UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

TRABAJO FIN DE GRADO
CURSO 2021-2022

UNA APROXIMACIÓN ALGORÍTMICA A LA MÚSICA DE ARVO PÄRT

AN ALGORITHMIC APPROACH TO THE MUSIC OF ARVO PÄRT

DANIEL VILLALBA GARCÍA

GRADO EN INGENIERÍA DEL SOFTWARE

TUTOR
PROF. JAIME SÁNCHEZ HERNÁNDEZ

CONVOCATORIA ORDINARIA JUNIO DE 2022

A Jaime por descubrirme a Arvo Pärt y el extenso mundo de la programación musical.

A mis padres y mi hermana.

“I have discovered that it is enough when a single note is beautifully played.“

- Arvo Part

Resumen

Arvo Pärt es un compositor de *música sacra* de origen estonio nacionalizado austriaco. Desde 2011 y durante 8 años seguidos fue el compositor contemporáneo más interpretado y algunas de sus piezas se han utilizado en la banda sonora de películas como *"La Gran Belleza"*, "*Gravity" o* la española *"Soldados de Salamina"*. Es conocido por ser uno de los precursores de la música minimalista y más concretamente del *Minimalismo Sacro* junto a *John Tavener* y *Henryk Górecki* entre otros.

Buena parte de su obra está impregnada de reglas y procesos algorítmicos como el Tintinnabuli, su técnica más icónica. Sin embargo, su obra no suele catalogarse como música algorítmica ni es creada para ser interpretada por ordenadores.

En este trabajo analizaremos algunas de esas obras desde un punto de vista algorítmico, extraeremos las reglas y procesos que las generan y le daremos ese carácter procedural a sus piezas. Para ello utilizaremos el entorno FoxDot que nos permite utilizar Python como lenguaje para escribir y generar algorítmicamente las composiciones.

Palabras clave

Minimalismo Sacro, Python, FoxDot, SuperCollider, Tintinnabuli, MIDI, música procedural, música algorítmica.

Abstract

Arvo Pärt is a composer of sacred music of Estonian origin naturalized Austrian. Since 2011 and for 8 years in a row he was the most performed contemporary composer and some of his pieces have been used in the soundtrack of films such as "The Great Beauty" or the Spanish "Soldiers of Salamis". He is known for being one of the precursors of minimalist music and more specifically of Sacred Minimalism along with John Tavener and Henryk Górecki among others.

Much of his work is impregnated with algorithmic rules and processes such as Tintinnabuli, his most iconic technique. However, his work is not usually categorized as algorithmic music, nor is it created to be performed by computers.

In this work we will analyze some of those works from an algorithmic point of view, we will extract the rules and processes that generate them and we will give that procedural character to his pieces. For this we will use the FoxDot environment that allows us to use Python as a language to write and algorithmically generate the compositions.

Keywords:

Holy Minimalism, Python, FoxDot, SuperCollider, Tintinnabuli, procedural music, MIDI, algorithmic music.

Índice de Contenidos

1 Introducción

La motivación de este trabajo surge de las posibilidades de crear música generativa usando los algoritmos y técnicas compositivas que Arvo Pärt utiliza en algunas de sus piezas más reconocidas y elogiadas.

La música minimalista alcanzó una gran popularidad en las décadas de los 60 y 70, donde compositores como Terry Riley o Steve Reich empezaron a separarse del serialismo y la música aleatoria para empezar a trabajar con armonías más sencillas y melodías repetitivas [1]. Continuando en los años 80 y 90 varios compositores comienzan a fijarse en estos ideales estéticos e inspirados por la música medieval y renacentista definen lo que hoy se conoce como Minimalismo Sacro.

Arvo Pärt es uno de los precursores del Minimalismo Sacro en el que se enmarcan las obras de un grupo de compositores de música clásica a finales del siglo XX como John Tavener, Henryk Górecki, Alan Hobhaness y Sofía Gobaidulina entre otros[1]. Todos ellos representan este estilo compositivo muy marcado por el minimalismo y un aura espiritual.

El reconocimiento de Arvo Pärt viene avalado por una larga trayectoria en la cual ha compuesto más de 80 obras [3]. Su música ha aparecido en la banda sonora de más de 50 películas [4] como *"La Gran Belleza"*, "*Gravity*", la española *"Soldados de Salamina"* o la más reciente *"Vengadores: la era de Ultrón"*. Durante ocho años seguidos (entre 2011 y 2019) y hasta ser superado por John Williams, fue el compositor contemporáneo más interpretado. En esta época, en 2016, es nombrado doctor *honoris causa* por la Universidad de Oxford [5]. Al año siguiente recibe el premio *Ratzinger,* siendo el primer no teólogo en recibir este reconocimiento [6] y en 2020 es galardonado con el Premio Fundación BBVA Fronteras del Conocimiento en la categoría de Música y Ópera [7].

1 A pesar de que muchos musicólogos los agrupan juntos dentro del Minimalismo Sacro, debido a sus distintas nacionalidades y creencias religiosas, los compositores suelen rechazar este término y de ninguna manera forman "escuela", pero su uso está extendido debido, principalmente, a la ausencia de un término mejor [2]

Arvo Pärt, venía de un primer periodo donde trabajó el dodecafonismo y la música neoclásica. Las obras enmarcadas en estos estilos, destacan por ser complejas y requerir una escucha exigente y un extenso conocimiento musical para llegar a comprenderlas. Durante este periodo, Pärt entró en una crisis creativa y quiso apartarse de toda esa sofisticación, volviendo a las raíces de la música occidental y de las obras del renacimiento, con armonías y acordes más sencillos y sin excesivos adornos, creando una atmósfera espiritual y religiosa [8]. En palabras del propio Pärt: "*Lo complejo y lo multifacético solo me confunde, y debo buscar la unidad*".

En este trabajo analizaremos algunas de las técnicas que Arvo Pärt idea para conseguir esta atmósfera, donde la más emblemática y la que ha servido de inspiración para este trabajo es el *Tintinnabuli* [9], introducido en su obra *Für Alina* (1976) [10], que analizaremos para codificarla algorítmicamente [11].

El *Tintinnabuli* es una técnica de contrapunto que se compone de dos voces, una voz principal que, por lo general, es una melodía sencilla y la voz *tintinnabular*, que la acompaña de forma paralela moviéndose únicamente entre las 3 notas de un acorde, que normalmente corresponde con el acorde de tónica de la pieza.

Música Procedural y Generativa

En la música han existido algoritmos desde hace siglos, mucho antes de la existencia de las computadoras. Estos algoritmos pueden encontrarse en algunas técnicas de contrapunto e incluso en composiciones antiguas, como puede ser el "juego musical de dados" (conocido en alemán como *Musikalisches Würfelspiel*) muy popular en el siglo XVIII en Europa [12], donde se componían una serie de motivos que después eran interpretados en el orden que aleatoriamente decidiese una tirada de dados.

