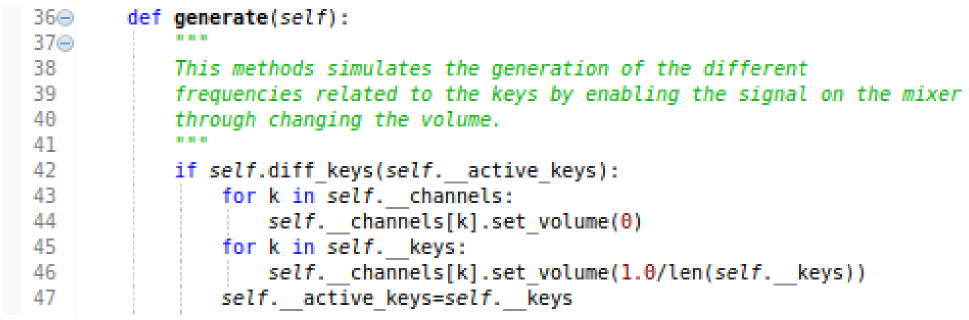


```python
def white_noise_adder(mu,sigma):
    """
    Closure that returns a function that adds
    white noise determined by mu and sigma
    """
    def add_noise(signal):
        return signal+np.random.normal(mu,sigma,len(signal))
    return add_noise

def sine_adder(f):
    """
    Closure returning a function that adds a
    sine wave to a signal
    """
    def add_sine(signal):
        t=np.arange(0,T,DT)
        return signal+np.sin(f*2*np.pi*t)
    return add_sine

def lp_filter(order,cutoff):
    """
    Closure that returns a low pass filter function to
    be applied on a signal
    """
    def filter_func(signal):
        b,a=scipy.signal.butter(order, cutoff/FS)
        return np.real(scipy.signal.lfilter(b, a, signal))
    return filter_func

def fold(*funcs):
    """
    Function that composes an arbitrary number of functions
    """
    def f_of_g(f,g):
        return lambda x:f(g(x))
    return reduce(f_of_g,funcs,lambda x:x)
```

Figure 8.3: Closures and functions needed to transform an input signal in the dsl module.

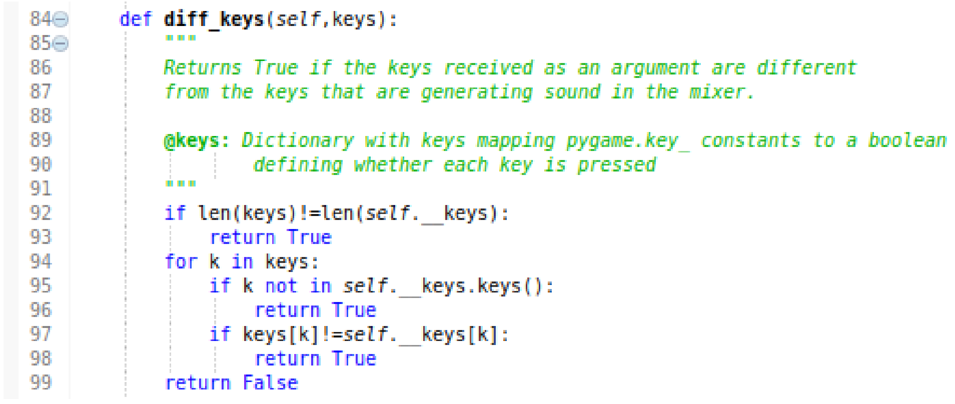
9. Logical solver

The control of the audio output of the program as previously depicted in the state diagram of an individual audio channel is mainly done by switching the volume on or off, for this to happen at each event of a key press, a check is made to see whether the keys pressed are different and then if so, all the channels are switched off and the corresponding active channels are turned on.



```python
def generate(self):
    """
    This methods simulates the generation of the different
    frequencies related to the keys by enabling the signal on the mixer
    through changing the volume.
    """
    if self.diff_keys(self.__active_keys):
        for k in self.__channels:
            self.__channels[k].set_volume(0)
        for k in self.__keys:
            self.__channels[k].set_volume(1.0/len(self.__keys))
        self.__active_keys=self.__keys
```

