eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA DE TECNOLOGÍA DE TELECOMUNICACIÓN

TRABAJO DE FIN DE GRADO

DESARROLLO DE SOFTWARE DE APOYO A LA ENSEÑANZA DE ACÚSTICA MUSICAL

Alumno: Díez García, Mikel
Directora: Hernáez Rioja, Inmaculada

Curso: 2018-2019

Fecha: Bilbao 30 Mayo, 2019

Resumen Laburpena Abstract

Actualmente las herramientas de software musicales se centran en mercados como los productores musicales o compositores y no siguen un estándar de diseño. El inconveniente de esto es que actualmente no hay herramientas para el aprendizaje de acústica musical teniendo que recurrir al software anterior como única alternativa. Se estudiará y experimentará con dicho software de audio, recolectando información. Se realizará el diseño de la interfaz gráfica y codificación de la funcionalidad de un software destinado a la enseñanza de acústica musical. Esto será posible gracias a los avances informáticos y el software gratuito.

Palabras Clave: Acústica musical, software, educación, herramienta.

Gaur egun, musika softwarearen tresnak musika ekoizleak edo musikagileak bezalako merkatuetan oinarritzen dira eta ez dute diseinu estandarra jarraitzen. Horren desabantaila da gaur egun musika akustika ikasteko tresnarik ez dagoela; beraz, aurreko softwarea alternatiba bakarra bezala erabili behar da. Audio-softwarea aztertuko da, informazioa biltzeko. Interfaze grafikoaren diseinua eta musika akustika irakasten duten softwarearen funtzionalitateen kodeketa burutuko da. Hau posible izango da ordenagailuaren aurrerapenak eta doako softwareari esker.

Gako-hitzak: musika akustika, software, hezkuntza, erreminta.

Currently, music software tools focus on markets such as music producers or composers and do not follow a standard design . The drawback of this is that currently there are no tools for learning musical acoustics having to resort to previous software as the only alternative. We will study and experiment thº audio software, collecting information. The design of the graphical interface and coding of the functionality of a software aimed at the teaching of musical acoustics is carried out. This will be possible thanks to computer advances and free software.

Keywords: Musical acoustics ,software, educational, utility.

Índice

Resumen Laburpena Abstract 1

Lista de figuras 4

Lista de tablas 5

Lista de acrónimos 6

1. Introducción 7

2. Contexto 8

3. Objetivos y alcance 10

3.1. Objetivos . . . 10

3.2. Especificaciones . . . 10

3.3. Diseño de los módulos funcionales . . . 11

3.4. Alcance . . . 11

4. Beneficios 13

4.1. Económicos . . . 13

4.2. Sociales . . . 13

4.3. Técnicos . . . 13

5. Análisis de alternativas 15

5.1. Criterios de selección . . . 15

5.2. Lenguajes de programación . . . 16

5.3. Generación de interfaces gráficas . . . 17

5.4. Distribución . . . 19

6. Descripción de la solución **21**

6.1. Preparación del entorno de desarrollo 22

6.2. Desarrollo del software 26

6.3. Compilación y distribución 31

6.4. Resultados 34

7. Descripción de tareas **39**

7.1. Diagrama de Gantt 40

8. Descripción del presupuesto **42**

8.1. Recursos humanos 42

8.2. Recursos técnicos 42

8.3. Gastos 43

9. Conclusiones **44**

9.1. Codificación de los módulos 44

9.2. Generación de ejecutables 44

9.3. Uso de la aplicación 44

Bibliografía **45**

10. Anexo I: Manual de desarrollo **46**

Lista de figuras

1. Mano Guidoniana. 7
2. Ventana de análisis de espectro de Audacity para una señal de 440hz. . . . 9
3. Cuadros de diálogo para generación de un tono de los programas Audacity y Ocenaudio 9
4. Diagrama de dependencias. El camino verde indica las tecnologías seleccionadas. 20
5. Estructura básica del proyecto 22
6. Estructura del directorio Python 24
7. Esquema del sistema de control de versiones 25
8. Proceso de creación de interfaces 26
9. Diálogo creado con QtCreator 27
10. Pasos para la realización de un instalador 31
11. Ventana principal en plataforma Windows 34
12. Ventana principal en plataforma Mac 34
13. Diálogo para la creación de una nueva señal pura. 35
14. Ventana para la generación de una señal cuadrada. 35
15. Ventana para la síntesis armónica. 36
16. Generación de ruido blanco. 36
17. Ventana para grabación. 37
18. Botones de diálogo cancel y ok. 37
19. Sistema de visualizado extendido en dominio temporal. 37
20. Sistema de visualizado extendido en dominio frecuencial. 37
21. Registro de aplicaciones de Windows. 38
22. Diagrama de Gantt del proyecto. 41

Lista de tablas

1. Comparación para la selección del lenguaje de programación 17
2. Comparación para la selección del sistema de interfaz de usuario 18
3. Comparación para selección de la librería de gráficos 19
4. Comparación para la selección de sistema de creación de ejecutables . . . 19
5. Tecnologías seleccionadas tras el análisis de alternativas 20
6. Tabla de funcionalidades codificadas 30
7. Desgloses de gastos derivados de recursos humanos. 42
8. Material amortizable utilizado durante el proyecto 43
9. Resumen del presupuesto 43

Lista de acrónimos

GPL GNU General Public License

MIT Massachusetts Institute of Technology

FFT Fast Fourier Transform

QML Qt Meta Language

UI User Interface

CSS Cascading Style Sheets

HTTP Hypertext Transfer Protocol

1. Introducción

El campo de la educación ha vivido numerosas transformaciones a lo largo de la historia, tanto favorables como no y la educación musical es un claro ejemplo de ello.

En la edad media ya se enseñaba canto coral a niños, donde se tenía como objetivo aprender todo el repertorio de misa y los oficios, para ello se necesitaba formar a los jóvenes durante 10 largos años. Guido d'Arezzo un monje benedictino y teórico musical del siglo XI logró concentrar todo ese trabajo de 10 años en tan solo dos demostrando que con mejores sistemas educativos y herramientas el mismo trabajo se puede hacer en menos tiempo.

En la figura 1 se muestra el sistema diseñado para facilitar la memorización de las notas mediante reglas nemotécnicas.

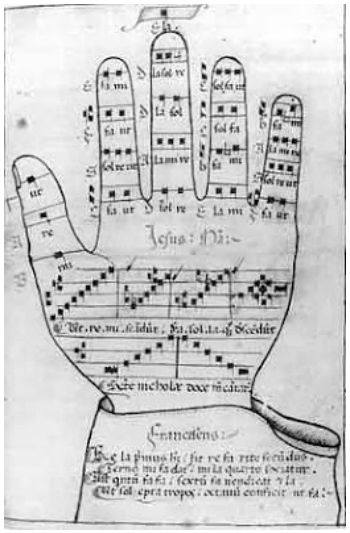


Figura 1: Mano Guidoniana.

En el siglo XVIII Jean-Jacques Rousseau, ideó otro sistema donde los números reemplazaban el nombre de las notas. Debido a ello el número de personas aficionadas a tocar un instrumento se vio incrementado.

Esto demuestra que un buen sistema y unas herramientas bien realizadas pueden conseguir enseñar de una mejor manera.

En la actualidad existen centros especializados en enseñanza musical y es para uno de estos centros que se realiza el desarrollo vía software de una herramienta de apoyo a la enseñanza musical.

2. Contexto

En la enseñanza de acústica musical es común encontrarse conceptos y terminología relacionada con las matemáticas y física como por ejemplo, las funciones sinusoidales, la propagación de ondas en un medio o el dominio temporal y frecuencial.

El mundo digital ha creado nuevas profesiones en el campo de la música y el músico actual tiene muchas probabilidades de desempeñar su trabajo haciendo uso de herramientas digitales.

La enseñanza de música y en concreto el área de acústica musical puede beneficiarse mucho de los avances digitales ya que con el auge de la programación software nuevas herramientas y utilidades son creadas y puestas a disposición de los músicos. Aunque gran parte de este software presenta una serie de complicaciones:

- Terminología compleja o ambigua.
- Cada diseñador usa sus reglas y no siguen una norma común.
- Modificar los parámetros de configuración de las aplicaciones es complicado y en ocasiones es requerido para las tareas mas básicas, esto hace que la curva de aprendizaje sea abrupta desde el primer momento en el que se usa la aplicación.
- Los manuales a menudo usan terminología técnica con la que el usuario no es familiar.
- La terminología puede crear errores de concepto.

Un ejemplo de este problema se puede apreciar en la figura 2 correspondiente al programa Audacity [1], donde se muestra una ventana del cálculo del espectro de una señal de 440 Hz [2]. En la imagen aparecen términos como el envenenando o el tipo de algoritmo usado para el análisis dando por hecho que se conocen esos parámetros.

El problema de la estandarización de términos puede ser uno de los mas comunes ya que cada desarrollador quiere introducir su propia terminología.

En la figura 3 se puede observar la ventana de creación de una señal básica para dos programas diferentes como son *Audacity* y *Ocenaudio*[3].

Se aprecian claras diferencias entre los dos destacando la mas notable el uso del término **Amplitude** por parte de *Audacity* e **Intensity** por parte de *Ocenaudio* para referirse ambos al concepto de **Amplitud de una señal.**

[1] Audacity es un software de producción musical gratuito y primera alternativa a descargar por los usuarios.

[2] Tono correspondiente a la nota musical La

[3] Ocenaudio es una alternativa software gratuita para la producción y edición de audio.

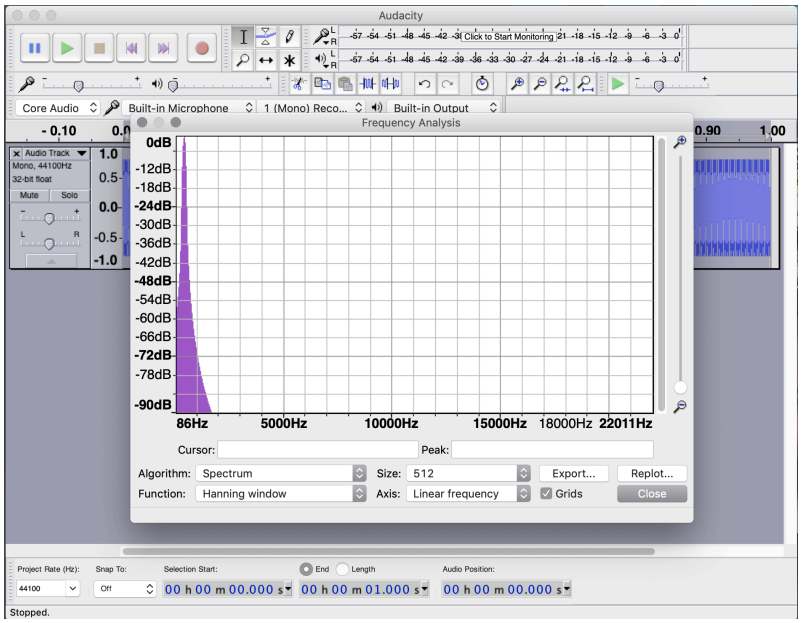


Figura 2: Ventana de análisis de espectro de Audacity para una señal de 440hz.

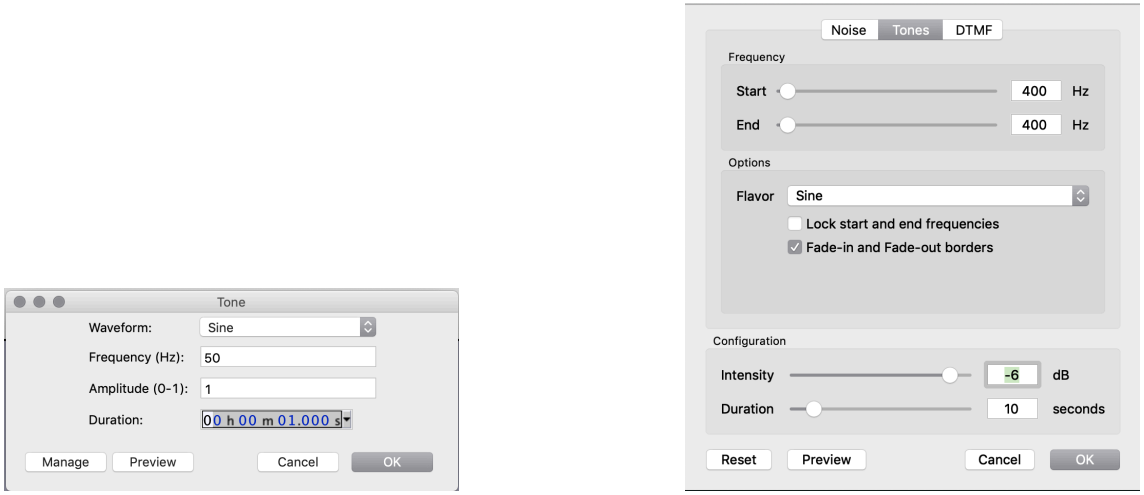


Figura 3: Cuadros de diálogo para generación de un tono de los programas Audacity y Ocenaudio

En este contexto surge la propuesta de realizar un software de apoyo a la enseñanza de los conceptos básicos de la acústica musical.

3. Objetivos y alcance

3.1. Objetivos

Este trabajo tiene diferentes finalidades que pueden ser clasificadas según su nivel de impacto.

- **Objetivos principales**
  1. Desarrollar un prototipo de software de apoyo a la enseñanza de acústica musical.
  2. Realizar un manual de desarrollo, que contenga la información necesaria para continuar con la codificación de la aplicación por parte de otros desarrolladores.
- **Objetivos secundarios**
  1. De fácil uso para cualquier usuario
  2. Acceso universal independientemente del sistema operativo que lo ejecute
  3. Estructura modular para facilitar el desarrollo de ampliaciones

3.2. Especificaciones

En este apartado se listan y justifican las especificaciones técnicas de la aplicación sin entrar en la funcionalidad específica (que se describirá en el apartado siguiente). Las especificaciones de este proyecto son las siguientes:

1. **Instalación:** La aplicación deberá contar con un programa específico que realice la labor de instalarse y eliminarse de sistema en el que resida. Esto mejora la experiencia de usuario eliminando configuraciones previas que tenga que realizar el usuario final.
2. **Plataforma:** La aplicación deberá ser multiplataforma de manera que se podrá ejecutar correctamente en los 3 Sistemas Operativos de escritorio (*Windows*, *Linux* y *macOS*) con el objetivo de lograr mayor disponibilidad.
3. **Código Abierto:** Tanto la aplicación como la totalidad de las herramientas utilizadas y desarrolladas contarán con licencias *GPL* o *MIT*.

3.3. Diseño de los módulos funcionales

El diseño del software se dividirá en módulos, de manera que cada módulo hace referencia a ciertas funcionalidades.

- **Módulo I: Representación de señales básicas y análisis de Fourier**

    1. Senoides puras y visualización de parámetros básicos
    2. Composición de señales básicas periódicas y no periódicas
    3. Análisis de Fourier: definición y representación en frecuencia

- **Módulo II: Digitalización de señales de audio**

    1. Frecuencia de muestreo: criterios para la elección de la frecuencia de muestreo.
    2. Rangos de frecuencias de las señales de audio (voz hablada y cantada, música)
    3. Aliasing: se propone presentar el fenómeno del aliasing y sus implicaciones
    4. Ejercicios de digitalización de señales por parte de los alumnos (variar diferentes parámetros). Guardar los sonidos para hacer ejercicios posteriormente
    5. Cuantificación: criterios para la selección del número de bits por muestra
    6. El flujo binario o bitrate (bits/seg) como resultado del muestreo
    7. Presentación de los estándares habituales (calidad CD, audio profesional)
    8. Compresión de audio (mpeg).

- **Módulo III: Digitalización de señales de audio**

    1. Caracterización frecuencial de las señales a lo largo del tiempo: el espectrograma
    2. Técnica de cálculo del espectrograma: parámetros implicados voz: espectrograma de banda ancha y de banda estrecha
    3. Interpretación del espectrograma para diferentes tipos de sonidos: vibrato, glisando

3.4. Alcance

El alcance de este proyecto se establece en la realización de la estructura básica de la aplicación y la codificación del primer módulo.

- **Estructura básica de la aplicación:** Codificación de una herramienta de comandos vía terminal para la producción de software. Esta herramienta concentra todas las operaciones que el desarrollador tiene que realizar cada vez que necesite distribuir su aplicación y permite la generación de nuevas aplicaciones.