Fue con la llegada de las computadoras cuando los algoritmos comenzaron a usarse con más profundidad en la música y comenzaron a aparecer términos como **música procedural** o **música generativa.**

Karen Collins define la música procedural como *“una composición que evoluciona en tiempo real, de acuerdo con un conjunto específico de reglas lógicas de control”* [13]. Esto es música generada basándose en algoritmos y comportamientos iniciales. Por lo tanto, el compositor no escribe una composición tradicional en forma de partitura, sino que define reglas que ejecutará el sistema para generar dicha música.

De manera similar, Brian Eno introduce el término **Música Generativa** en su álbum homónimo *Generative Music 1,* un álbum musical con 12 composiciones escritas en software y presentadas en un *Floppy Disk* [14]

El término "música generativa" suele utilizarse indistintamente con el de "música algorítmica", ya que gran parte de la música generativa contemporánea se crea con la ayuda de algoritmos generados por ordenador (Collins, 2008) [15].

La diferencia entre música generativa y música procedural es algo difusa, ambos estilos están relacionados con la “Música algorítmica”. La diferencia principal es que en la música generativa suele hablarse de composiciones interpretadas por un sistema o una computadora; en el álbum de Eno, *Generative Music 1*, las piezas eran presentadas como programas de ordenador. Mientras que en la música procedural hablamos también de reglas de control, pero no necesariamente entra en juego el computador.

No podemos decir que Arvo Pärt componga música procedural o generativa pues sus obras siguen una estructura convencional y son escritas para ser interpretadas por músicos de orquesta, pero como veremos, en muchas de ellas utiliza patrones y algoritmos para escribirlas, empezando por su *Tintinnabuli,* una técnica de contrapunto puramente algorítmica.

Objetivos

Los objetivos principales fijados al comienzo del proyecto fueron:

- Analizar algunas de las composiciones de Arvo Pärt desde un enfoque algorítmico. En particular, *Für Alina*, *Fratres* o *Cantus in Memoriam Benjamin Britten*.
- Reconstruir estas obras de Pärt en forma de composiciones procedurales e interpretarlas con el ordenador.
- Estudiar herramientas y entornos de programación donde implementar las piezas.
- Aplicar las técnicas de Pärt para crear composiciones que puedan usarse como música ambient [16].

Estado del arte

En internet pueden encontrarse implementaciones algorítmicas de algunas técnicas y obras de Arvo Pärt. En GitHub encontramos algunos repositorios donde vemos estas implementaciones de algunas de sus técnicas en diferentes lenguajes [17], un "Tintinnabulador" [18]: un instrumento virtual que nos genera el Tintinnabuli de cualquier motivo que interpretemos con él e incluso una aproximación matemática a esta técnica que ha sido parte de la inspiración para este trabajo [19].

Algunas de estas implementaciones están escritas en SuperCollider, entorno que vamos a explorar en este trabajo, aunque nosotros vamos a trabajar en FoxDot que se comunica internamente con SuperCollider para hacer sonar la música. Hemos elegido FoxDot para el trabajo porque utiliza Python como lenguaje, haciendo el código más comprensible para un programador no músico. Además, trataremos de ir un poco más allá y explorar en detalle estas y otras de sus técnicas, para, de forma algorítmica, generar de una manera lo más fiel posible sus obras al completo.

Estructura de la memoria

En el capítulo 2 hablaremos de la obra musical de Arvo Pärt centrándonos en su etapa minimalista. Analizaremos las técnicas de composición que usa en sus piezas desde un enfoque algorítmico. Para este capítulo se requiere un conocimiento musical básico, ya que hablaremos de notas, escalas, tonalidades y acordes.

Para el capítulo 3 estudiaremos las diferentes herramientas y entornos de síntesis musical, justificaremos el por qué las hemos elegido y explicaremos algunas de sus funcionalidades principales.

En el capítulo 4 explicaremos las implementaciones de cada una de estas obras y como las hemos llevado a código y a hacer que suenen. Para terminar, en el capítulo 5, presentaremos las conclusiones del trabajo y plantearemos el posible trabajo futuro.

Código de generación y demos de las piezas

En el siguiente repositorio se puede consultar el código del proyecto;

https://github.com/Visalba/ArvoPart

Para cada una de las piezas hemos generado dos versiones, una preparada para usarse con el MIDI y otra más sencilla, que puede ejecutarse únicamente teniendo FoxDot instalado con los sintetizadores por defecto.

Además de ello, hemos creado un SoundCloud donde poder escuchar las demos de las piezas:

https://soundcloud.com/daniel-villalba-garcia/sets/arvo-part

2 Arvo Pärt y el Tintinnabuli

La obra de Arvo Pärt puede clasificarse en dos periodos. En este trabajo nos vamos a centrar exclusivamente en su etapa más tardía donde se engloba su producción minimalista, pero cabe destacar que no siempre compuso este tipo de música. Su primer periodo es completamente opuesto al minimalismo, comenzando con un neoclasicismo muy próximo a *Dmitri Shostakóvic,* para desembocar en una música fuertemente influenciada por *Arnold Schönberg*, y la corriente dodecafónica y *serialista*, en la cual sus obras pueden ser bastante "densas". Una de las obras más representativas de este periodo es su *Sinfonía número 1* [20].

Centrándonos ya en su periodo minimalista, Arvo Pärt habla de su música como *"una luz blanca que contiene todos los colores. Solo un prisma podría dividir esos colores y hacer que aparezcan. Este prisma podría ser el espíritu del oyente* [21]*"*. Con esta descripción, Pärt ilustra la idea de su minimalismo y en lo que nosotros nos vamos a enfocar para este trabajo: un contenido sencillo, sin demasiadas pretensiones (la *luz blanca*), pero que no deja de lado la complejidad de los *colores* que la componen y de los cuales se hace partícipe al oyente para su interpretación.

Nuestro "prisma" va a ser el de la algoritmia, en este capítulo vamos a analizar algunas de las obras de este periodo y a encontrar las reglas y patrones que las componen y más tarde usaremos estos "colores" para interpretarlas.

Vamos a empezar entonces con su técnica más conocida y sobre la que se van a cimentar las obras que vamos a analizar.

Tintinnabuli

El término *Tintinnabuli* fue acuñado por el propio Arvo Pärt y hace referencia al sonido "tintineante" de una campana. Él lo define así:

"Tintinnabuli es la conexión matemáticamente exacta de una línea a otra… Tintinnabuli es la regla que convierte la melodía y el acompañamiento...en uno. Uno más uno, es uno – no es dos. Este es el secreto de esta técnica [22]*."*

La música *tintinnabular* se compone fundamentalmente de dos voces, una voz principal (main voice) o M-Voice y una voz tintinnabular o T-Voice.