(a) `generate` method of the `SignalGenerator class`

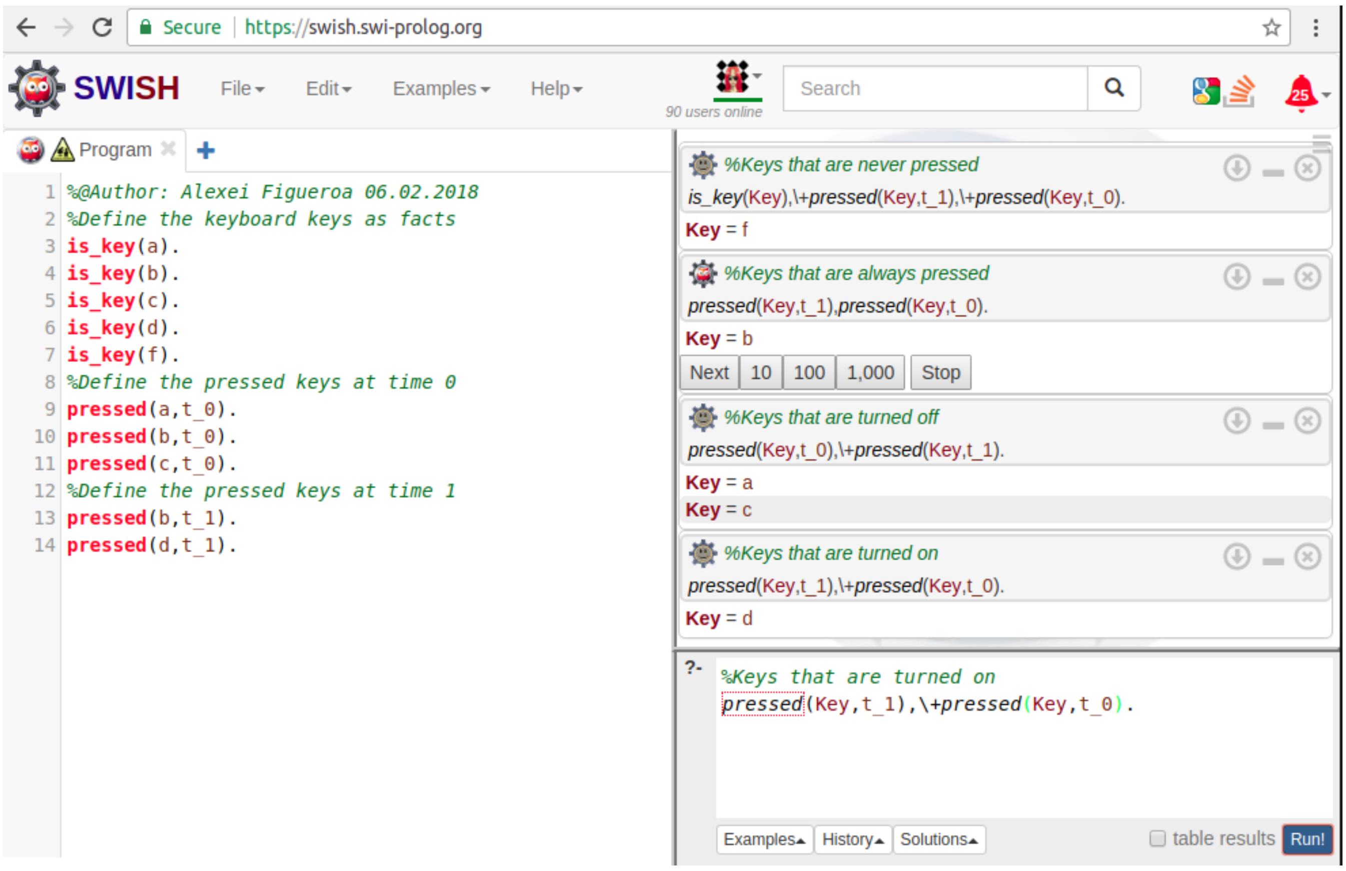


```python
def diff_keys(self,keys):
    """
    Returns True if the keys received as an argument are different
    from the keys that are generating sound in the mixer.

    @keys: Dictionary with keys mapping pygame.key_ constants to a boolean
            defining whether each key is pressed
    """
    if len(keys)!=len(self.__keys):
        return True
    for k in keys:
        if k not in self.__keys.keys():
            return True
        if keys[k]!=self.__keys[k]:
            return True
    return False
```

(b) `diff_keys` method of the `SignalGenerator class`

Figure 9.1: Methods from the SignalGenerator class that control the logic behind the swiching on and off of the audio channels

A way of optimizing this implementation would be to selectively turn on or off the channels that are different between the two distinct points in time and such very simple scenario could be approached by a logical solver. In the following example are shown in `prolog` the definition of the facts and rules for any arbitrary two time steps (key events) ,as well as the queries that would give full information of what channels to turn on or off.


https://swish.swi-prolog.org

```
%@Author: Alexei Figueroa 06.02.2018
%Define the keyboard keys as facts
is_key(a).
is_key(b).
is_key(c).
is_key(d).
is_key(f).
%Define the pressed keys at time 0
pressed(a,t_0).
pressed(b,t_0).
pressed(c,t_0).
%Define the pressed keys at time 1
pressed(b,t_1).
pressed(d,t_1).
```