  Funciones de herramienta de desarrollo:

  - Generación de proyectos *PyQt* con librerías incluidas
  - Generación de entornos virtuales de *Python* de manera automática
  - Congelación de dependencias con soporte para entornos virtuales
  - Generación de ejecutables multiplataforma
  - Generación de informes de error

- **Módulo 1:** Las funcionalidades respectivas al primer módulo se han descrito en el apartado de diseño de la aplicación en la sección 3.3.

4. Beneficios

4.1. Económicos

Aunque el software sea gratuito no desaparecen los beneficios económicos. Aquí se contemplan unos beneficios relacionados con el ámbito del desarrollo de software.

- **Explotación de las utilidades:** Con las dependencias de desarrollo descritas en la sección 6.1.2 podría construirse **todo tipo de software de escritorio**. Esto ofrece dos posibilidades:
  - Distribuir las utilidades para construir software de escritorio de propósito general bajo una licencia de pago. De esta manera desarrolladores podrían construir software con ellas.
  - Desarrollar software de escritorio usando las utilidades anteriores y distribuirlo bajo una licencia de pago.

4.2. Sociales

Tanto las dependencias de desarrollo como el software final cuentan con licencias *GPL* [1], esto ofrece una serie de beneficios que repercuten en el ámbito de la enseñanza musical y en el del desarrollo de software.

- **Software gratuito:** Ya que se trata de un software que se puede adquirir sin ningún tipo coste económico dará la posibilidad a estudiantes de acústica musical de usarlo para sus estudios.

4.3. Técnicos

Los beneficios técnicos se centran la posibilidad de la explotación de la herramienta por desarrolladores terceros. Esto se debe a que las herramientas y el código fuente se pueden conseguir de manera gratuita desde su repositorio en internet.

- **Herramientas Libres:** Algunas herramientas necesarias para el proyecto necesitaron ser modificadas, ya que requerían de funcionalidades mas avanzadas. Mejorar estas herramientas dará la posibilidad a otros desarrolladores de utilizarlas en

[1] Tipo de licencia de software común el proyectos libres

sus proyectos llegando a recibir nuevas funcionalidades programadas por ellos y así, evolucionar aún más dicha herramienta.

- **Proyecto de código libre:** Todo el código fuente se encuentra en Internet y puede descargarse de manera gratuita desde su repositorio. Esto permitirá a otros desarrolladores usar la herramienta para sus propios fines pudiendo llegar a incluir nuevos módulos funcionales.

5. Análisis de alternativas

5.1. Criterios de selección

En el análisis de alternativas se va a realizar la comparación y selección de todos los componentes que serán necesarios para la realización del proyecto. Dichos componentes se han organizado en las siguientes categorías:

- **Librerías matemáticas y de tratamiento de señal**: Ya que el software debe implementar funciones para la generación y análisis de señales, se considera imprescindible que las librerías matemáticas y de tratamiento de señal aporten las funciones de cálculo básicas necesarias. Estas funciones son:
  - Manipulación de listas o arrays numéricos iguales o superiores a 48.000, ya que esta es frecuencia de muestreo por defecto de la aplicación
  - Funciones sinusoidales
  - Funciones periódicas
  - Análisis de Fourier (FFT)
  - Cálculo del espectrograma
  - Reproducción de estas señales usando la tarjeta de sonido del ordenador que ejecuta la aplicación
- **Librerías de interfaces gráficas**: La interfaz gráfica se encarga de la presentación de datos al usuario y hace posible que este interactúe con la aplicación. Estas librerías se encargarán de dibujar las ventanas y todos los elementos que estas contienen además de darles una funcionalidad. Las funcionalidades básicas que estas librerías deberán tener son:
  - Botones para ejecutar órdenes
  - Cuadros para ingresar números o texto
  - Gráficas para visualizar datos
  - Integración con el sistema de ficheros del ordenador que ejecuta el software
- **Utilidades de desarrollo y distribución**: Para el desarrollo de la aplicación son necesarias herramientas que aunque no agregan funcionalidad del lado del cliente, generan utilidad a la hora del desarrollo. Estas herramientas son útiles para codificar haciendo que el desarrollador solo se preocupe de implementar la funcionalidad concreta de la aplicación. Las herramientas necesarias para el desarrollo son:
  - Utilidades de depuración de código

- Herramientas gráficas de diseño de interfaces
- Generación de instaladores automáticos
- Generación de informes de errores para posibles anomalías de ejecución de la aplicación

Una vez seleccionadas las necesidades se identifican los elementos sobre los que se realizará el análisis de alternativas. Estos elementos son el **lenguaje de programación**, la **generación de interfaces gráficas** y la **distribución**.

5.1.1. Valoración y puntuación de las alternativas

Se han analizado las características referentes a la sencillez, rendimiento, flexibilidad, coste económico y coste de aprendizaje de cada una de las alternativas:

La puntuación para cada una de ellas tendrá un rango de cero a diez, siendo cero la peor puntuación posible y diez la mejor.

- **Sencillez:** Define el grado de sencillez con la que se integra la alternativa en el proyecto. y tendrá un peso del 15 % sobre el total.
- **Rendimiento:** Se analizará cómo se comporta la alternativa cuando sea necesario operar con cantidades de datos grandes, sin sufrir congelaciones de la interfaz de usuario. Tiene un peso del 15 % sobre el total.
- **Flexibilidad:** Define cómo se comporta la alternativa a medida que el proyecto software crece en tamaño. Se valorará con un peso 25 % sobre el total.
- **Coste Económico :** Inversiones económicas asociadas a la elección de esa alternativa. Se valora con un peso del 15 % sobre el total.
- **Coste de Aprendizaje:** Tiempo que se necesita para aprender a usar la alternativa. Se valora con un peso del 30 % del total.

5.2. Lenguajes de programación

Se han analizado los lenguajes de programación *Python* y *Javascript*. No se han valorado soluciones como *Matlab* o *Java* ya que un requisito fundamental es que el software cuente licencia *GPL*. La licencia *GPL* hace posible que usuarios terceros modifiquen el software además de protegerlo frente a apropiaciones indebidas o robos. Incluir un lenguaje que no cuente con licencia *GPL* automáticamente elimina la posibilidad de poseer dicha licencia.

- ***Python*:** Lenguaje por excelencia en el ámbito científico con librerías muy útiles de tratamiento de señal y cálculo numérico. Su código es capaz de ejecutarse en cualquier entorno, gracias a que es un lenguaje interpretado. También es posible compilar código. Por otro lado, la generación de interfaces es compleja y se apoya en herramientas de terceros. Además, existen multitud de librerías desarrolladas para *Python* con muy variada funcionalidad.

- ***Javascript*:** Cuenta con librerías de cálculo numérico pero después de realizar una serie de pruebas, se comprobó que no tenían el suficiente rigor científico. Esto puede ser debido a que han sido diseñadas para aplicaciones multimedia para ocio y entretenimiento. Además la documentación es escasa. Al igual que *Python* su código es interpretado y también puede compilarse. Como principal ventaja, cuenta con buenas librerías para el desarrollo de interfaces gráficas capaces de integrar tecnologías web.

La tabla 1 muestra los criterios de valoración y la puntuación total obtenida por cada una de las alternativas.

	Python	Javascript
Sencillez	8	9
Rendimiento	8	6
Flexibilidad	8	8
Coste Económico	10	10
Coste Aprendizaje	7	8
Total	8.2	8.2

Tabla 1: Comparación para la selección del lenguaje de programación

5.3. Generación de interfaces gráficas

Las librerías de generación de interfaces gráficas analizadas han sido *Qt*, *Kivy* y las tecnologías Web. La figura 4 tal muestra las dependencias entre los lenguajes de programación y las librerías de interfaces gráficas, esto es debido a que en muchos casos las librerías y lenguajes no son interoperables.

La figura 2 muestra las puntuaciones obtenidas por las diferentes alternativas en el momento en el que se realizó el análisis.

- **Qt:** Ofrece un conjunto de herramientas muy completo para la creación de interfaces de aplicaciones de escritorio. Es multiplataforma y tiene soporte para *Python* 3 en su variante *PyQt*. Se necesita recompilar su código en *QML*[1] a un código funcional en *Python*. Al estar escrita en *Python* no da soporte para *Javascript*.
- **Kivy:** Íntegramente en *Python* y multiplataforma, no cuenta con un editor gráfico con lo que la dificultad de diseño aumenta ya que es necesario realizar la programación de la interfaz de manera manual.

  Su sistema de generación de ejecutables depende de herramientas de terceros.
- **HTML, CSS JavaScript y NodeJS:**.
  - HTML y CSS: Estas dos tecnologías permiten estructurar interfaces gráficas añadiendo elementos pero sin lógica y con funcionalidad limitada.
  - JavaScript: Permite codificar la parte de la lógica e incrustarla en la interfaz definida previamente con *HTML* y *CSS*.

[1] Lenguaje para interfaces de *Qt*

- NodeJS: Las tecnologías anteriores, conocidas como **tecnologías Web** solo son compatibles si las ejecuta un navegador de internet, *NodeJS* permite ejecutar estas tecnologías sin la necesidad de un navegador instalado, de esta manera se pueden construir interfaces gráficas para escritorio.

  Aunque la creación de interfaces es sencilla debido a la gran cantidad de documentación, el rendimiento que ofrecen es mucho mas bajo que el de las alternativas listadas anteriormente, la razón de este bajo rendimiento es que *NodeJS* solo es capaz de ejecutar instrucciones en un solo hilo sin la posibilidad de abrir procesos paralelos.

	Qt	Kivy	Web
Sencillez	6	2	9
Rendimiento	8	7	5
Flexibilidad	7	2	10
Coste Económico	10	10	10
Coste Aprendizaje	8	9	5
Total	7.8	6	7.8

Tabla 2: Comparación para la selección del sistema de interfaz de usuario

5.3.1. Librería de visualización de datos

Ninguna librería de las anteriores incluye una utilidad para visualizar datos. Es por ello que será necesario añadir una compatible. La compatibilidad viene reflejada en la figura 4.

- **PyQtgraph:** Se puede integrar fácilmente en un entorno *Qt* ya que fue concebida para eso. Además es capaz de representar gran cantidad de datos con muy buen rendimiento, esto es debido a que la librería está optimizada para ello.
- **Matplotlib:** Librería *de facto* para visualización de datos en *Python*. De uso muy simple. No ofrece personalización pero cuenta con botones útiles para hacer zoom y recortar áreas. El rendimiento no es tan bueno como el de *PyQtgraph* para las mismas pruebas realizadas.
- **Librerías gráficas de Javascript:** Este lenguaje soporta cualquier librería de visualización de datos compatible con tecnologías web. A continuación se listan algunas de ellas:
  - Chartist.js
  - D3.js

  Ambas heredan el problema descrito anteriormente de *NodeJS* y su único hilo de ejecución.

	PyQtgraph	Matplotlib	Javascript
Sencillez	6	9	9
Rendimiento	8	5	2
Personalización	10	2	10
Coste Aprendizaje	7	3	7
Total	7.75	4.75	7

Tabla 3: Comparación para selección de la librería de gráficos

5.4. Distribución

Aunque parte de las librerías y utilidades presentados en este análisis de alternativas están concebidos para ejecutarse en sus respectivos interpretes, para este proyecto será necesario que el software pueda ejecutarse en cualquier dispositivo objetivo sin tener que instalar las librerías y paquetería analizadas anteriormente. Para la generación de ejecutables se estudiaron *Fman Build System*, *Electron* y *Meteor*, la figura 4 muestra las compatibilidades de las mismas.

- **Fman Build System:** También llamada simplemente *fman*, es una dependencia de desarrollo que contiene muchas utilidades en su interior, con el objetivo de crear ejecutables independientes usando *Python* como código base.

  Cuenta con la característica de poder codificar funciones tanto en *Python* como en *Bash* para automatizar comandos o crear nuevas funcionalidades. Las Herramientas mas relevantes que ofrece son:

  - Py2exe: Creación de ejecutables en *Windows*
  - fbm: Creación de ejecutables en *Linux*
  - yoursway-create-dmg: Creación de ejecutables en *macOS*

  Todas ellas tienen como objetivo crear software que puede ser ejecutado si usar el interprete de Python.

  La parte negativa reside en que no cuenta con la función de generación de instalables.

- **Electron:** Compila *Javascript* a un estado nativo aunque con matices y el rendimiento deja que desear ya que sufre del mismo problema que *NodeJS* comentado anteriormente.

- **Meteor:** Similar a *Electron* con el añadido de que incluso puede producir aplicaciones para dispositivos móviles.

	fman	Electron	Meteor
Sencillez	5	8	7
Rendimiento	8	5	5
Flexibilidad	9	9	9
Coste Económico	10	10	10
Coste Aprendizaje	8	7	7
Total	8	7.8	7.6

Tabla 4: Comparación para la selección de sistema de creación de ejecutables

La elección de *Python* como lenguaje condicionó una parte del análisis, ya que las librerías entre los lenguajes *Python* y *Javascript* no son interoperables de manera nativa lo que complica el desarrollo teniendo que codificar módulos que realicen de pasarela entre un lenguaje y otro.

En la tabla 5 se muestra el resultado del análisis de las tecnologías seleccionadas para el desarrollo de la aplicación.

Lenguaje	Python
Sistema de ventanas	PyQt
Visualización de datos	PyQtgraph
Compilador	fman

Tabla 5: Tecnologías seleccionadas tras el análisis de alternativas

La figura 4 muestra en un diagrama de dependencias las compatibilidades de todas las alternativas y las posibles combinaciones entre dependencias que se pueden realizar. El camino verde indica las tecnologías seleccionadas al final del análisis.

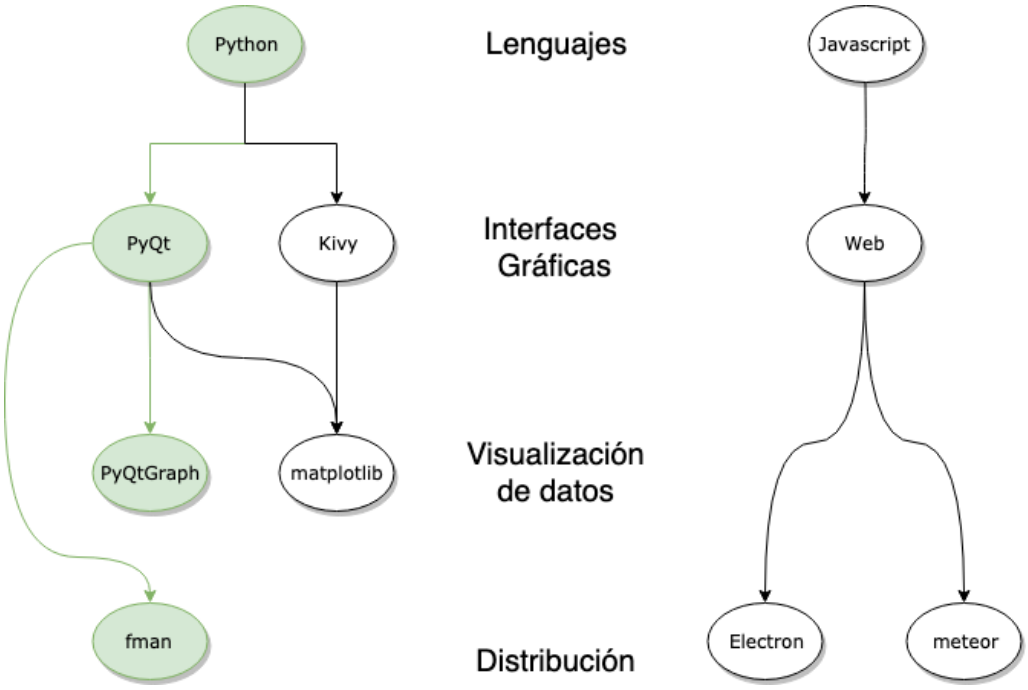


Figura 4: Diagrama de dependencias. El camino verde indica las tecnologías seleccionadas.