La voz principal toca una melodía generalmente sencilla en una tonalidad fija, mientras que la voz tintinnabular la acompaña con notas del acorde perfecto, generalmente el de tónica.

De esta manera, a cada nota de la escala, le corresponde una única nota del acorde de primer grado de la tonalidad. Veamos el siguiente ejemplo en la escala de La menor:

Escala La menor	La	Si	Do	Re	Mi	Fa	Sol	La'	Si'	Do'
Tintinnabuli	Mi	La	La	Do	Do	Mi	Mi	Mi	La'	La'

En la fila verde tenemos las notas de la escala de La menor que correspondería con la M-Voice y en la azul, las notas del acorde de tónica que le corresponden a cada una de ellas. La única disonancia la encontraríamos en el sexto grado de la escala, es decir, en el Fa-Mi.

En este ejemplo, la T-Voice corresponde con la nota del acorde de La menor (La, Do, Mi) más cercana por debajo de la nota de la M-Voice, a esto le vamos a llamar **T-1**. El Tintinnabuli puede extenderse en varios niveles:

T+3	La	La	Do	Do	Mi	Mi	Mi	La	La	Do
T+2	Mi	Mi	La	La	Do	Do	Do	Mi	Mi	La
T+1	Do	Do	Mi	Mi	La	La	La	Do	Do	Mi
M-Voice	La	Si	Do	Re	Mi	Fa	Sol	La	Si	Do
T-1	Mi	La	La	Do	Do	Mi	Mi	Mi	La	La
T-2	Do	Mi	Mi	La	La	Do	Do	Do	Mi	Mi
T-3	La	Do	Do	Mi	Mi	La	La	La	Do	Do

Tabla 1: Correspondencias Tintinnabuli

En la fila central de esta tabla tenemos la escala de La menor actuando de voz principal o *main voice.* Tanto por encima, como por debajo suya, tenemos los diferentes niveles del Tintinnabuli que se mueven en las notas del acorde de tónica, es decir, el acorde de La mayor. Para las filas positivas tenemos las notas de ese acorde, que se encuentran por encima de las de la voz principal.

En cada nivel (T+n), la n representa el número de saltos desde la nota de la M-Voice hasta la nota más cercana del acorde de tónica. Si hablamos de las teclas del piano, sería la tecla correspondiente a la nota del acorde más cercana por la derecha dando n saltos. Lo homólogo para las filas inferiores de la tabla (T-n).

Un acorde simple está formado por 3 notas, a estas notas se les conoce como la 1ª (La en este caso), 3ª (Do) y la 5ª (Mi) en referencia al lugar que ocupan a partir de la tónica. Si cogemos la fila T-1 de la tabla, vemos que la sucesión de notas sería 5, 1, 1, 3, 3, 5, 5… De esta manera podría transportarse a cualquier otra tonalidad. Si por ejemplo hablamos de Re menor la 1ª es Re, la 3ª Fa y la 5ª La. En el capítulo 4, construiremos esta tabla en forma de matriz y definiremos estas correspondencias de manera general.

A continuación, vamos a analizar la obra esencial de este estilo.

Für Alina

Con esta obra, Pärt introdujo el Tintinnabuli. Es una pieza en Si menor para piano solo, que consta de dos voces, la voz principal (M-Voice) y la voz tintinnabular (T-Voice), en el piano, la primera se toca con la mano derecha (en una octava por encima de lo indicado en la partitura) y la segunda con la izquierda, ambas escritas en clave de Sol.

La interpretación de la pieza es bastante libre, no hay señal de tempo. La única anotación de la que disponemos es *"Ruhig, erhaben, en sich hineinhorchend"* que significa, algo así como: *pacíficamente, en una manera elevada e introspectiva.* Solo hay presentes dos tipos de notas, redondas, y notas sin plicas, es decir, de libre duración.

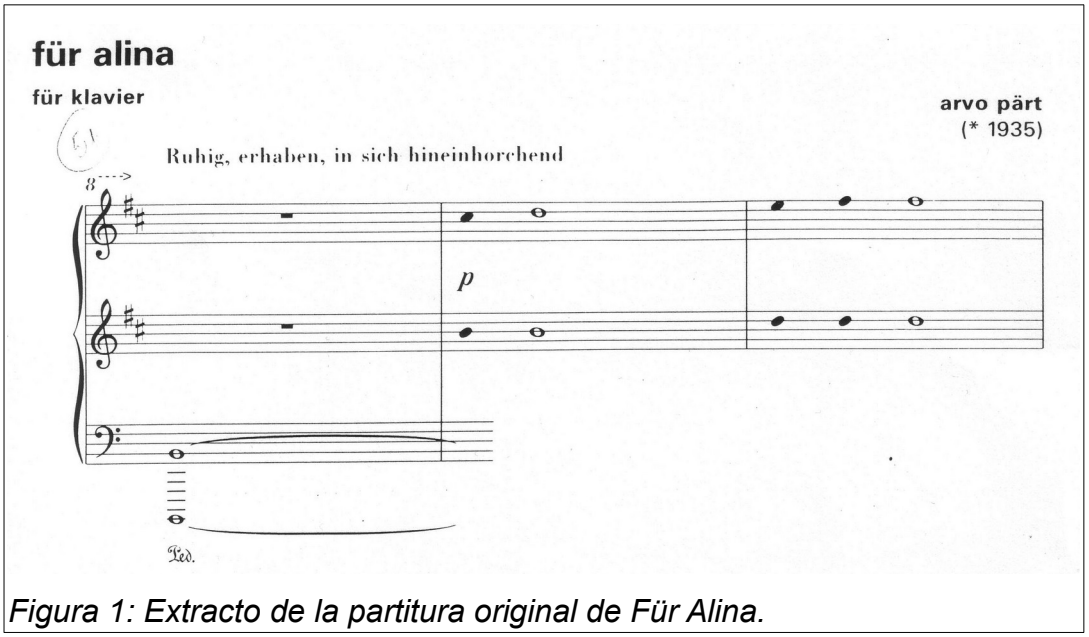


Figura 1: Extracto de la partitura original de Für Alina.

Con estas directrices tan difusas, podría parecer que la obra no va a seguir una estructura identificable, pero nada más lejos de la realidad.