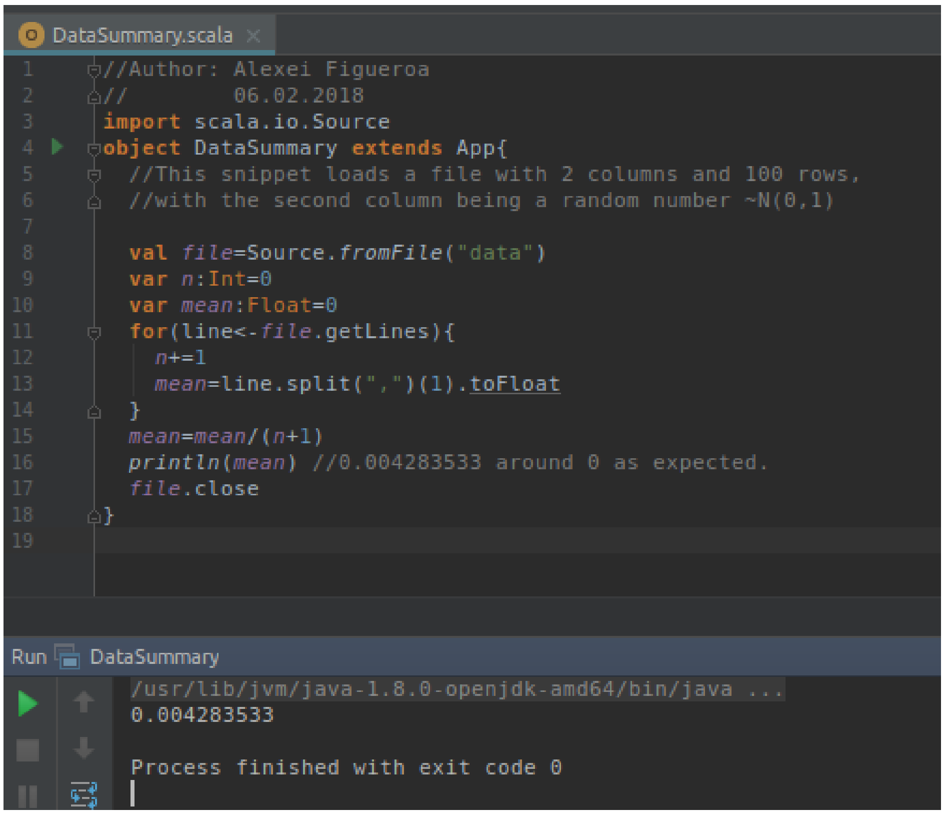
```
%Keys that are never pressed
is_key(Key),\+pressed(Key,t_1),\+pressed(Key,t_0).
Key = f

%Keys that are always pressed
pressed(Key,t_1),pressed(Key,t_0).
Key = b

%Keys that are turned off
pressed(Key,t_0),\+pressed(Key,t_1).
Key = a
Key = c

%Keys that are turned on
pressed(Key,t_1),\+pressed(Key,t_0).
Key = d
```

```
?- %Keys that are turned on
pressed(Key,t_1),\+pressed(Key,t_0).
```

Figure 9.2: Prolog simple model solution to "diff" the channels to activate/deactivate given an key event.

10. Code fragment in Scala

A small fragment of code computing the mean of a set of 100 standard normally distributed random numbers is depicted next.



```scala
//Author: Alexei Figueroa
//        06.02.2018
import scala.io.Source
object DataSummary extends App{
  //This snippet loads a file with 2 columns and 100 rows,
  //with the second column being a random number ~N(0,1)

  val file=Source.fromFile("data")
  var n:Int=0
  var mean:Float=0
  for(line<-file.getLines){
    n+=1
    mean=line.split(",")(1).toFloat
  }
  mean=mean/(n+1)
  println(mean) //0.004283533 around 0 as expected.
  file.close
}
```

```
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
0.004283533

Process finished with exit code 0
```

Figure 10.1: Scala code snippet that reads a csv and computes the mean of the 2nd column.

Bibliography

[1] Cleancodedeveloper.com,*Clean Code Developer: Virtues*
`https://ccd_school.gitbooks.io/clean-code-developer-com/content/01abstractions/03_Virtues.html`
visited on 21.12.2017

[2] Wikipedia,*Model View Presenter*
`https://en.wikipedia.org/wiki/Model-view-presenter`
visited on 21.12.2017

[3] A.M.Kuchling,*Functional Programing HOWTO*
`https://docs.python.org/3.6/howto/functional.html`
visited on 21.12.2017

[4] Wikipedia,*Synthesizer*
`https://en.wikipedia.org/wiki/Synthesizer` visited on 21.12.2017

[5] M. Fowler, *GUI Architectures*
`https://www.martinfowler.com/eaaDev/uiArchs.html` visited on 20.09.2017

[6] M. Fowler*Passive View*
`https://martinfowler.com/eaaDev/PassiveScreen.html`
visited on 20.09.2017