6. Descripción de la solución

Se pasan a detallar las tareas que fueron necesarias para la realización del software en su integridad. Estas tareas se dividieron en diferentes bloques según su función dentro del proyecto. A continuación se muestran los tres bloques que conforman el desarrollo del software y cada una de sus funciones asociadas:

1. **Preparación del entorno de desarrollo:** Ciertas librerías elegidas en el análisis de alternativas requieren de una configuración previa para su correcto funcionamiento.
    - **Estructura básica del proyecto:** Estructura raíz con una clara diferenciación entre los tipos de ficheros que serán necesarios para el proyecto.
    - **Control de versiones:** El desarrollo se realiza usando un sistema de control de versiones, esto permite saber cual es el estado del proyecto en todo momento y tener un seguimiento de los fallos y las correcciones a implementar.
2. **Desarrollo del software:** Todo lo relativo al diseño de la interfaz y a la codificación de los distintos módulos del software.
    - **Diseño de la interfaz:** Diseño de las interfaces de usuario usando la herramienta *QtCreator* y el retranspilado[1] a clases de *Python*.
    - **Codificación de la funcionalidad:** Codificación de toda la lógica necesaria para que el software cuente con la funcionalidad especificada en los módulos de los objetivos.
3. **Compilación y distribución:** Generación de los ejecutables e instaladores para el usuario final.
    - Congelación de las dependencias necesarias para construir los ejecutables y codificación de los *scripts* para la generación tanto de instaladores como de ejecutables.
    - Generación del instalable para su distribución.

[1] Acción de traducir un lenguaje programación a otro

6.1. Preparación del entorno de desarrollo

6.1.1. Estructura básica del proyecto

Como se observa en la figura 5 los ficheros se han organizado usando la misma jerarquía empleada en los paquetes de Python presente en la documentación oficial[2], al tratarse de un estándar especifíca reglas para la organización de los directorios y la nomenclatura de los módulos y ficheros de código *Python*.

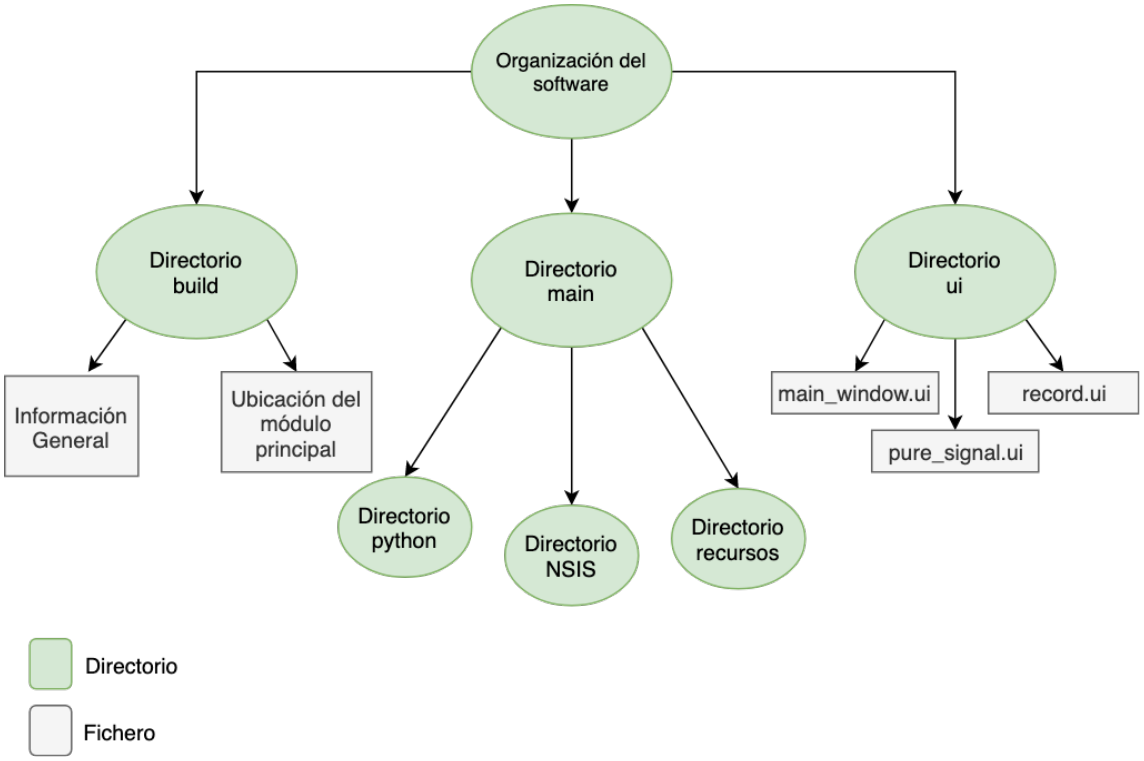


Figura 5: Estructura básica del proyecto

A continuación se describe la funcionalidad de los directorios y ficheros más relevantes:

- **Directorio *build*:** Incluye los directorios y ficheros de configuración básicos y específicos de plataforma tales como:
  - Nombre del proyecto
  - Versión actual
  - Autor
  - Ubicación del módulo principal
  - Ubicación de dependencias de característica especial
  - Ficheros de configuración para los sistemas operativos *Linux, Mac y Windows* donde especificar criterios especiales para cada plataforma

- **Directorio *main*:** Ficheros que necesita el programa, scripts de código, imágenes y hojas de estilos. Estos ficheros se encuentran repartidos en los directorios de Python, Iconos y Recursos respectivamente. La generación de instalables en Windows

[2] https://packaging.python.org/tutorials/packaging-projects/

requiere de paquetería adicional, esta paquetería esta contenida en el directorio NSIS.

- Iconos
- NSIS: Generador de instalables
- Python: Código fuente
- Recursos: Hojas de estilo, imágenes y fuentes de texto.

- **Directorio de ficheros de interfaz gráfica (*UI*):** Serán reconvertidos a código *Python* de tal modo que sea posible acceder a todas sus funciones simplificando el desarrollo.

6.1.1.1. Directorio Python

Define el punto de entrada de la aplicación, esto quiere decir que contiene todos los módulos *Python* codificados para el funcionamiento del programa, los módulos *Python* contienen todos los ficheros de código fuente separados cada uno por la función que desempeñan.

- **Fichero *main.py*:** Punto de entrada, gestiona el uso de todos los módulos y renderiza la ventana principal del programa.
- **Módulo de diálogos:** Los diálogos se encargan de recoger los datos que introduzca el usuario.
- **Módulo de utilidades:** Las utilidades se definen como una interfaz de programación hacia los diálogos ya que les ofrecen una serie de clases y métodos.

En la figura 6 se muestra el esquema del directorio *Python* de la aplicación con algunos de los ficheros y directorios usados para la codificación del software.

Para la nomenclatura de los módulos y ficheros se consultó la guía de estilo de *Python* presente en la documentación oficial que especifíca el uso de minúsculas y separadores usando el carácter '_' como se aprecia en la figura 6.

6.1.2. Paquetería de propósito general y entorno virtual

La paquetería acumula clases *Python* con métodos y variables útiles para el desarrollo.

Para una mejor gestión de los recursos se integró un entorno virtual de *Python* con toda la paquetería de propósito general que hiciese falta.

Un entorno virtual permite contener paquetería *Python* diferente a la paquetería global que puede existir en un sistema, de modo que se puede conseguir trabajar en un entorno aislado instalando solo la paquetería necesaria.

La principal ventaja de este tipo de entornos es que el proyecto se puede transportar de una máquina a otra llevando en él todo entorno virtual o generando uno a partir de un fichero de configuración de nombre *reqs.txt* que contiene nombre y versión necesaria de la paquetería usada.

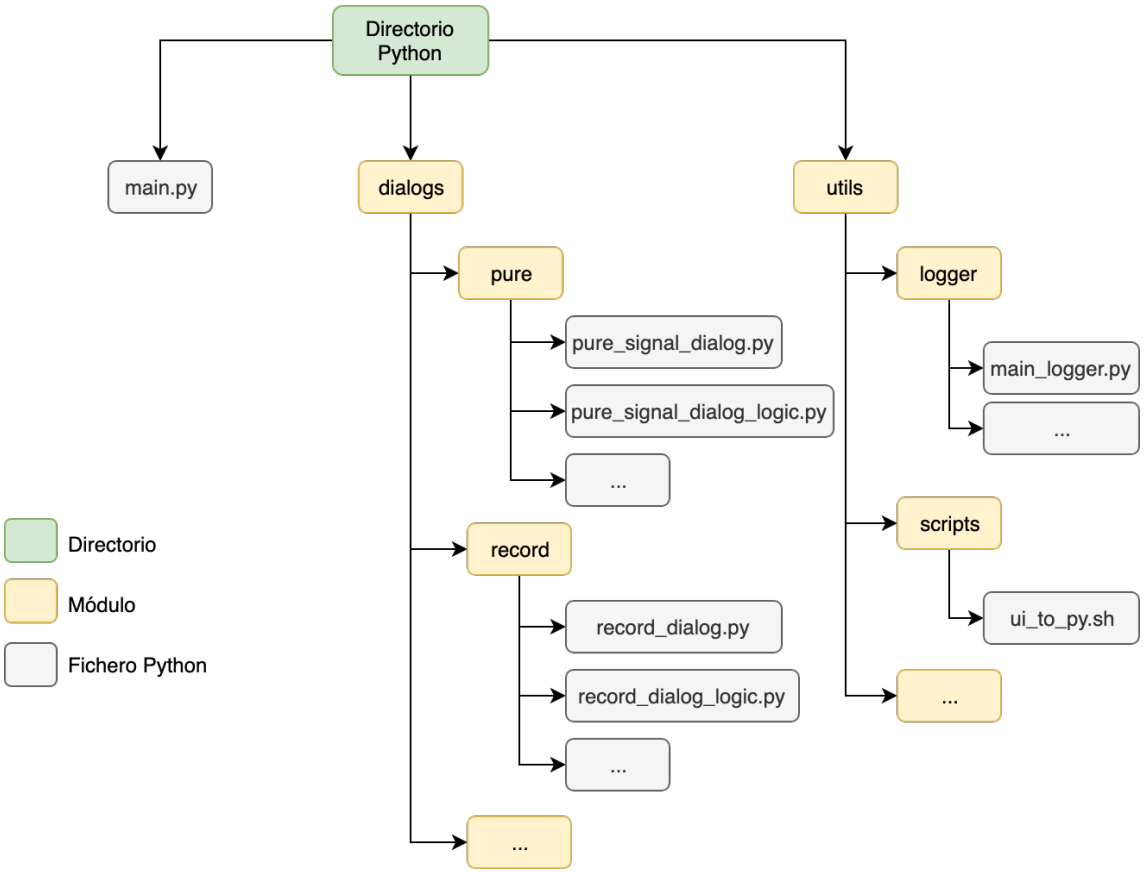


Figura 6: Estructura del directorio Python

Estos paquetes se diferencian por su función, existe paquetería que se requiere para la implementación en código de los módulos definidos en el alcance del proyecto y paquetería que tiene un propósito de desarrollo como por ejemplo llevar un *registro* de fallos.

Las dependencias instaladas y su versión para la realización de los módulos fueron:

- **numpy (1.15.4):** Utilidades de cálculo numérico.
- **PyQt5 (5.11.3):** Gestión y creación de interfaces gráficas.
- **PyQt5-sip (4.19.13):** Funciones adicionales a *PyQt5*, añade mas elementos de interfaz gráfica que *PyQt5* y definiciones *Python* para *c* y *c++*.
- **pyqtgraph (0.10.0):** Librería para realizar gráficas.
- **requests (2.21.0):** Permite realizar peticiones HTTP simples.
- **SciPy (1.1.0):** Incluye algunas funciones usadas en tratamiento de señal útiles para este proyecto.
- **simpleaudio (1.0.2):** reproducción de audio.
- **soundddevice (0.3.12):** reproducción de audio.
- **PyAudio (2.10.0):** Permite la grabación de audio a través de una linea de entrada.

Las dependencias de desarrollo fueron:

- **fman o fbs (0.5.3):** Contiene utilidades software para crear aplicaciones de escritorio sus características se encuentran desarrolladas en sección 5.4.
- **pip (9.0.1):** Gestión de la paquetería python.
- **PyInstaller (3.4):** Permite construir ejecutables e instaladores.
- **py2exe (0.9.2.2):** Permite construir ejecutables en windows.

6.1.3. Control de versiones

Empleando la herramienta de control de versiones *git* el desarrollo del software se dividirá en ramas, cada rama representa el estado del proyecto en un momento determinado. La figura 7 muestra como se crearon las ramas maestra, desarrollo y menú básico.

- **Rama maestra:** Siempre tendrá código estable y probado, cada versión del programa acabará en esta rama.
- **Rama maestra de desarrollo:** De esta rama salen ramas de desarrollo menores, esas ramas menores volcarán sus cambios en esta rama. Una vez estos cambios están integrados con el resto, la rama maestra de desarrollo volcará sus cambios en la rama maestra.
- **Ramas menores:** Cada funcionalidad añadida o mejora añadida tiene su rama correspondiente, cuando esta sea completada los cambios se volcarán en la rama de desarrollo. Un caso de uso sería crear una rama para añadir una nueva función al menú o arreglar un fallo, como se observa en la figura 7.

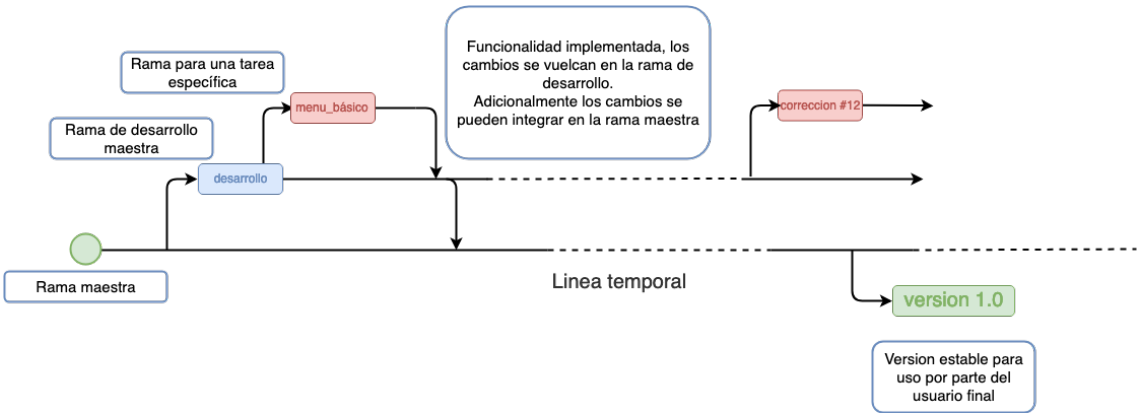


Figura 7: Esquema del sistema de control de versiones

El desarrollo con control de versiones aporta las siguientes ventajas:

- Seguimiento del estado del programa.
- Retroceder a estados anteriores a modo de copia de seguridad.
- División de las tareas de programación.

6.2. Desarrollo del software

6.2.1. Diseño de la interfaz

Por una parte existe código estático diseñado con la herramienta *Qt Designer*, este código será procesado por un script con la misión de generar ficheros de interfaz gráfica con extensión *Python* a partir de ficheros con extensión *QML* generados con el editor. El código de salida permitirá dibujar la interfaz en *Python*.

La figura 8 esquematiza el proceso de generación de los ficheros de interfaz gráfica.

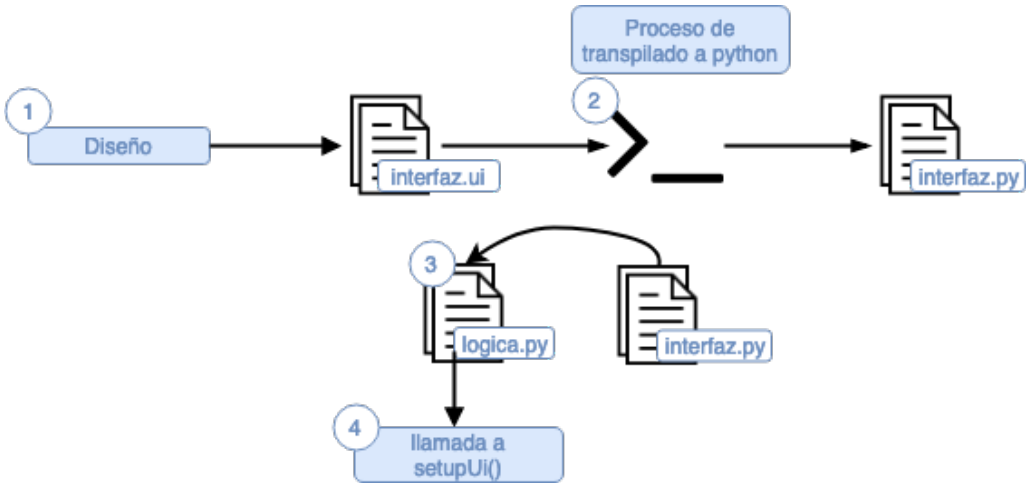


Figura 8: Proceso de creación de interfaces

6.2.1.1. Ficheros estáticos QML y Python

Para la generación de las interfaces gráficas se usó el programa *QtDesigner*.