En primer lugar, identificamos el Tintinnabuli. Como la pieza está en tonalidad de Si menor, la voz tintinnabular siempre se moverá entre las notas Si, Re y Fa# que forman el acorde de Si menor. Según lo definido anteriormente, la voz tintinnabular sería la **T-1**, es decir, a cada nota de la escala de Si menor le corresponde la nota del acorde más próxima por debajo. Esta sería su tabla de correspondencia:

Si	Do#	Re	Mi	Fa#	Sol	La
Fa#	Si	Si	Re	Re	Fa#	Fa#

Si nos fijamos en las líneas o compases de la pieza, vemos que hay únicamente 15, cada una con un número de notas distinto[2], la sucesión de estas es de la siguiente manera: la primera empieza con una 16*va* baja que se alarga durante toda la obra, la segunda tiene una nota de libre duración seguida de una redonda, la siguiente dos notas de libre duración y una redonda y así sucesivamente aumentando el número de notas de libre duración, de manera que siempre acabamos la línea con una redonda. Esta secuencia se repite hasta que llegamos a las 8 notas, donde la secuencia se invierte y empiezan a reducirse el número de estas, 7, 6, 5… siendo la penúltima línea de dos notas (una nota de libre duración y una redonda) para volver a aumentar. La serie, en cuanto al número de notas es entonces: 1 2 3 4 5 6 7 8 7 6 5 4 3 2 3… Esta simetría hace que la pieza sea cíclica y pueda interpretarse un número ilimitado de veces.

Es interesante recalcar como, en un primer vistazo de la partitura, podemos pensar que es una composición caótica, no tenemos compases ni un tempo definido y, además, la mayoría de notas son de duración arbitraria, pero si profundizamos en su análisis, vemos como Pärt sigue una estructura muy firme, con una sucesión muy sencilla pero que la propia notación musical no le permite expresar de una manera clara de comprender a simple vista.

2 Ver **Anexo A** con la partitura original de Für Alina

Cantus in Memoriam Benjamin Britten

Esta pieza es un canon escrito en La menor con un compás de 6/4. Es un *Treno*[3] al compositor Benjamin Britten compuesto para orquesta de cuerda y campana [23].

La pieza es en forma de **canon proporcional** (*Prolation Canon*). Este tipo de canon se caracteriza por tener una melodía principal acompañada por una o varias imitaciones que no interpretan la melodía idénticamente a la principal, sino que lo hacen a diferentes velocidades o proporciones.

La melodía principal del "Cantus" consiste en un descenso por la escala de La menor alternando notas largas y cortas, comenzando por la nota más alta y formando la secuencia de manera que, cada vez que se llega al final de esta, se añade la nota siguiente de la escala.

La pieza comienza con el sonido de una campana tubular afinada en La (la nota fundamental de la composición), con 12 pulsos entre cada golpe y una separación de 18 pulsos en cada agrupación de tres.

Después de 3 pulsos del séptimo compás, entran los primeros violines reproduciendo la secuencia descrita, comenzando en la nota más alta de su registro, alternando blancas y negras.

Para todas las secciones de la orquesta (excepto las violas), tenemos la mitad de ella tocando la secuencia en la escala de La menor y a la otra mitad acompañándola con un *Tintinnabuli* de **T-1**. La tabla de correspondencia es la siguiente:

La	Si	Do	Re	Mi	Fa	Sol
Mi	La	La	Do	Do	Mi	Mi

Los segundos violines repiten exactamente lo mismo pero una octava por debajo y a la mitad de velocidad, por lo tanto, entran 6 pulsos más tarde del inicio del compás número 7 (justo al inicio del compás 8) y alternan redondas y blancas.

3 Canto fúnebre o lamentación por alguna calamidad o desgracia. (definición de la RAE)

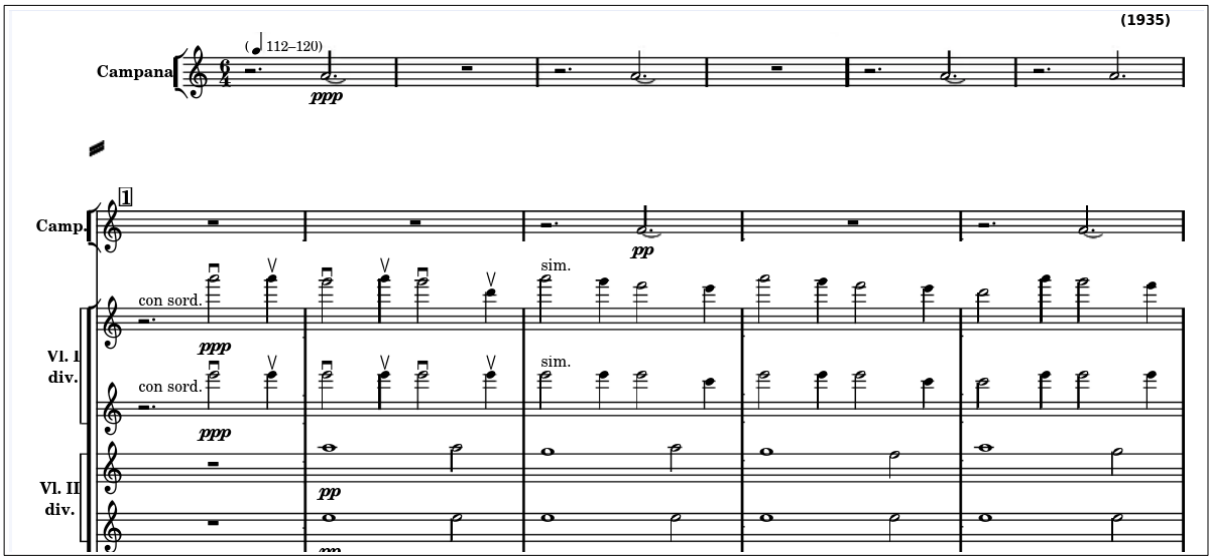
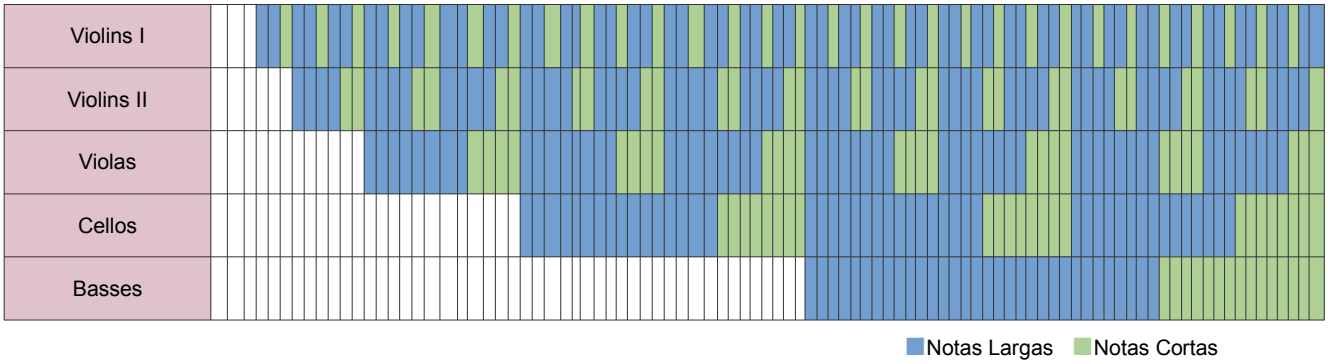


Figura 2: Cantus in Memoriam Benjamin Britten. Extracto de la partitura original.

La siguiente sección en entrar son las violas a la mitad de velocidad que los Violines II (a un cuarto de los Violines I) una octava por debajo, siendo el centro del registro de la pieza. Es la única voz que no que no es doblada por el Tintinnabuli.

Después entra la sección de violoncelos, a la mitad de las violas y por último los contrabajos, de nuevo a la mitad de los violoncelos.

Tabla 2: Línea temporal de Instrumentos. "Cantus".



En el compás 65, los violines primeros alcanzan el Do central, en ese momento dejan de reproducir la secuencia y se mantienen en esa nota por el resto de la pieza. Lo mismo hacen los segundos violines con el La bajo. Cuando ambas voces han alcanzado su objetivo, en ese punto de la obra toda la orquesta está interpretando un La menor, la tónica de la pieza, que se alarga durante 5 compases más hasta que para, momento en el que puede percibirse de nuevo la campana del principio.

En las siguientes dos composiciones, veremos como Pärt varía ligeramente su Tintinnabuli.

Fratres

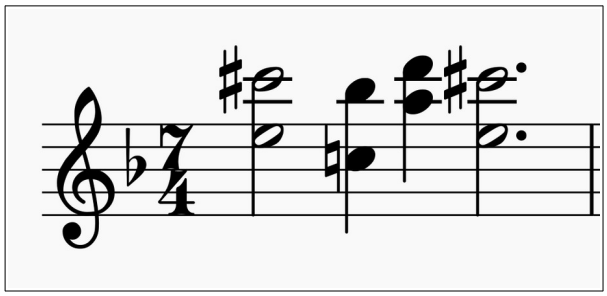
Esta es una de las piezas más ricas en cuanto al uso del Tintinnabuli de Arvo Pärt [24].

En ella, tenemos un motivo principal de 6 compases que se repite con variaciones a lo largo de los 11 minutos que dura la pieza, en el cual la melodía principal repite una secuencia de notas de la escala de La Mayor frigia y por debajo de ella tenemos otra melodía que sigue de manera exactamente paralela a la principal, pero una décima por debajo. Entre medias de ellas tenemos la voz Tintinnabular, que sería un **T-2,** pero en el segundo grado de la escala (Sib) al que le correspondería un Mi, Pärt le asigna un Do.

La	Sib	Do#	Re	Mi	Fa	Sol
Do	Do	Mi	La	La	Do	Do

Vamos a ver como se construye el motivo que se va a repetir durante la pieza.

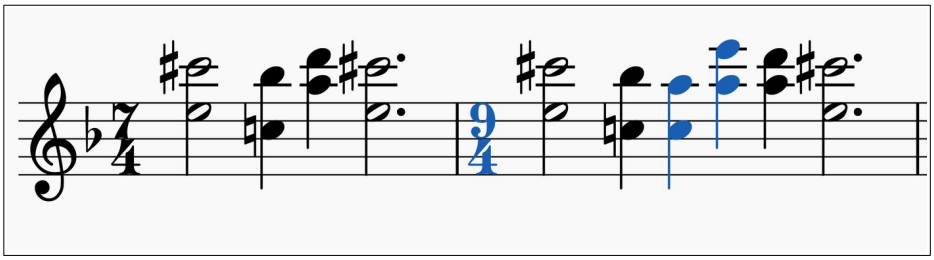
Empezamos con estas 4 notas:



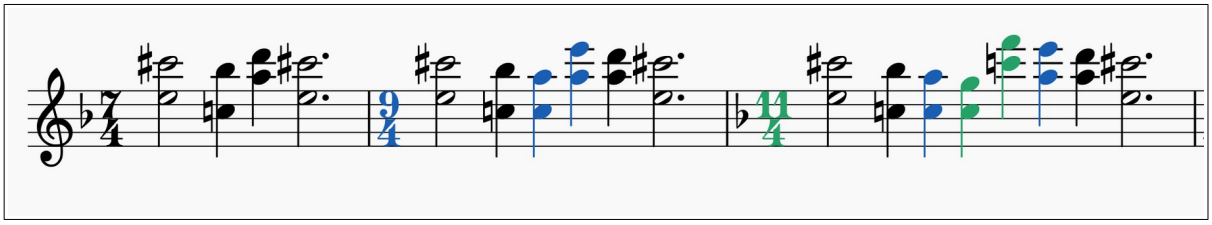
Las notas altas representan la melodía principal y las de abajo el Tintinnabuli[4].

Como vemos esta melodía empieza y termina en Do# y está compuesta de dos descensos.

Para el siguiente compás añadimos 1 nota de la escala a cada uno de los descensos, al primero una nota por debajo y al siguiente por arriba quedando así:

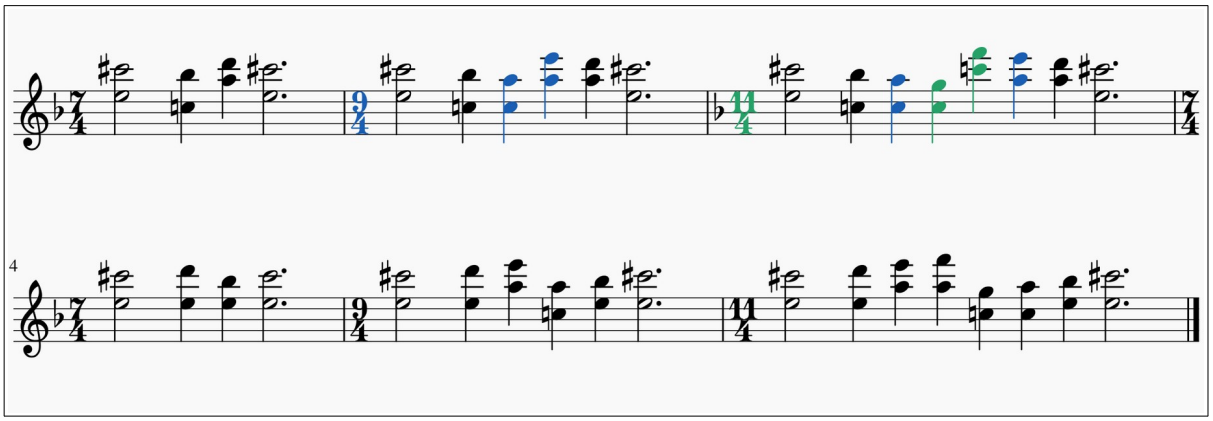


Y para el siguiente hacemos lo análogo aplicando la misma lógica:



4 A esto habría que añadirle la décima por debajo que no incluimos para hacer más clara la explicación.

Con esto ya tendríamos los tres primeros compases del motivo principal. Para los tres siguientes, lo que Pärt hace es invertir cada uno de los anteriores, es decir, tocar sus notas de manera inversa:



Este es el motivo que va a acompañar toda la obra repitiéndose 9 veces, en cada repetición se desciende una tercera.