El programa *QtDesigner* permite crear interfaces gráficas arrastrando elementos. Estos elementos son los encargados de interactuar con los usuarios, mostrar información u organizar la ventana. La documentación oficial[3] se refiere a ellos como *widgets*.

La figura 9 muestra una de las ventanas generadas con este editor y los *widgets* que contiene para el diseño de una de las interfaces gráficas de proyecto.

Los *widgets* empleados son:

- Texto para mostrar información
- Campos para la entrada de valores numéricos por parte del usuario
- Zona de visualización de gráficos para mostrar figuras
- Botones para la interacción con el usuario.
- Reglas de alineamiento vertical y horizontal

[3] https://doc.qt.io/qt-5/qtdesigner-manual.html

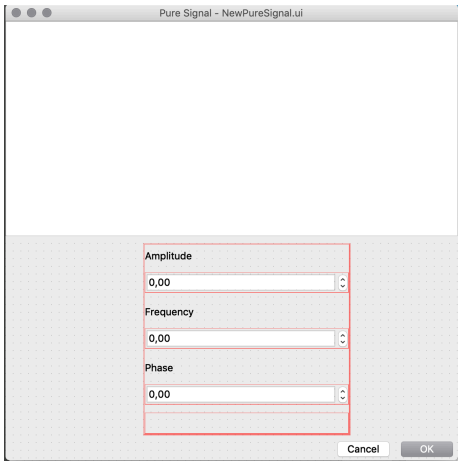


Figura 9: Diálogo creado con QtCreator

La documentación oficial también denomina *widget* al conjunto de *widgets* dentro de una ventana. Por lo tanto a la ventana de la figura 9 también se le denomina *widget*.

Dividiremos estos *widgets* en dos grandes grupos separados por el tratamiento de los datos que realizan:

- **Diálogos:** Encargados de recoger datos de los usuarios, los diálogos no procesan los datos solo generan una previsualización para una mejor experiencia de usuario y sirven de pasarela de datos hacia los siguientes *widgets*.
- **Funciones:** usando los datos de los diálogos realizan las funciones de gráfica, reproducción y grabación de audio o salvado de datos entre otras.

Los ficheros de diseño de interfaz gráfica generados para este proyecto fueron los siguientes:

- Main.ui
- NewPureDialog.ui
- NewSawtoothDialog.ui
- NewSquareDialog.ui
- HarmonicSynthesisDialog.ui
- PeriodicHarmonicSynthesisDialog.ui
- RecordDialog.ui
- NewNoiseDialog.ui
- BaseDialog.ui
- GraphicWidget.ui
- GraphicWidgetFFT.ui

Existe un fichero por cada módulo descrito en el alcance del proyecto.

Por cada uno de estos ficheros existe un fichero con extensión *Python* generados con ayuda de un *script* que se muestra a continuación.

```
# script encargado de realizar la transpilación de un fichero ui
# a un fichero python
pyuic5 src/ui/dialogs/main.ui -o src/python/dialogs/main.py
```

Adicionalmente si el entorno de desarrollo detecta algún cambio en uno de los ficheros de interfaz gráfica *.ui* el *script* se volverá a ejecutar

De esta manera se consigue automatizar una parte del trabajo ya que no es necesario ejecutar a mano el *script* cada vez que sea necesario retranspilar una interfaz.

Una vez convertidos, estos ficheros de carácter estático (ya que aún no albergan lógica) contienen una clase *Python* con 2 métodos usados para dibujar la interfaz, estos métodos son:

1. **setupUi:** Dentro de este método se encuentran definidos todos los *widgets* generados en el programa *QtDesigner* con información acerca de su posición, estado por defecto y relación con otros *widgets*. La relación con otros *widgets* define la manera en la que estos se intercambian información de su estado actual y la documentación se refiere a ella como relaciones *signal-slot*.

   Además el método *setupUi* es el encargado de generar la ventana posicionando todos los *widgets* en ella y de dibujarlos.

2. **_retranslateUi:** Se encarga modificar el aspecto de los *widgets* pasando de un aspecto por defecto y sin ninguna personalización a un aspecto que hemos definido en el editor cambiado el tamaño, color o el tipo de tipografía. Este método es llamado dentro de en la linea final de *setupUi*.

A continuación se muestran unos fragmentos de ambas funciones, donde se observan las características descritas anteriormente para el caso concreto de la ventana principal del programa.

```
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(433, 207)
        ......
        self.retranslateUi(MainWindow)


 def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.label.setText(_translate("MainWindow", "Signal Visualizer))
        ......
```

6.2.2. Codificación de los módulos funcionales

La implementación en código de los módulos de hace dentro de ficheros de lógica.

Para generar un fichero que contendrá lógica se siguen los pasos 3 y 4 vistos en la figura 8.

3. Se crea un nuevo fichero con nombre *nombre_del _fichero _logic.py*, el *logic* final indica que es un fichero que únicamente contiene lógica.
4. Se declara una clase en *Python* y se hace que herede los métodos[4] de la clase del fichero generado anteriormente con la ayuda del *script* de transpilación. Se invoca al constructor padre de la clase heredada. De esta manera se accede a los métodos y atributos del fichero de interfaz desde el fichero de lógica.

La implementación en código del ultimo paso se puede ver a continuación.

```
# se importa el diálogo
 from .NewPureSignal import Ui_PureSginalDialog
 #heredamos los metodos y atributos
 class Ui_NewPureSignalDialogLogic(Ui_PureSginalDialog):
     def __init__(self): #constructor padre
     ...................
```

La llamada al constructor padre ejecutará las funciones **setupUi** y **_retranslateUi** que son necesarias para la configuración inicial de la interfaz.

Este proceso permite que exista una segmentación bien definida entre la interfaz y la lógica.

Los ficheros de lógica generados para este proyecto fueron:

- MainLogic.py
- NewPureDialogLogic.py
- NewSawtoothDialogLogic.py
- NewSquareDialogLogic.py
- HarmonicSynthesisDialogLogic.py
- PeriodicHarmonicSynthesisDialogLogic.py
- RecordDialogLogic.py
- LoadAudioFile.py
- NewNoiseDialogLogic.py
- BaseDialogLogic.py
- GraphicWidgetLogic.py
- GraphicWidgetFFTLogic.py

[4] La herencia es una propiedad de los lenguajes de programación orientados a objetos que permite adquirir las características del objeto heredado

6.2.2.1. Módulos codificados

Finalmente las funcionalidades implementadas en el software son las listadas a continuación.

1. Generación de sinusoidales puras
2. Generación de señales periódicas conocidas
3. Síntesis armónica
4. Generación de ruido blanco
5. Reproducción de señales generadas
6. Grabación reproducción y salvado a fichero de audio de señales capturadas por un micrófono
7. Sistema de visualización extendido de señales

En la tabla 6 se muestra qué fichero *Python* realiza la lógica de cada uno de los módulos codificados en el software.

Módulo	Lógica
Generación de señales puras	new_pure_dialog_logic.py
Generación de señales periódicas conocidas	new_sawtooth_dialog_logic.py new_square_dialog_logic.py
Síntesis armónica	harmonic_synthesis_dialog_logic.py
Generación de ruido blanco	new_noise-dialog-logic.py
Reproducción de las señales generadas	graphic_widget_logic.py
Grabación, reproducción y salvado a fichero de audio	record_dialog_logic.py load_audio_file.py
Sistema de visualización genérico	graphic_widget_logic.py graphic_widget_fftlogic.py

Tabla 6: Tabla de funcionalidades codificadas

6.3. Compilación y distribución

La fase de compilación y distribución permite al *software* ejecutarse sin depender del interprete de *Python* y con una configuración especifica para el sistema operativo que ejecuta dicho *software*.

La compilación de software es un proceso por el cual a partir del código fuente se genera un ejecutable que puede ser transportado a plataformas del mismo sistema operativo y arquitectura de procesador.

La distribución hace referencia a una configuración especifica para el sistema operativo pudiendo dotar al software de funcionalidades extra, para este proyecto esas funcionalidades son las de generar un instalable que simplifique la instalación y desinstalación del software, permita llevar un registro al sistema operativo sobre el programa y realizar una firma digital que verifique que el software no es malicioso.

En la figura 10 se muestran los pasos necesarios para generar un instalador firmado, este proceso debe realizarse en la maquina que ejecute el sistema operativo para el que se desee distribuir el software.

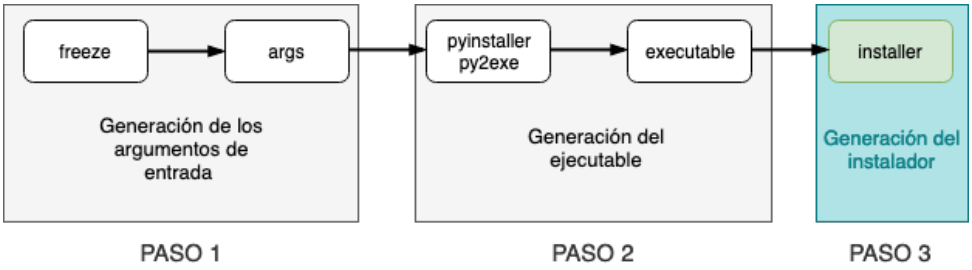


Figura 10: Pasos para la realización de un instalador

Los pasos necesarios son para realizar la tarea de compilación y distribución son:

1. Congelación de dependencias y ejecución de *scripts* para generar un ejecutable.
2. Generación de un ejecutable.
3. Generación de un instalador firmado.

6.3.1. Congelación de dependencias

La congelación de dependencias o *freeze* sirve para capturar qué paquetería de *Python* es requerida para el desarrollo del software.

El proceso de congelación añade la ubicación o ruta de estos paquetes dentro de proyecto a una serie de variables para su uso en el segundo paso, bajo la variable *args* que se observa en el código.

La variable *args* es un diccionario que contiene la ubicación de toda la paquetería y módulos que requiere el software y que por lo tanto será compilada.

El hecho de usar un entorno virtual de *Python* hace que solo se capture aquella paquetería que pertenece a dicho entorno, sin incluir paquetería instalada de manera global que pertenezca por ejemplo a otro desarrollo diferente. Esto se traduce en menos problemas de compilación y un ejecutable final mas liviano

Adicionalmente *fman* cuenta con la opción de especificar que los paquetes que se capturan solo son los que el software necesita para funcionar con lo que se excluye todo tipo de paquetería que tiene fines de desarrollo. Para este proyecto las dependencias excluidas del proceso de congelación son *fman*, *Py2exe* y *pip*.

Las dependencias incluidas en el software se listaron en la sección 6.1.2

Con la ubicación de todas estas dependencias ya en la variable *args*, se añade la ruta del directorio *Python* visto en la sección 6.1.1 que contiene el código fuente de la aplicación estructurada en módulos *Python*.

La congelación de dependencias, varía en función del sistema operativo usado por lo que es necesario hacerlo en cada sistema operativo objetivo.

El comando usado para congelar las dependencias es el siguiente: `fbs freze`

En la función *freeze_mac* muestra los pasos que realiza el comando:

1. Carga el icono de la aplicación.
2. Mira si el modo *debug* esta activo
3. Carga las dependencias dentro de la variable *args*
4. Ejecuta *PyInstaller*

```
#freezeado de dependencias para macOS
def freeze_mac(debug=False):
    if not exists(path('target/Icon.icns')):
        #generacion de iconos
        _generate_iconset()
        run(['iconutil', '-c', 'icns', path('target/Icon.iconset')], check=True)
    args = []
    #modo debug
    if not (debug or SETTINGS['show_console_window']):
        #eleccion del modo ventana
        args.append('--windowed')
    args.extend(['--icon', path('target/Icon.icns')])
    bundle_identifier = SETTINGS['mac_bundle_identifier']
    if bundle_identifier:
        args.extend([
            '--osx-bundle-identifier', bundle_identifier
        ])
    #ejecutamos pyinstaller con los argumentos deseados
    run_pyinstaller(args, debug)
    _remove_unwanted_pyinstaller_files()
    _fix_sparkle_delta_updates()
    _generate_resources()
```

Después de la ejecución se generará un ejecutable funcional a no ser que se den errores de compilación. Para resolver estos errores se puede solicitar un *log* de errores con el *flag debug* cuando se realiza la congelación de dependencias. El ejecutable será generado en el directorio *target*, la ubicación de este directorio puede ser definida dentro del fichero de configuración *build* mencionado anteriormente y por defecto reside en el directorio raíz del proyecto.

6.3.2. Generación del instalable

Ultima parte del proceso de distribución. Para la generación usaremos una utilidad diferente dependiendo del sistema operativo.

1. **yoursway-create-dmg:** Utilidad para crear ejecutables en un entorno *macOS*.
2. **fpm:** Utilidad para crear ejecutables en un entorno *Linux*.
3. **NSIS:** Utilidad para crear instaladores en un entorno *Windows*.

Aunque *fman* no dispone en su interior del programa *NSIS*, da la posibilidad de establecer una variable de entorno en *Windows* que permita acceder a *fman* al software *NSIS*. Esta variable de entorno no es más que una ruta donde se encuentran todos los ficheros de *NSIS*.

Antes de proceder a crear el instalador es necesario que la congelación de dependencias vista en el apartado 6.3.1 haya finalizado con éxito.

El comando a invocar es el siguiente.

```
#detectar entorno y generar instalador
fbs installer
```

El instalable se creará dentro del fichero *target* mencionado anteriormente junto con el ejecutable.

Una vez se ha generado el instalador, el software esta preparado para ser distribuido a la plataforma que se desee.

6.4. Resultados

Una vez codificadas las funcionalidades que se definieron en el alcance se comentará el resultado mostrando capturas de cada una de las ventanas generadas y discutiendo la funcionalidad conseguida.

Ventana principal: Concentra todas las funcionalidades en una barra de tareas en la parte superior de la que se despliegan las opciones.

Lleva a cabo tareas de gestión de todas las ventanas abiertas y se encarga de poder generar mas de un diálogo de mismo tipo acumulando estas en una pila.

Las figuras 11 y 12 muestran los campos **Generate Input/Output y Analize** en la parte superior de la ventana para dos plataformas diferentes.

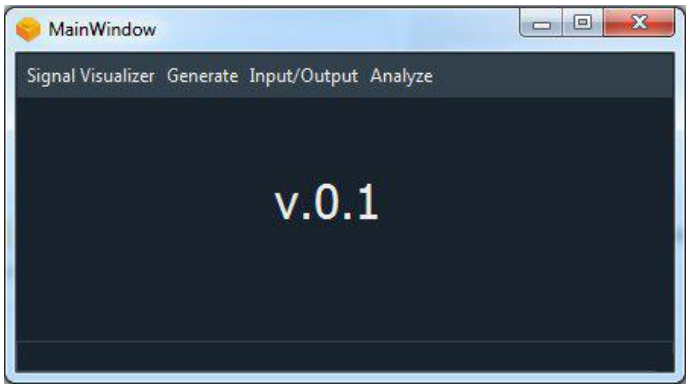


Figura 11: Ventana principal en plataforma Windows

Figura 12: Ventana principal en plataforma Mac

6.4.1. Apartado de generación.

1. **Generación de señales puras:** Se codificó un diálogo con los siguientes *widgets*.
   - Campo para introducir amplitud de la señal
   - Campo para introducir frecuencia de la señal
   - Campo para introducir fase inicial de la señal
   - Gráfica de previsualización

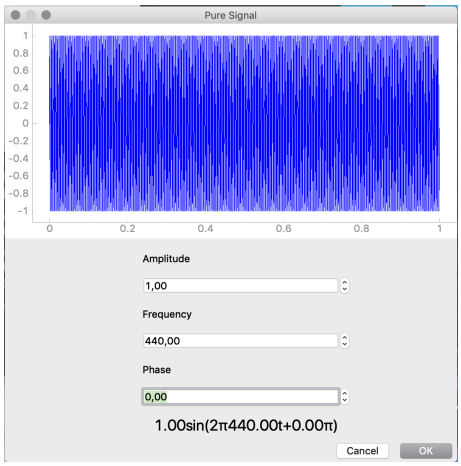


Figura 13: Diálogo para la creación de una nueva señal pura.