A las repeticiones las separa lo que en música habitualmente se llama “refugio”, esto es un motivo (en este caso de percusión) que es idéntico en todas las repeticiones y que marca el final de una parte y comienzo de la siguiente.

Pärt ha escrito 17 versiones de *Fratres* [25] tanto para conjuntos, como para solo y acompañamiento, con distintas variaciones, pero el motivo principal que hemos analizado es idéntico en todas ellas.

Dependiendo de la versión este motivo formará parte de la base armónica. Normalmente por encima hay un instrumento solo.

Spiegel im Spiegel

Aunque esta composición no vamos a implementarla en el capítulo 4, ya que no es tan representable algorítmicamente como el resto, merece que la analicemos musicalmente ya que es una de las piezas más reconocidas de Arvo Pärt y es una de las más utilizadas tanto en películas como en televisión [26] y el uso que hace Pärt del Tintinnabuli en ella puede ser inspirador [27].

Spiegel mi Spiegel, es una pieza escrita originalmente para piano y violín, aunque a veces es reemplazado por cello o viola. Es una composición en un compás de 6/4 en Fa mayor.

La línea del violín es la voz principal, se basa en una línea melódica ascendente de notas largas, que comienza con una escala de dos notas de Sol y La, que asciende alternativamente y luego desciende a La por pasos. Con cada ascenso y descenso posterior, se añade una nota a la línea, un proceso que podría continuar indefinidamente, algo a lo que, como hemos visto, Pärt recurre mucho en su minimalismo. El piano hace la voz tintinnabular, aunque en este caso, no acompaña a la voz principal nota por nota, si no que cada nota tiene asignado una inversión del acorde de Fa mayor, y arpegia de manera ascendente en negras sobre esa inversión hasta que el violín cambia de nota [28].

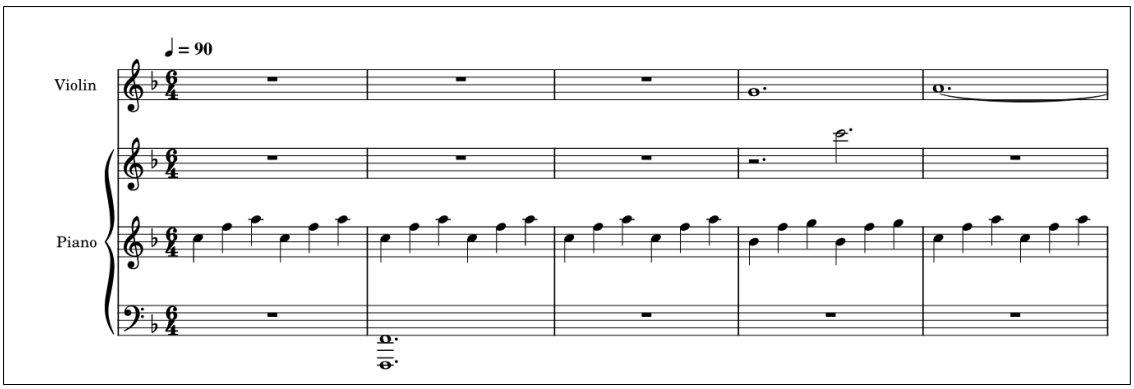


Figura 3: Extracto de la partitura original de Spiegel im Spiegel

3 Herramientas de Síntesis Musical

En las últimas décadas, la infórmatica y la música han estado muy unidas. Hoy en día, son pocas las piezas musicales que no son producidas digitalmente, parcial o totalmente y tenemos infinidad de software dedicado tanto a producción como a análisis y escritura musical [29].

El software de síntesis musical puede ser clasificado en software de notación musical para escribir partituras: MuseScore (usado en este trabajo para la generación de las imágenes explicativas de Fratres en el capítulo anterior) o Sibelius. Software de producción musical: los llamados DAWs como Ableton, Pro Tools o Reaper (que exploraremos mínimamente) y los entornos de programación musical: Csound, Chuck, MAX/MSP o Pure Data, donde las composiciones son escritas en forma de código o instrucciones para interpretadas por la computadora.

Es esta última categoría en la que más nos vamos a centrar para este trabajo y en sus posibilidades para generar música a partir de algoritmos.

FoxDot

De entre todos los lenguajes orientados a la programación musical, como puede ser SuperCollider o TidalCycles y SonicPi, más orientados a el Live Coding, nos hemos decantado por FoxDot por estar basado en Python. Esto nos interesa especialmente porque es un lenguaje muy eficaz para prototipado, flexible y que al ser interpretado facilita el live coding, además de que hace que cualquier composición sea muy visual y comprensible incluso para las personas que no lo conocen.

FoxDot es un entorno de código abierto de programación en vivo o *Live Coding* que nos permite crear música de manera muy directa sin, necesariamente, tener un conocimiento extenso en programación ni en música, lo cual no nos impide que podamos utilizar funciones e implementaciones complejas en Python. Corre por encima de SuperCollider, un conocido entorno de programación musical, que le aporta un potente motor de síntesis para producir el sonido final.

Sobre SuperCollider, FoxDot nos da una facilidad extrema para sincronizar instrumentos gracias a su scheduler del tempo, donde mantiene un beat con el compás y tempo que definamos y cualquier instrumento que definamos va seguir ese latido interno.

Existe además una extensa documentación oficial sobre todas sus funcionalidades, un tutorial de instalación [30], así como un foro para consultas de los usuarios [31].

Procedemos a detallar algunas de las funciones principales que nos van a interesar para el desarrollo de este trabajo.

Players

FoxDot nos permite definir players que tienen asignados sintetizadores donde proporcionaremos una secuencia de números en forma de lista que serán las notas que queremos que suenen en bucle. Estos instrumentos tienen una serie de atributos que podremos modificar dependiendo de cómo queremos que interpreten esas notas. Veamos un ejemplo.

```
p1 >> pluck([0, 2, 4], dur=[1, 1/2, 1/2], amp=0.75)
```

Se define el player p1 que utiliza un sintetizador que simula una cuerda (*pluck*) tocando las notas Do (0), Mi (2) y Sol (4) en bucle, con una duración de negra y dos corcheas respectivamente con una amplitud (o volumen) de 0.75.