La figura 13 muestra la disposición de cada uno de los *widgets*.

Gracias a este diálogo se podrá generar una senoide y previsualizar su aspecto. El diálogo es capaz de ejecutarse múltiples veces por lo que se pueden generar un número considerable de diálogos.

2. **Generación señales periódicas conocidas:** Encargada de la generación de señales diente de sierra y cuadrada. Los *widgets* que componen el diálogo son:

   - Campo para introducir la frecuencia fundamental
   - Campo para introducir la amplitud
   - Gráfica de previsualización

   La figura 14 muestra una ventana para la generación para una señal cuadrada.

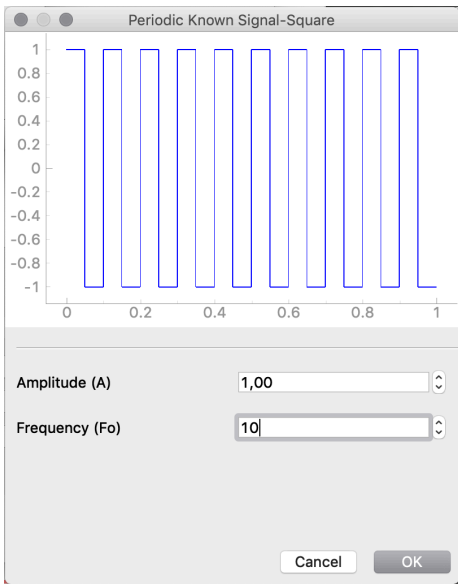


Figura 14: Ventana para la generación de una señal cuadrada.

La funcionalidad es idéntica a la generación de señales puras salvo por la diferencia de parámetros de entrada. También permite ejecutar varias instancias a la vez.

3. **Síntesis armónica libre:** Encargada de generar señales con múltiples armónicos.

   Los *widgets* que componen el diálogo son:

- Campo numérico de amplitud
- Cuadro de texto para introducir la frecuencia
- Gráfica de previsualización

La figura 15 muestra cómo es posible introducir un máximo de diez armónicos.

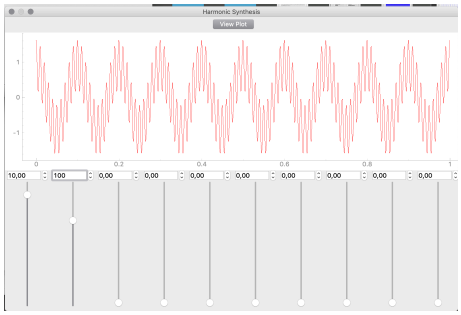


Figura 15: Ventana para la síntesis armónica.

Gracias a este diálogo podremos generar una señal compuesta por varias senoides. El diseño mas amplio permite que los manejadores sean cómodos de usar.

Introducir una frecuencia superior a 10^9 puede congelar la gráfica durante unos segundos antes de mostrarla.

4. **Generación de ruido blanco:** Para este sencillo caso se pasa directamente a la ventana de visualización extendida como se ve en la figura 16 ya que no es necesario introducir ningún parámetro.

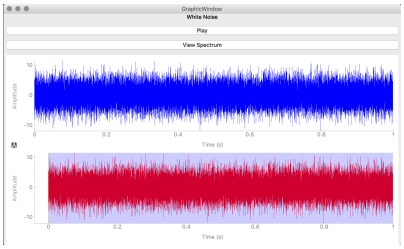


Figura 16: Generación de ruido blanco.

6.4.1.1. Funciones de entrada y salida

1. **Grabación de audio:** Permite la grabación y previsualización de audio capturado por un micrófono.

   Los *widgets* que lo componen son:

   - Botón de grabación
   - Botón de reproducción
   - Botón de salvado a disco
   - Indicador de grabación en curso
   - Entrada de duración de grabación
   - Entrada de frecuencia de muestreo para la grabación

   La figura 17 muestra la disposición de estos *widgets*.

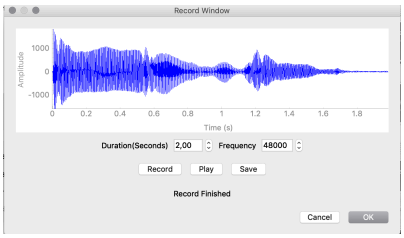


Figura 17: Ventana para grabación.

2. **Importación de un fichero de audio:** Utilidad que permite la importación de audio previamente grabado para su inspección en una ventana de visualizado extendido.

   Los *widgets* que componen el diálogo son:

   - Botón de navegación al fichero
   - Botón de reproducción

6.4.2. Sistema de visualizado extendido:

Como elemento complementario a todos los diálogos, en la figura 18 se aprecian dos botones, el primero de ellos **cancel** permite cancelar el diálogo y cerrar la ventana, el botón **ok** permite pasar al sistema de visualizado extendido.

Figura 18: Botones de diálogo cancel y ok.

Esta ventana esta disponible tanto en el análisis de una señal en el dominio del tiempo como en el dominio de la frecuencia como se observa en las figuras 19 y 20.

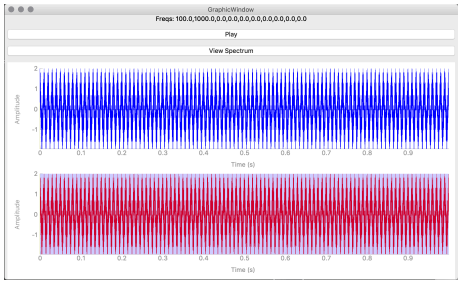


Figura 19: Sistema de visualizado extendido en dominio temporal.

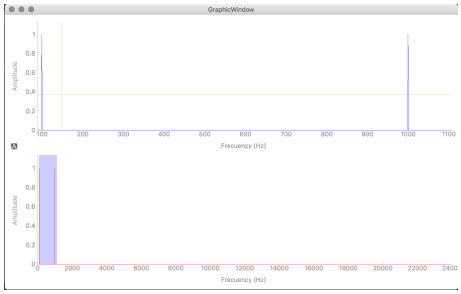


Figura 20: Sistema de visualizado extendido en dominio frecuencial.

Esta ventana ofrece utilidades extra a la hora de realizar el análisis de una señal previamente generada.

Sus características principales son:

1. Doble ventana con una opción de zoom en una de ellas.
   En la figura 20 se observa el sistema de zoom
2. Reproducción de la señal analizada en el dominio temporal
3. Acceso a opciones extra usando el clic de ratón secundario sobre la gráfica superior

El modulo correspondiente a *analize* no se realizo por falta de tiempo.

6.4.3. Ejecutables

Gracias al sistema de generación de ejecutables se pudieron generar instaladores para las principales plataformas.

Figura 21: Registro de aplicaciones de Windows.

El la figura 21 se muestra como gracias a los procesos generados en la fase de compilación y distribución el software esta presente en el registro de programas de *Windows* bajo el nombre de *Signal Visualizer* mostrando su autor y tamaño en disco, de esta manera la desinstalación o reinstalación del mismo es mas sencilla.

7. Descripción de tareas

En esta sección se muestran las tareas despeñadas a lo largo de todo el proyecto y se describen las etapas necesarias para su realización. Se detallan diversos factores como las personas involucradas y los paquetes de trabajo con sus respectivos hitos establecidos. Esta información se complementa presentando un diagrama de *Gantt* con el fin de visualizar la planificación y donde se especifican otros aspectos como la duración de las tareas y la relación entre las mismas.

A continuación se detallan las tareas realizadas:

- **Investigación**
  - **Investigación sobre software de audio:** Experimentación con diversos programas de audio con el fin de analizar como su software realiza sus funciones, como presentan datos al usuario y bajo que tecnología están construidos.
  - **Investigación sobre codificación de aplicaciones:** Investigación sobre diferentes librerías software para codificar aplicaciones de escritorio.
  - **Codificación de ejemplos:** Con los datos de la investigación se realizaron pruebas para tener una idea sólida para el análisis de alternativas.
  - **Definición de los objetivos:** Establecimiento de los puntos que se realizarán en el proyecto.
  - **Definición del alcance:** Establecer los limites de trabajo para cubrir el tiempo esperado.
- **Diseño de la solución**
  - **Selección de las herramientas de desarrollo:** Realizando un análisis de alternativas se seleccionaron posibles utilidades software que podían ser útiles en el proyecto.
  - **Integración y codificación de las herramientas de desarrollo:** Basándonos en los ejemplos codificados anteriormente se codificaron utilidades software para agilizar el desarrollo.
  - **Codificación del software:** implementación en código de los módulos especificados en el alcance.
  - **Corrección de errores:** Tanto en las utilidades como en las funcionalidades.

- **Integración del sistema de compilación:** El proyecto requirió de una reintegración de las utilidades de compilación dado su tamaño.

- **Corrección de errores generales**

- **Pruebas y Mejoras**

  - **Prueba de las funcionalidades del programa:** A medida que se implementan nuevas funciones estas son inspeccionadas para comprobar si cumplen los objetivos fijados.

  - **Mejora de las herramientas de desarrollo:** A medida que crece el programa es necesario actualizar los *scripts* que hacen posible la compilación o el seguimiento de fallos.

- **Documentación:** Redacción del documento final y del manual de desarrollo.

- **Reuniones de coordinación:** Reuniones semanales o bisemanales con la directora del trabajo y complementariamente con personal del centro de estudios musicales. El horario se mantuvo de esta manera salvo en periodo de exámenes o indisponibilidad que no se realizaron. Estas reuniones tenían los siguientes propósitos:

  - **Valoración y orientación:** por parte del tutor del estado del programa y la propuesta de mejoras para este, tanto en el apartado gráfico como el funcional.

7.1. Diagrama de Gantt

En la figura 22 se presenta el diagrama de Grantt de las tareas y subtareas requeridas para la realización del proyecto.

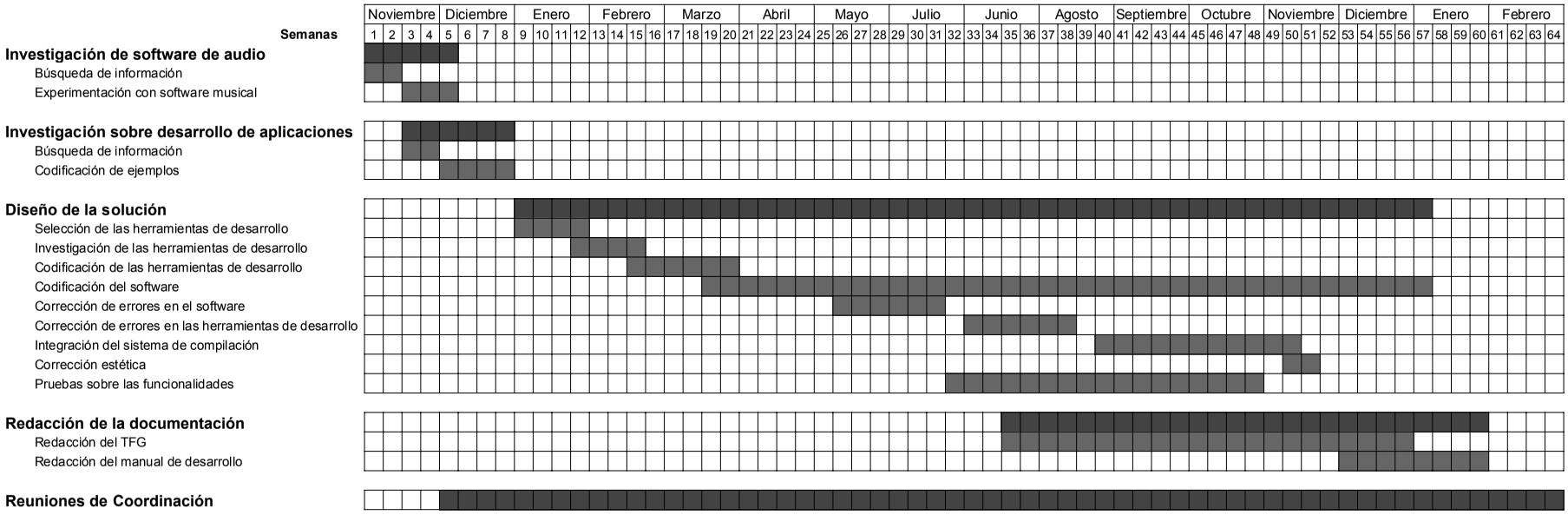


Figura 22: Diagrama de Gantt del proyecto.

8. Descripción del presupuesto

A continuación de presenta el resumen de los costes que han sido necesarios para la realización de este proyecto. Dichos costes se han dividido en recursos humanos y recursos técnicos.

8.1. Recursos humanos

El coste de las horas dedicadas al proyecto se ha estimado según lo que se cobraría por un trabajo similar, para el alumno se han aproximado a horas de ingeniero junior y para el director de proyecto como ingeniera senior.

Cargo	Coste (€/h)	Dedicación (horas)	Coste total (€)
Director de proyecto	55	60	3.300
Ingeniero Junior	20	380	6.800
Total			**10.100**

Tabla 7: Desgloses de gastos derivados de recursos humanos.

8.2. Recursos técnicos

Para el proyecto se han necesitado los siguientes equipos electrónicos:

- **Ordenador personal:** Para la codificación del proyecto realización de la documentación y pruebas de compilación en entorno macOS.
- **Portátil con sistema operativo Windows:** Para la integración del proyecto en un entorno Windows y pruebas de compilación en Windows. No hubo que adquirir una licencia ya que el equipo contaba con una.
- **Raspberry Pi:** Para las pruebas de compilación en Linux.

Para la realización del proyecto no se ha usado ninguna licencia de software de pago y todas las herramientas utilizadas son de software libre y pueden adquirirse sin ningún coste.

8.3. Gastos

Se entiende por gastos aquellos recursos que no puede ser reutilizados en ningún otro proyecto. Este trabajo no hace uso de licencias de pago en ningún equipo, por lo que los gastos directos son los siguientes:

Concepto	Precio adquisición (€)	Vida útil (años)	Utilización (días)	Coste diario (€/día)	Coste total (€)
Ordenador personal	1500	6	200	0,684	136,8
Ordenador Windows	1000	6	15	0,456	6,84
Raspberry Pi	45	6	5	0.020	0.1
Total					**143,78**

Tabla 8: Material amortizable utilizado durante el proyecto

No existen gastos **no amortizables** en el proyecto. La conexión a internet y electricidad se imputarán como gastos adicionales en la partida de gastos indirectos.

8.3.1. Resumen del presupuesto

Concepto	Coste (€)
Recursos Humanos	10.100
Amortizaciones	143,78
Total costes directos	10.243,78
Costes indirectos (Estimación)	350
Total (sin I.V.A)	**10.593,78**
Total (con I.V.A)	**12.818,47**

Tabla 9: Resumen del presupuesto

El coste total del proyecto es de 12.818,74€ (I.V.A incluido)

9. Conclusiones

9.1. Codificación de los módulos

Los apartados referentes al primer módulo completados en su totalidad fueron:

- Senoides puras y visualización de parámetros básicos
- Composición de señales básicas periódicas y no periódicas

El apartado referente al análisis de Fourier no pudo completarse, debido a que la integración de la librería *sounddevice* con el sistema operativo *Windows* generó un error que requirió tiempo para identificar.

9.2. Generación de ejecutables

El método de generación de ejecutables codificado permite generar instaladores y ejecutables multiplataforma tanto para *macOS*, *Linux* y *Windows*.

Su uso facilita el desarrollo, aunque el tiempo invertido para codificarlo supero por mas del doble las horas estimadas en la planificación del proyecto. Esto se debe a que a pesar de que *Python* es multiplaforma se necesitan herramientas diferentes en función del sistema operativo para el que se genera el ejecutable, como *NSIS* en el caso de *Windows* y *yours-way-create-dmg* para *macOS* con lo que fue necesario aprender a usar cada una.

9.3. Uso de la aplicación

La herramienta puede usarse para la visualización, composición y salvado de señales básicas por parte de estudiantes de acústica musical, para ayudarles entender acústica realizando simulaciones con dicho software.