Por supuesto, podemos definir tantos players como queramos y estos se sincronizarán entre ellos, pudiendo alterar además esta sincronización.

Escalas, Tonalidades y Tempos

Si hablamos de características de la composición, podemos modificar varios de sus parámetros.

Con `Root.default` podemos definir la tónica de la escala en la que se va a interpretar la composición. Antes hemos visto como la nota Do se corresponde con el número 0, esto es porque FoxDot, por defecto, establece la tonalidad en Do mayor, si nosotros quisiéramos establecer una tonalidad de, por ejemplo, Mi menor tendríamos que ejecutar:

```
Root.default = “E”
Scale.default = “minor”
```

De esta manera ahora el `0` sería un Mi, el 2 un Sol y el 4 un Si. Esto nos permite poder modificar en tiempo real la armonía de lo que esté sonando dando muchísima flexibilidad a todo lo que queramos tocar, pudiendo interpretar una misma composición en cualquier tonalidad y escala que queramos.

Reloj

El reloj controla el pulso interno de FoxDot, aunque no estemos ejecutando nada, el pulso se mantiene en el número que le hayamos asignado. Podemos modificar el tempo con:

```
Clock.bpm = 90
```

Además de modificar el tempo, podemos usar el Clock para sincronizar instrumentos, por ejemplo, para indicarles el pulso en el que queremos que entren con Clock.schedule:

```
Clock.schedule(lambda: p1 >> pluck(0, dur=2, amp=0.75)
                       Clock.now()+4)
```

El *Clock.schedule()* tiene dos parámetros, el primero es la expresión lambda que en nuestro caso va a ser un *player* y el segundo es el pulso en el que queremos que entre el instrumento, en este caso *Clock.now()* nos devuelve el pulso en el momento que se ejecuta la instrucción y a eso le sumamos 4 pulsos, es decir, el instrumento empezaría a sonar 4 pulsos después de haber ejecutado la instrucción.

SuperCollider

Para este trabajo no vamos a profundizar mucho en el funcionamiento de SuperCollider, pero sí merece la pena explicar sus funcionalidades principales, ya que, como hemos mencionado anteriormente, es el que presta la base a FoxDot para sonar.

SuperCollider es un entorno de programación para síntesis de sonido en tiempo real y composición musical. Tiene un lenguaje propio llamado *scsynth* [32] y su aplicación de servidor [33] es compatible con plugins en C, además de que se puede utilizar de manera independiente y es posible comunicarlo con el exterior con multiples canales de entrada y salida, siendo esta la característica que más nos va a interesar, ya que FoxDot mandará sus señales MIDI a través de él para que suenen en instrumentos virtuales que veremos más adelante. Con esto vamos a poder producir el sonido final que generemos, pudiendo utilizar plugins profesionales de emulación de instrumentos reales.

Gracias a SuperCollider, FoxDot suena, ya que le provee del servidor de audio en tiempo real y ofrece la posibilidad al usuario de definir sintetizadores virtuales que FoxDot puede utilizar. Además, cuando iniciamos FoxDot, este arranca con una gama amplia de sintetizadores de distintos tipos que le da SuperCollider, igualmente podemos definir nosotros nuevos a través de SC.

Cabe destacar que cualquier cosa que hagamos en FoxDot puede hacerse en SuperCollider, pero como vimos en el epígrafe anterior. FoxDot nos permite hacerlo todo de manera más directa ya que nos proporciona un scheduler de eventos y nos facilita la gestión del tempo sobre SuperCollider.

MIDI en Reaper

MIDI (siglas de Musical Instrument Digital Interface) es un estándar tecnológico que describe un protocolo, una interfaz digital y conectores que permiten que varios instrumentos musicales electrónicos, ordenadores y otros dispositivos relacionados se conecten y comuniquen entre sí. Una simple conexión MIDI puede transmitir hasta dieciséis canales de información que pueden ser conectados a diferentes dispositivos cada uno [34].

FoxDot nos da la posibilidad de enviar mensajes MIDI a través de SuperCollider y este último puede conectarse como instrumento MIDI. En nuestro caso vamos a emplear el DAW[5] Reaper para conectarnos y enviarle las señales MIDI.

Reaper [35] es un software de producción musical y secuenciador MIDI con un modelo de licencia en el cual nos permiten su uso gratuito sin ninguna restricción de funcionalidades confiando en la **honestidad** del consumidor a la hora de valorar el software.

A pesar de que Reaper nos ofrece infinidad de funcionalidades para la producción musical, nosotros para este proyecto nos vamos a aprovechar de su posibilidad para alojar instrumentos virtuales (VST instruments) y tratar de conseguir un sonido lo más profesional posible..

Los plugins VST son interfaces que simulan de manera virtual instrumentos, sintetizadores y diversos sistemas de grabación y producción musical, en el caso de los instrumentos virtuales, son capaces de recibir señales MIDI y reproducirlas, ideal para nuestro objetivo.

5 Siglas de Digital Audio Workspace

Nosotros vamos a correr toda la instalación en Windows. En Linux, SuperCollider puede conectarse directamente por MIDI a Reaper o cualquier otro DAW, pero en Windows no.

Para comunicar SuperCollider y Reaper necesitamos una aplicación que sirva de "puente" entre ambas, donde SuperCollider envíe las señales y Reaper se conecte a ella para recibirlas, en nuestro caso vamos a usar **LoopMIDI** [36] aunque existen muchas alternativas con funcionalidad similar.

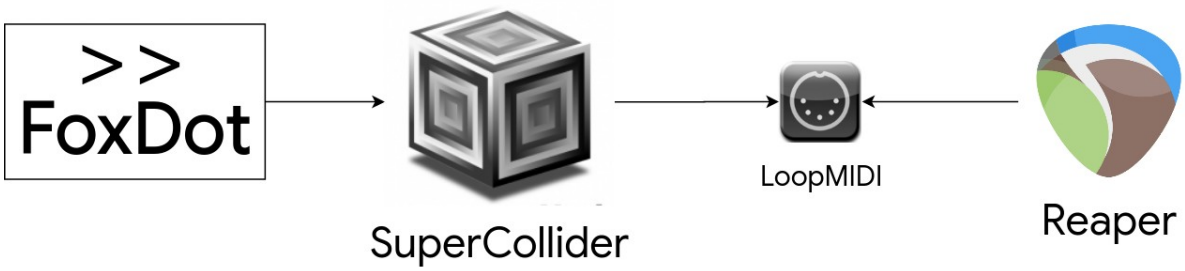


Figura 4: Diagrama de arquitectura FoxDot->Reaper

4 Implementación

En este capítulo vamos a aplicar todo lo que hemos visto hasta ahora. Extraeremos los algoritmos que hemos explicado en el capítulo 2, los implementaremos en Python y los integraremos en FoxDot.