Bibliografía

[1] Brian Jones, David Beazley
Python cookbook, third edition.
2016

[2] Mark Summerfield
Rapid GUI Programming with Python and Qt.
2014

[3] B. M. Harwani
Introduction to Python Programming and Developing GUI Applications with Pyqt.
2015

[4] Martin Fitzpatrick
15 Minute Apps
https://github.com/mfitzp/15-minute-apps
2019

[5] James Crook
Audacity.
https://wiki.audacityteam.org/wiki/Audacity_Wiki_Home_Page
2019

[6] Michael Herrmann
Fman Build System
https://github.com/mherrmann/fbs
2018

[7] Max Ogden
Electron fundamentals
https://maxogden.com/electron-fundamentals.html
2018

[8] Kristian Poslek
Building a desktop application with Electron
https://medium.com/developers-writing/building-a-desktop-application-with-electron-204
2018

10. Anexo I: Manual de desarrollo

Guía de desarrollo

IDE para el desarrollo

Integración de Signal Visualizer en el proyecto

Desarrollo de nuevas funcionalidades

- Creación de un fichero de interfaz gráfica (.ui) usando el software Qt Creator
- Transpilación de código qml a código válido python.
- Integración el Menú Principal.
- Ejecución de cuadros de diálogo desde la ventana principal.
- Creación de la ventana Lógica

Signal Visualizer

Mikel Diez Garcia

IDE para el desarrollo

Para el desarrollo de aconseja el uso de Pycharm Community

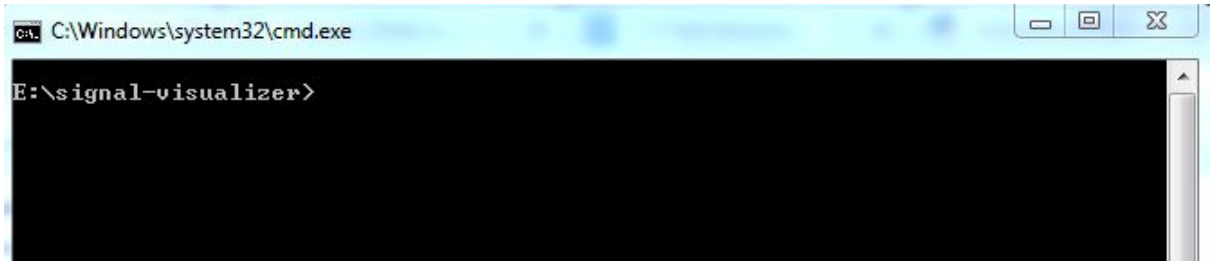
https://www.jetbrains.com/pycharm/

Esta guia da por sentado que se tiene python 3.5 o 3.6 instalado y el asistente de paquetería pip si no es así se puede adquirir aquí

https://www.python.org/

Instalación de fman y creación de un proyecto vacío

1 creamos una carpeta vacía donde se ubicara nuestro proyecto y lo abrimos en el terminal



Una vez abierto se crea un **nuevo entorno virtual de python**.

```
python -m venv venv
```

Se activa

```
call venv\scripts\activate.bat
```

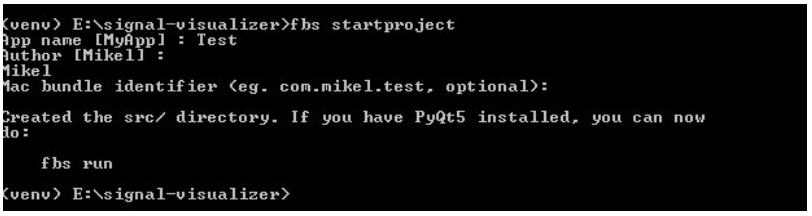
Se instalan las dependencias básicas y su versión

```
pip install fbs PyInstaller==3.4 PyQt5==5.9.2
```

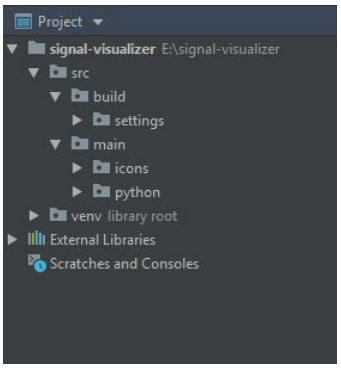
.
Se crea un nuevo **proyecto en blanco.**

```
fbs startproject
```

Se rellena el formulario el **App name es obligatorio “Signal Visualizer”** aunque es modificable posteriormente.



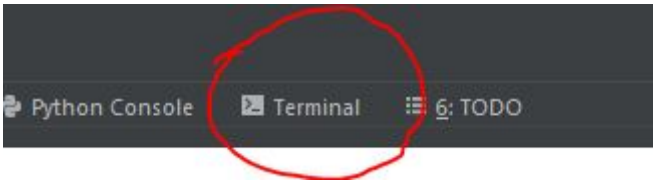
Abrimos el proyecto con el IDE antes mencionado



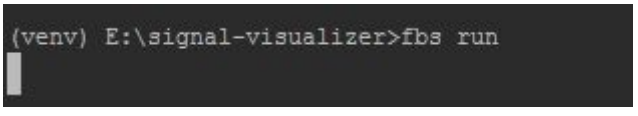
La jerarquía de carpetas es la siguiente

- src/ — Root directory for your source files
  - build/ — Files for the build process
    - settings/ — Build settings:
      - base.json — all platforms
      - (mac.json) — specific to Mac
      - (windows.json) — ...
      - (linux.json) — all Linux distributions
      - (arch.json) — specific to Arch Linux
      - (fedora.json) — ...
      - (ubuntu.json) — ...
      - (release.json) — during a release
  - main/ — Implementation of your app
    - icons/ — Your app's icon.
    - python/ — Python code for your application
    - (resources/) — Data files, images etc.
  - (freeze/) — Files for freezing your app
  - (installer/) — Installer source files
    - (windows/)
    - (mac/)
    - ...
- (requirements/) — Your Python dependencies:
  - (base.txt) — on all platforms
  - (linux.txt) — on all Linux distributions
  - (arch.txt) — on Arch Linux
  - ... — ...

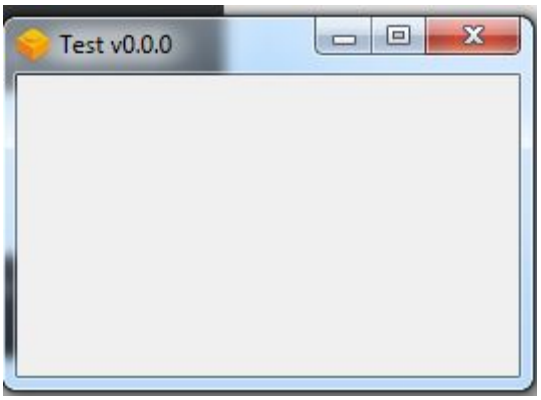
Para comprobar que el proyecto se ha creado correctamente ejecutamos el siguiente comando desde la consola de PyCharm que es accesible desde la parte inferior del IDE




```
fbs run
```



Si el proyecto se realizó correctamente el resultado debería ser una ventana de este estilo



Congelación del proyecto

La congelacion de proyecto permite crear un ejecutable que puede ser distribuido y ejecutado sin necesidad de usar el interprete de python.

Para crear un **ejecutable** se usa el comando.

```
fbs freeze
```

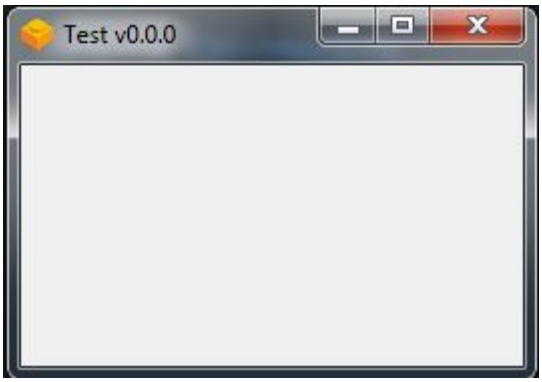
Esto ejecuta todo el proceso automatizado para generar el ejecutable y nos da su ruta para ejecutarlo

```
(venv) E:\signal-visualizer>fbs freeze
Done. You can now run `target\Test\Test.exe`. If that doesn't work,
see https://build-system.fman.io/troubleshooting.

(venv) E:\signal-visualizer>
```

Para ejecutarlo

```
target\Test\Test.exe
```

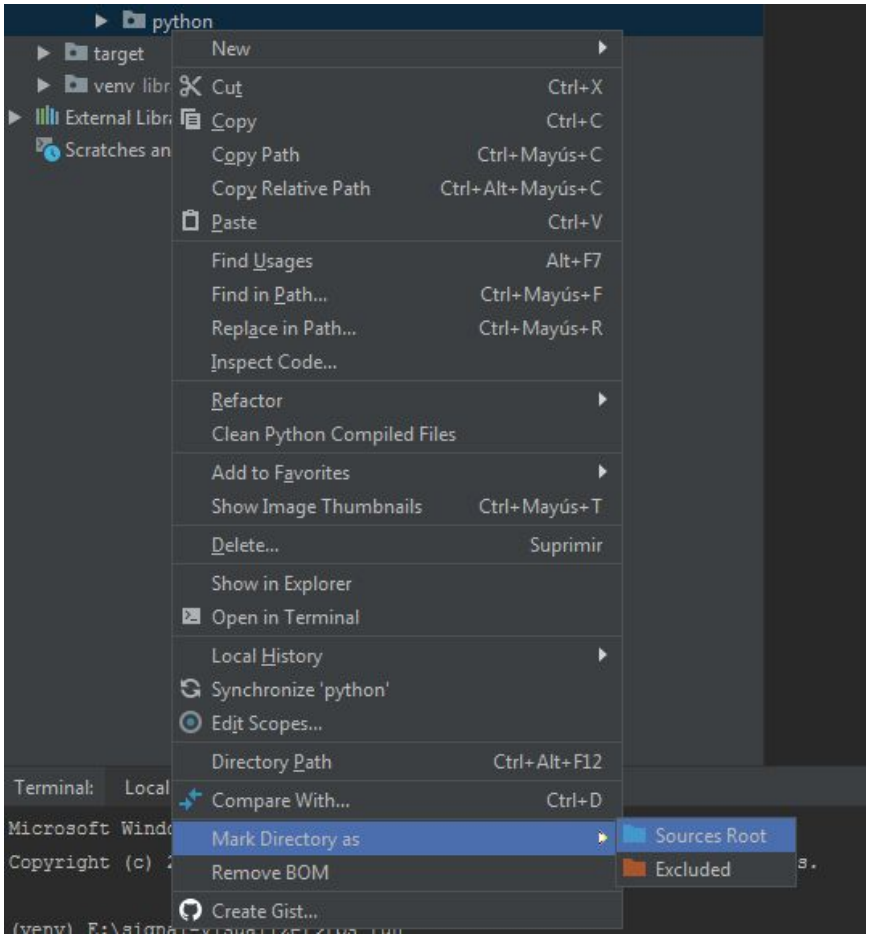
El resultado esperado es:


Test v0.0.0

Integración de Signal Visualizer en el proyecto

Ajuste de ficheros python

Es necesario decir a PyCharm cual es el origen de ficheros fuentes para ello **se marca el fichero python como directorio fuente.**



Esto permite generar **clases de python en otros ficheros e integrarlas en el proyecto**. El color de directorio pasará a ser de color azul.

Este paso es de especial importancia ya que **fbs no sabrá qué dependencias usar para ejecutar con el intérprete y construir el ejecutable.**

Dependencias necesarias para Signal Visualizer y versión.

PyAudio	0.2.11
PyInstaller	3.4
PyQt5	5.9.2
altgraph	0.16.1
boto3	1.9.86
botocore	1.12.86
certifi	2018.11.29
cffi	1.11.5
chardet	3.0.4
docutils	0.14
fbs	0.6.6
future	0.17.1
idna	2.8
jmespath	0.9.3
macholib	1.11
numpy	1.15.4
pefile	2018.8.8
pip	10.0.1
pycparser	2.19
pyqtgraph	0.10.0
python-dateutil	2.7.5
pywin32	224

pywin32-ctypes	0.2.0
pyz	0.4.3
requests	2.21.0
s3transfer	0.1.13
scipy	1.2.0
setuptools	40.7.0
simpleaudio	1.0.2
sip	4.19.8
six	1.12.0
sounddevice	0.3.12
urllib3	1.24.1

Ya están incluidas en el proyecto

PyCharm dispone de un **gestor de dependencias con un entorno gráfico** para la instalación de la paqueteria

La ruta **File → Settings → Project Interpreter** nos muestra la paqueteria actual y nos permite instalar nueva paquetería dandole al **símbolo “+”**

Una vez t**oda la paquetería está instalada** Signal Visualizer es capaz de ejecutarse si problema.

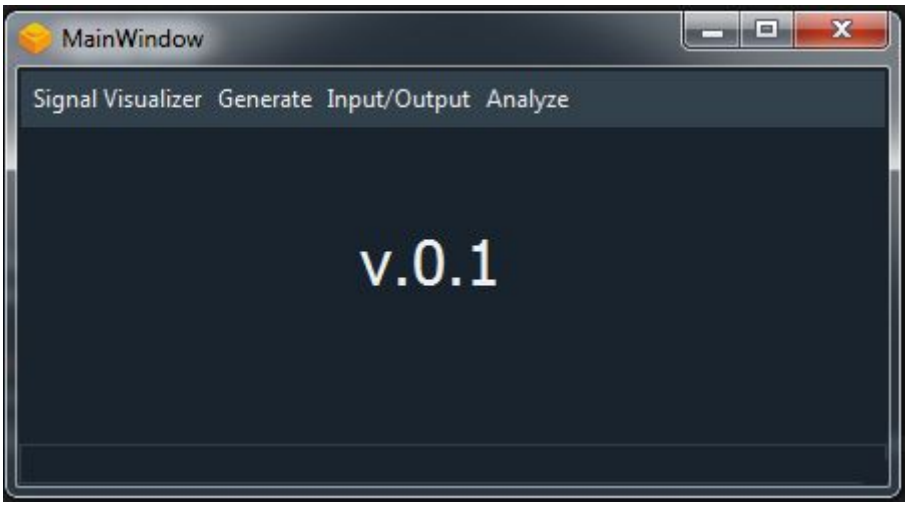
Para se ejecuta

```
fbs run
```

Para generar un ejecutable

```
fbs freeze
```

El programa se ejecuta en pantalla.



En este punto la configuración del entorno es la idónea para el desarrollo

Desarrollo de nuevas funcionalidades

El desarrollo de nuevas funcionalidades comprende de:

1 Creación de un fichero de **interfaz gráfica (.ui) usando el software Qt Creator**

2 **Transpilado de la interfaz gráfica por medio de la utilidad pyuic5** para generar la clase python nueva.

3 Integración con la ventana principal.

4 Implementación de la funcionalidad.

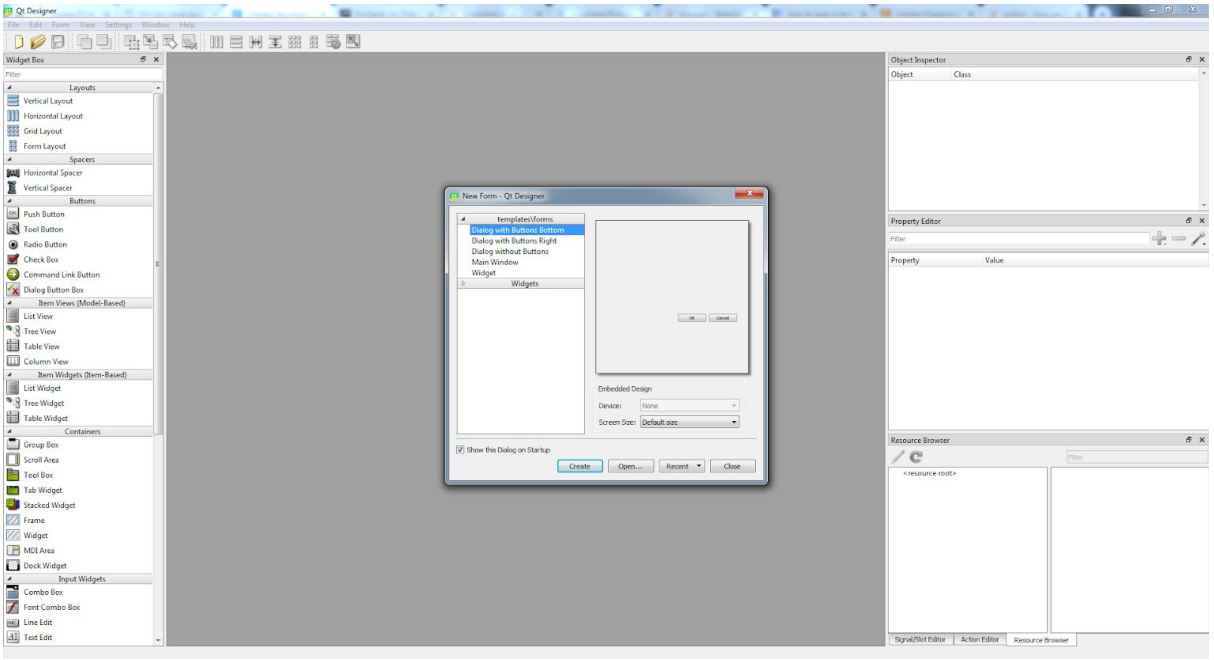
Creación de un fichero de interfaz gráfica (.ui) usando el software Qt Creator

QtCreator es un entorno grafico para el diseño de interfaces gratuito y viene incluido con el paquete PyQt5.

Para su instalación desde una nueva terminal **sin un entorno virtual** ejecutamos

```
pip install pyqt5-tools
```

La utilidad aparecerá en programas instalados bajo el nombre **designer**



Una vez dentro de la utilidad se detalla como crear una nueva interfaz .ui

Para este proyecto se han utilizado 3 tipo de widgets

-Dialog with buttons bottom

-Main Window

-Widget

Aunque el principio de funcionamiento es idéntico entre ellos su diferencia reside en que tanto **Dialog with buttons bottom y Main Window** incluyen una serie de widgets en su interior.

Cada cuadro de diálogo tiene sus características en la documentación de PyQt se puede profundizar sobre esta.

Seleccionamos el **Dialog with buttons bottom** en el menú.

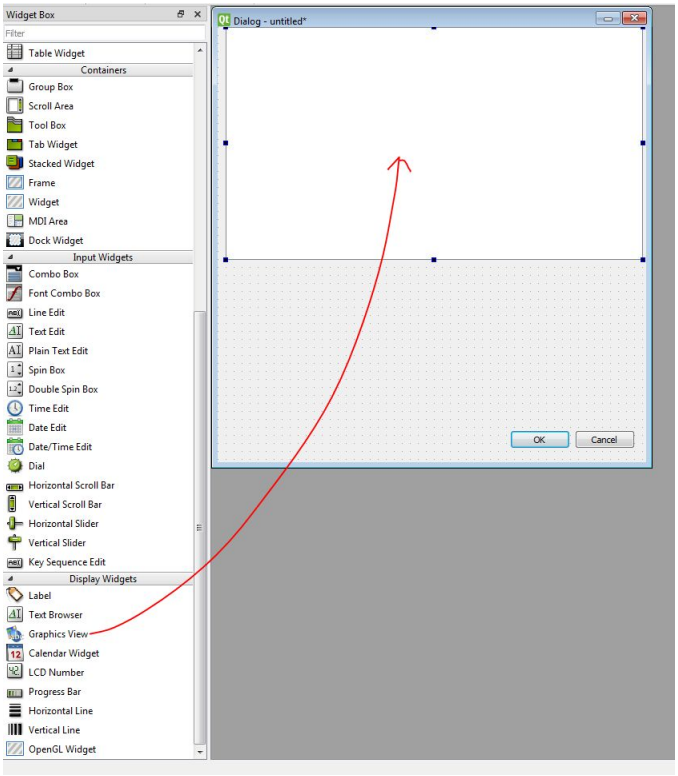
Una vez en el diseñador añadimos los elementos necesarios, para este fin incluiremos la dependencia pyqtgrap usada para incrustar gráficos en los diálogos ya que es la única librería de gráficos externa empleada.

Añadir un widget externo

En este proyecto fue necesario añadir widgets que no se encontraban en la paqueteria por defecto como es el caso de pyqtgraph

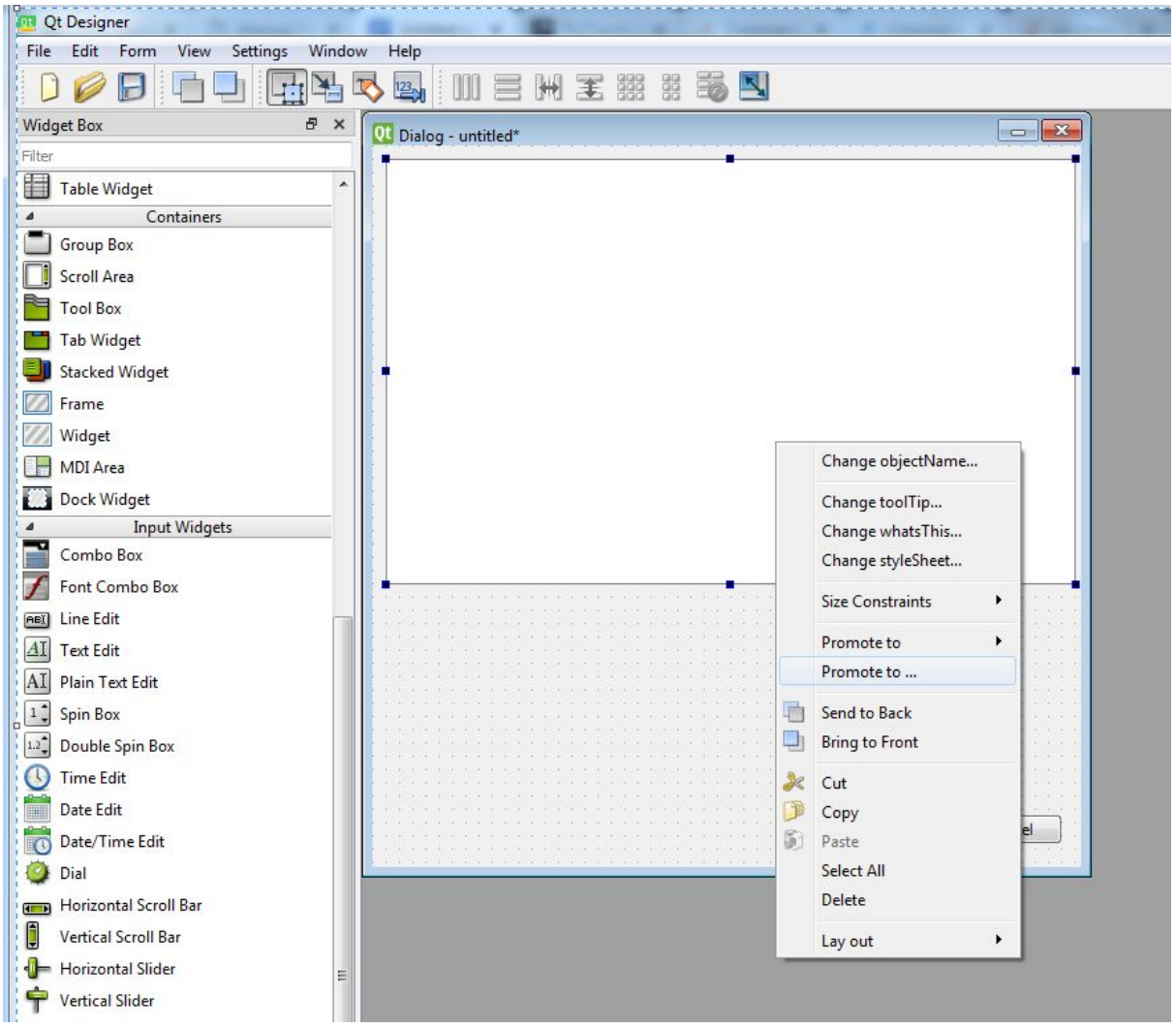
Para añadir un widget que no viene por defecto sobre la utilidad qt seguimos los pasos siguientes.

1 Arrastrar el widget **QGraphicView** desde el editor hasta la ventana seleccionada anteriormente.



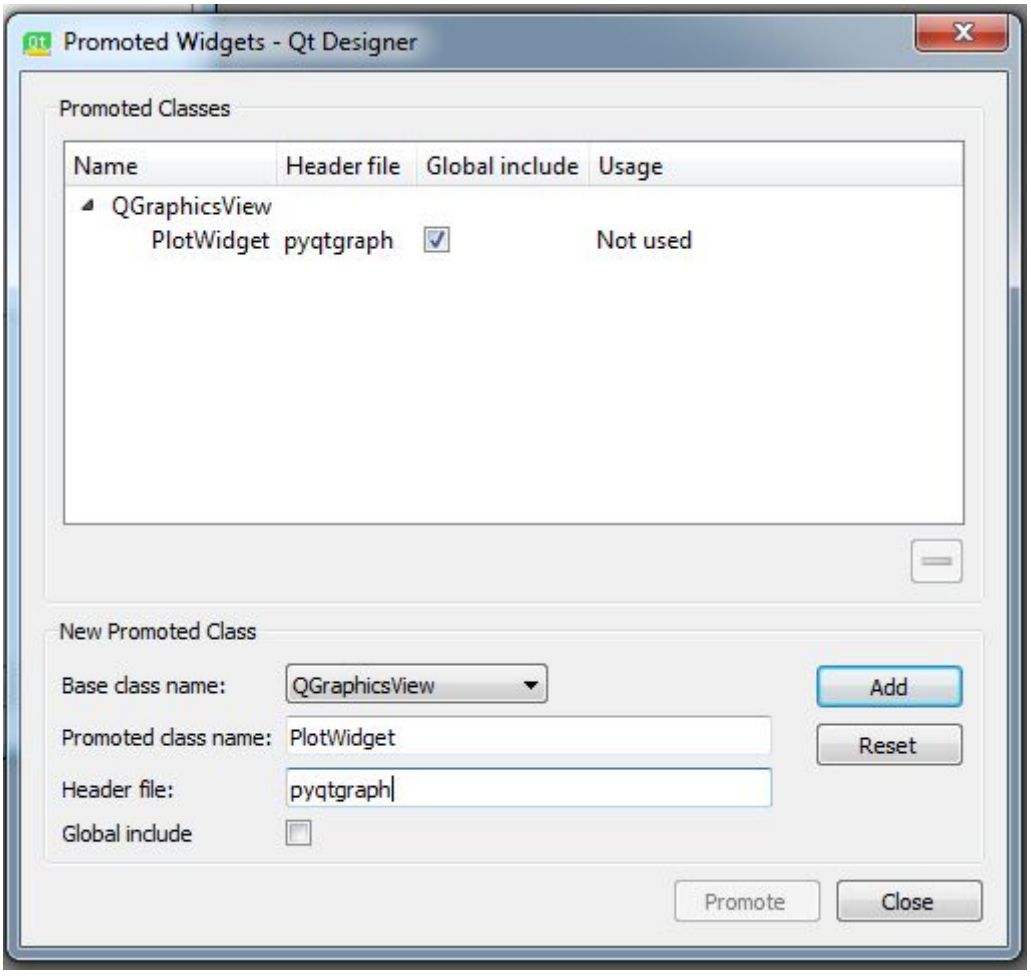
Lo expandimos en función de nuestro diseño

Click derecho sobre el para abrir sus propiedades



seleccionar en **promote to** con esto heredaremos la funcionalidad de la librería de ploteo pyqtgraph instalada antes bajo las dependencias de signal visualizer.

Aparecerá el siguiente cuadro de diálogo

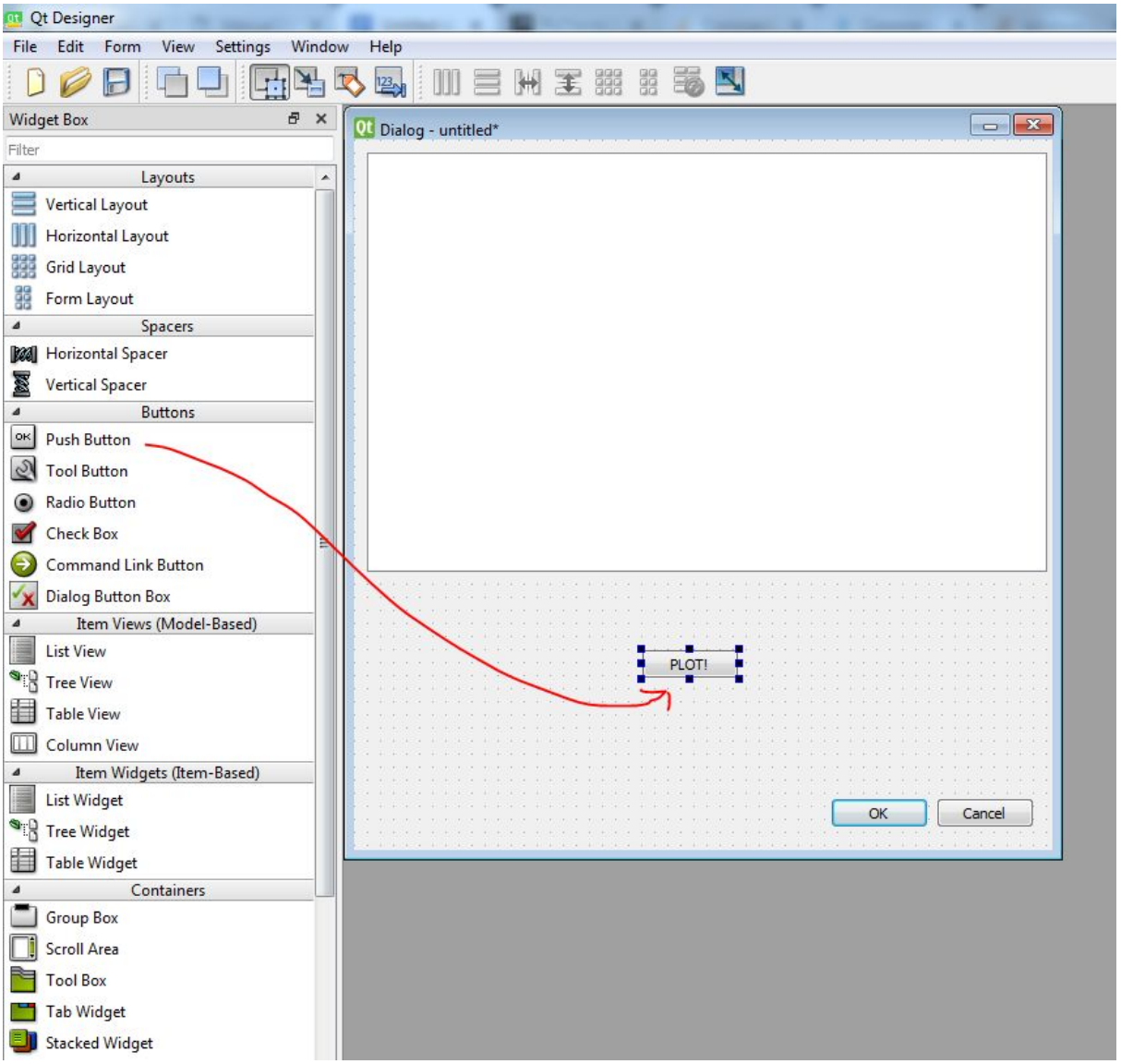


Bajo **promoted class name:** PlotWidget o un nombre significativo para nuestro widget
Header file: pyqtgraph → es importante que este campo sea identico de lo contrario python no encontrara la dependencia.

Finalmente seleccionados **Add** y después **Promote**.

Estado final de la ventana.

Como último paso añadiremos un botón para mostrar la funcionalidad



La generación de interfaces gráficas es muy amplia, este manual solo ofrece ejemplos de los diálogos creados para el proyecto para más profundidad se puede consultar la API de PyQt.

Transpilación de **código qml** a **código válido python.**

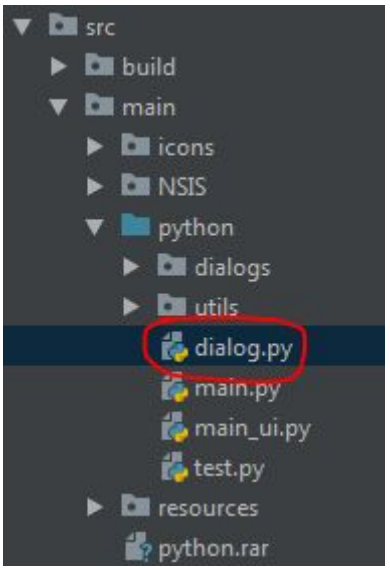
Guardamos nuestro diálogo en el directorio ui en la carpeta de nuestro proyecto (si no existiese la carpeta creamos una) dentro de este se encuentran todas las ventanas generadas para el proyecto.

Una vez salvado el fichero se ejecuta el comando de **transpilación en la consola de PyCharm:**

```
pyuic5 <fichero_fuente>.ui -o <fichero_de_salida>.py
```

```
pyuic5 ui\dialog.ui -o src\main\python\dialog.py
```

Nótese que el fichero python se genera bajo el directorio python.



El fichero puede ser reordenado donde se desee pero siempre debajo del fichero python.

Integración el Menú Principal.

El paso restante es ejecutar este diálogo desde la ventana principal de Signal Visualizer

Los pasos son:

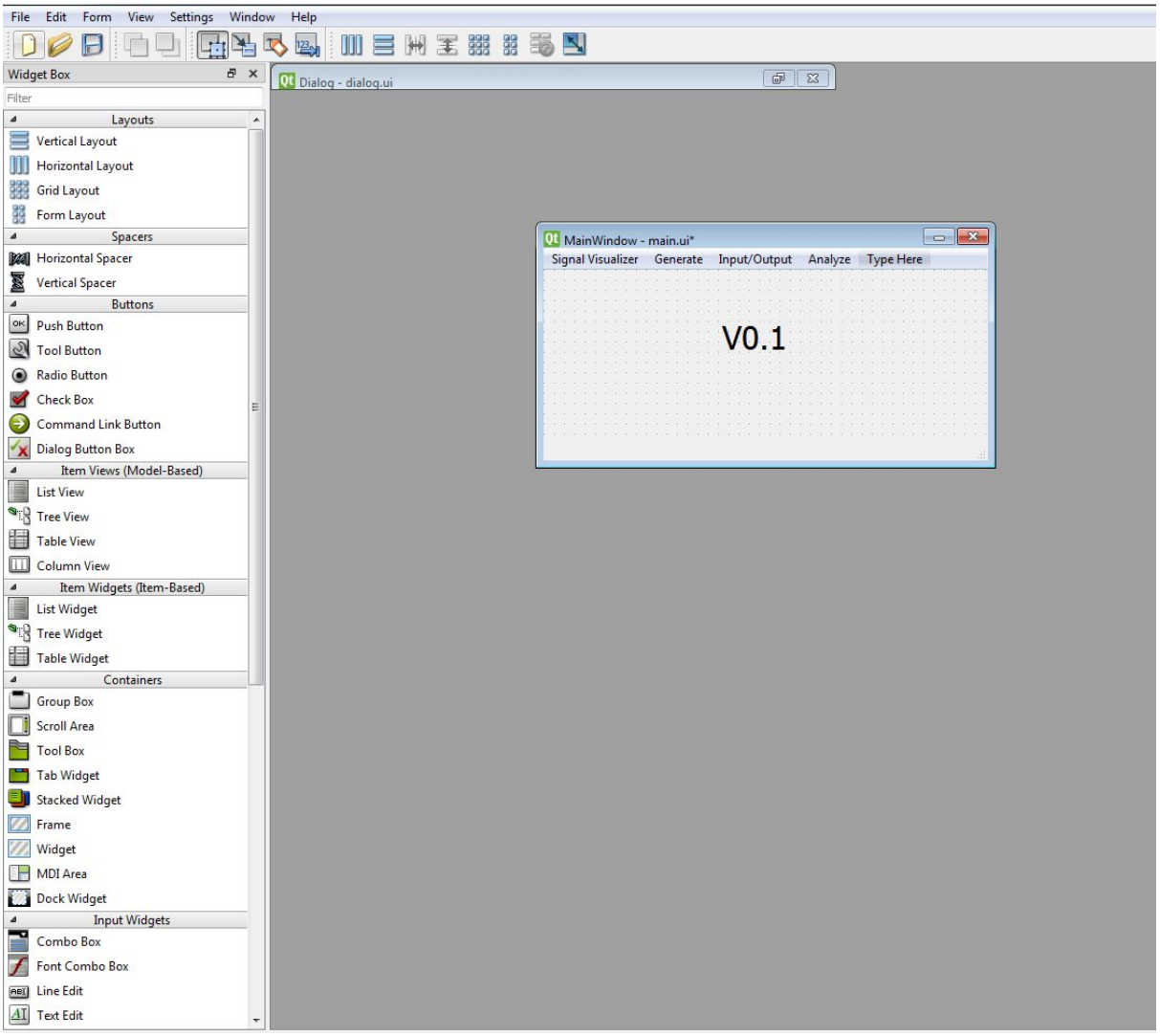
1 Añadir una nueva entrada en la ventana principal

> Usando Qt Designer se añade la función
> Se transpila el codigo

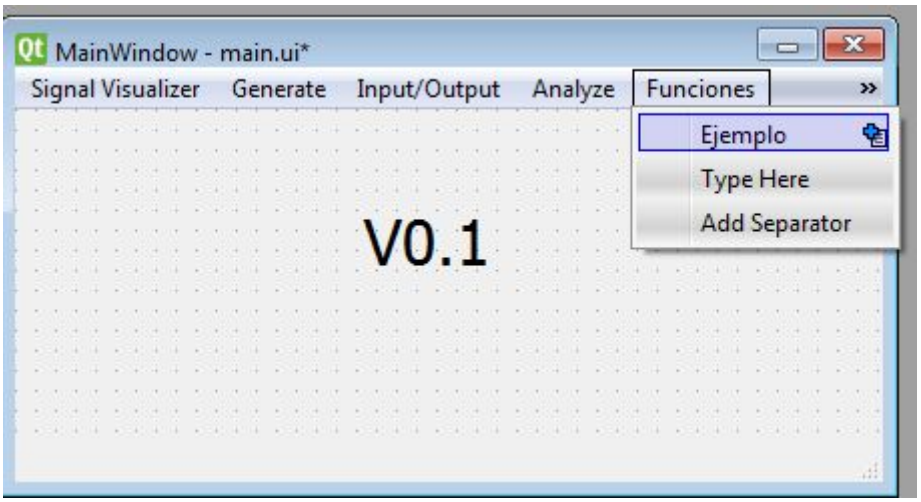
2 Se añade el código necesario en la parte de lógica.

Ejecución de cuadros de diálogo desde la ventana principal.

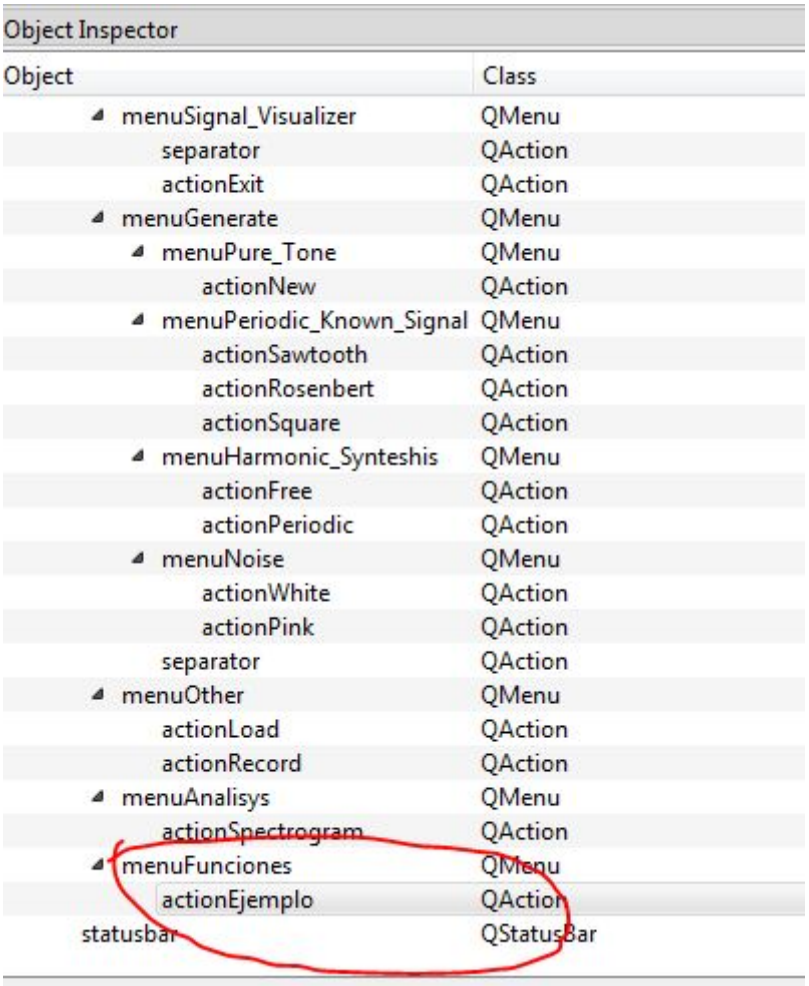
Dentro de QtDesigner abrimos el fichero main.ui que se encuentra en el **directorio ui**



Para crear una nueva funcionalidad usamos el ratón para crear una nueva entrada en la barra de opciones.



Basta con hacer clic y poner un nombre
En la parte del inspector podemos ver como tenemos una **nueva action** para su uso las actions no son mas que **metodos python para ejecutar código.**



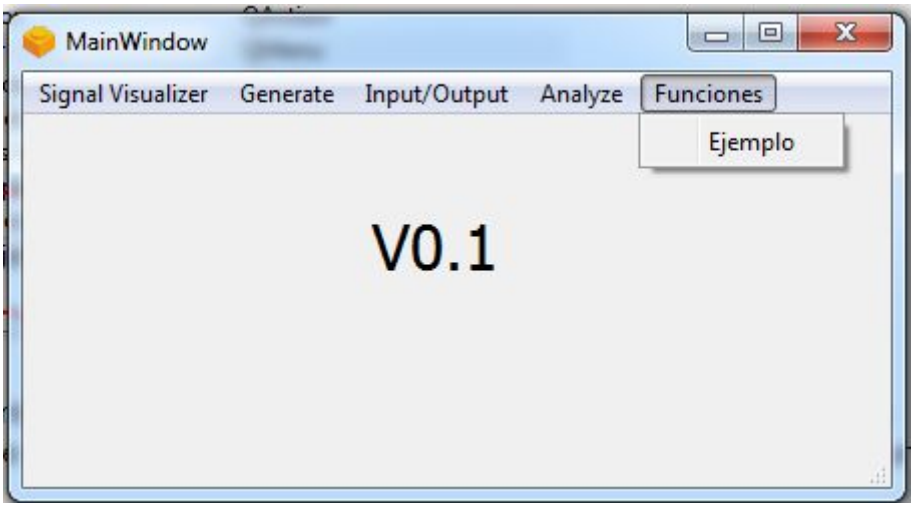
Ahora es el momento de **transpilar** la ventana principal para que los **cambios tengan efecto** pero en este caso el fichero de salida se llamara main_ui.py el nombre solo es un comvenio.

```
Pyuic5 src\ui\main.ui -o src\main\python\main_ui.py
```

Con esto aplicaremos los cambios generados en el editor.

Ahora ejecutamos

```
fbs run
```



Ahora disponemos de la nueva función integrada en la ventana el último paso es agregar la lógica.

Se dispone de un fichero main.py que integra toda la lógica de la ventana principal con sus funciones más importantes comentadas.

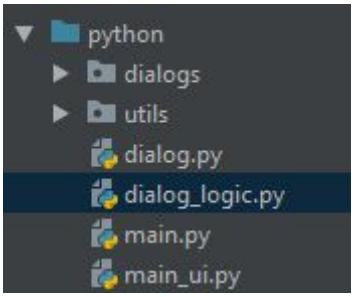
Para crear la lógica de una nueva función solo tenemos que darla de alta aquí

```
   def initialize_menu(self):
        """ ALL top menu functions go here """
        self.ui.actionNew.triggered.connect(self.open_pure_signal_dialog)

self.ui.actionSquare.triggered.connect(self.open_square_signal_dialog)
        self.ui.actionSawtooth.triggered.connect(self.open_saw_signal_dialog)
        #MD change this name to Free Harmonic Synthesis
        self.ui.actionFree.triggered.connect(self.open_hs_signal_dialog)
        self.ui.actionWhite.triggered.connect(self.open_noise_signal_dialog)
        self.ui.actionRecord.triggered.connect(self.open_record_dialog)
        #nuestra funcion
        self.ui.actionEjemplo.triggered.connect(self.open_test_dialog)
```

Para este proyecto solo se uso la barra superior de tareas de la ventana principal y es la que se cubre en este manual.

Creación de la ventana Lógica

Dentro de nuestro proyecto creamos un nuevo fichero con nombre dialog_logic.py



Este será encargado de generar la lógica, para ello heredara las funciones de dialog.py para poder usaarlas.

Su codigo sera el siguiente

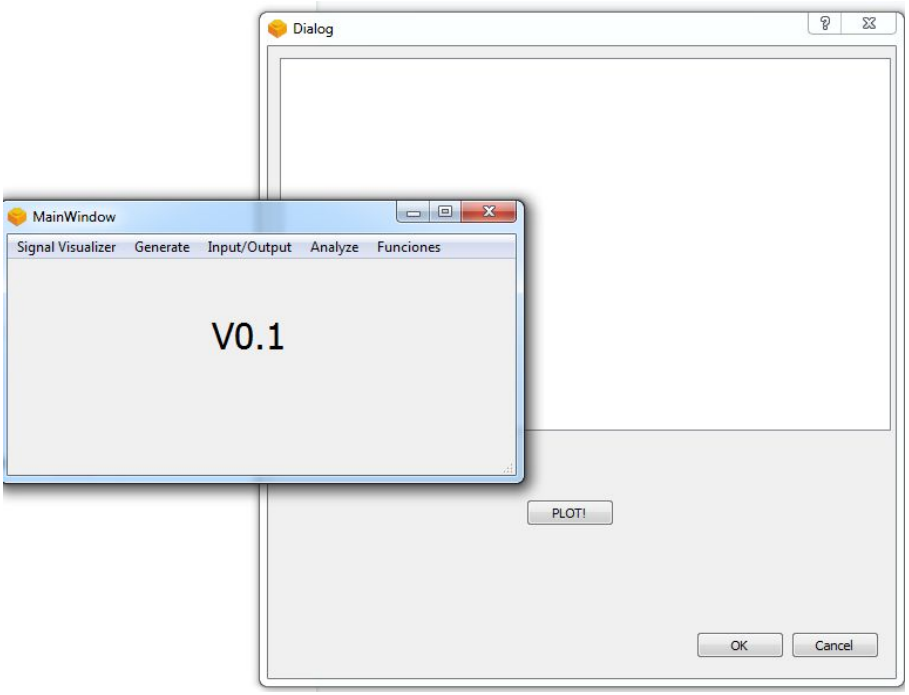
```
# relative imports
from dialog import Ui_Dialog


class Ui_DialogLogic(Ui_Dialog):
    def __init__(self):
        pass
```

Finalmente solo queda añadir el código de la función antes creada en el menú superior de fichero main.

```
#import de la clase
from dialog_logic import Ui_DialogLogic


def open_test_dialog(self):
        log.info("pure opened")

        #se crea un cuadro vacío
        self.window = QDialog()

        #se llama a la interfaz
        self.interface = Ui_DialogLogic()

        #se cargan los gráficos
        self.interface.setupUi(self.window)
        self.interface.setup_binds()

self.interface.buttonBox.accepted.connect(self.open_plot_and_close
_window)
        log.info("window added to list")
        self.windows.append(self.window)
        self.windows[len(self.windows) - 1].show()
        self.windows[len(self.windows) - 1].activateWindow()
        self.windows[len(self.windows) - 1].raise_()
        # @TODO delete windows from list when 'ok' or 'cancel'
buttons are pressed
```

Ahora importamos el fichero en main.py y llamamos al objeto creado para comprobar su funcionalidad con.

```
 fbs run
```



Una vez creada esta jerarquía se puede ejecutar código python y integrarlo con la aplicación definiendo funciones que se llaman desde la ventana de lógica.

Este manual ha cubierto los siguientes apartados:

- Instalación de Fman y un proyecto vacío
- Correr un Nuevo proyecto
- Congelar y compilar el proyecto
- Integrar Signal Visualizer en el proyecto
- Instalación de las dependencias necesarias
- Cargar el proyecto Signal Visualizer
- Añadir una nuevo diálogo con objetos personalizados en el editor
- Ejecutar código en esa ventana por medio de código python

Documentacion

https://github.com/mherrmann/fbs

https://doc.qt.io/qt-5/

http://www.pyqtgraph.org/