La matriz del Tintinnabuli

Empezamos implementando el Tintinnabuli porque es el pilar fundamental en el que se va a apoyar buena parte de nuestro trabajo.

Como hemos visto, una aproximación correcta a las correspondencias entre las notas del Tintinnabuli puede ser una matriz, en la que las filas correspondan a cada uno de los niveles (T+1, T+2…) y las columnas sean las notas en escala. Lo tenemos de la siguiente manera.

```
mat = [
    [0,0,2,2,4,4,4],    # 0
    [2,2,4,4,7,7,7],    # t+1
    [4,4,7,7,9,9,9],    # t+2
    [7,7,9,9,11,11,11], # t+3
    [-7,-5,-5,-3,-3,0,0],  # t-3
    [-5,-3,-3,0,0,2,2], # t-2
    [-3,0,0,2,2,4,4,4]  # t-1
]
```

Si comparamos esta matriz con la explicada en la sección del Tintinnabuli en el capítulo 2, vemos que la abstracción de las notas es directa. En ese capítulo hablábamos de que el acorde básico está compuesto de las notas 1ª, 3ª y 5ª, la conversión para esta matriz es idéntica, pero restando 1 a cada una de ellas.

La fila #0 corresponde con el acorde relativo, veremos que nos puede ser útil en determinados casos.

Teniendo definida la matriz de correspondencia, vamos a definir y explicar la siguiente función que, dada una nota y un nivel del Tintinnabuli, nos devuelve su nota correspondiente de ese nivel.

```
def getTvoice(m,t): #m=nota, t=nivel
    oct, m = m//7, m%7
    return mat[t][m]+oct*7
```

De manera que si ejecutamos getTvoice(4,3) nos devuelve 11, es decir, el valor 4 de la columna 3.

En primer lugar, calculamos el modulo y la división entera de la nota dada, con el primero hacemos que las listas de la matriz sean circulares y con la segunda, conseguimos extender el Tintinnabuli a notas de cualquier altura, ya que nos devuelve la octava en la que se encuentra dicha nota, de esta manera, la nota 67 sabríamos que se encuentra en la octava 4.

```
def getTvoiceFromMelody(motive, tintinnabuli):
    tvoice = []
    for note in motive:
        tvoice.append(getTvoice(note, tintinnabuli))
    return tvoice
```

Esta función devuelve una lista de notas correspondiente a la voz Tintinnabular. Recorre la lista de notas dada y para cada una de ellas aplica la función anterior como en el siguiente ejemplo:

```
          mVoice = [6,7,8,9,8,7,6,5,9,8,9,7,8,7,9,12]
            Tvoice = getTvoiceFromMelody(mVoice, 6)

  Tvoice = [4, 4, 7, 7, 7, 4, 4, 4, 7, 7, 7, 4, 7, 4, 7, 11]
```

La flexibilidad que nos da FoxDot en cuanto a escalas y tonalidades, hace que esta implementación sirva para cualquier escala y tonalidad que deseemos.

La esencia del Tintinnabuli en Für Alina

La sencillez de Für Alina puede ser bastante engañosa, ya que requiere de gran técnica y oído para tocarla con la pureza de Pärt. Es por eso que va a ser complicado replicarla de una manera adecuada de forma procedural, pero es una muy buena carta de presentación para introducir el entorno sobre el que vamos a trabajar.

Comenzamos indicándole a FoxDot los parámetros relativos al tempo y la tonalidad que vamos a usar. En cuanto al tempo, vamos a indicar 50 bpm, aunque esta pieza es muy libre en ese sentido e incluso es variable dentro de una misma interpretación, es una buena forma de aproximarnos.

La tonalidad de la pieza sabemos que es Si menor, pero en este caso y para demostrar la flexibilidad que nos ofrece FoxDot y nuestra implementación del Tintinnabuli vamos a utilizar la tonalidad relativa mayor, es decir Re mayor.

Definimos entonces estos dos parámetros:

```
Root.default("D")
Clock.bpm = 60
```

A continuación, vamos a crear una lista con las que serán las duraciones de las notas en la obra. Como hemos visto, tenemos *n* notas de libre duración que van aumentando de número en cada compás y una redonda para terminar. Como esta libre duración es difícil de imitar, vamos a utilizar negras:

```
furAlina = P[1,4].stutter([[1,2,3,4,5,6,7,6,5,4,3,2],[1]])
```

El método stutter se aplica sobre una lista de valores, en nuestro caso P[1,4][6], que van a ser las duraciones de las notas sobre las que vamos a trabajar (negras [1] y redondas [4]) y recibe una lista por cada uno de estos valores con las veces que queremos que se repita cada uno de esos valores, por eso tenemos una lista con varios valores del 1 al 7 y otra con un único valor, porque la redonda solo queremos que se repita una vez por compás. Esto nos devuelve lo siguiente:

```
P[1, 4, 1, 1, 4, 1, 1, 1, ..., 1, 1, 4, 1, 1, 4]
```

Con esto ya podemos definir las 2 voces de la pieza, la principal y la tintinnabular:

```
p1 >> MidiOut(mVoice,
              dur = furAlina,
              channel = 1,
              oct = 6,
              sus = p1.dur-0.1,
              amp = PWhite(0.3, 0.5)
              )
p2 >> MidiOut(getTvoiceFromMelody(mVoice, 0),
              dur = furAlina,
              channel = 2,
              oct = 6,
              sus = p2.dur-0.1,
              amp = PWhite(0.3, 0.5)
              )
```

Para estas voces no estamos utilizando directamente un sintetizador de los predeterminados, si no que utilizamos MidiOut. Con esto hacemos que el player envíe señales MIDI a través de SuperCollider a cualquier secuenciador, en nuestro caso Reaper. Como queremos enviar cada uno de ellos por canales diferentes, con *channel* indicamos el número de canal que queremos utilizar.

6 P[1,4] es lo que en FoxDot llaman patterns, son similares a las listas pero con funciones propias, como es stutter en este caso.

Vamos a hacer un inciso sobre el parámetro "*sus = p1.dur-0.1".* Esto es debido a un conocido bug de FoxDot, documentado oficialmente: *"Note: Be careful when repeating the same note with the same duration; if a MIDI note-on event is triggered slightly before the MIDI note-off for the previous event, it will be stopped by the note-off. This is a known bug and being looked into"* [37].

Esto hace que las notas iguales de la misma duración enviadas por MIDI puedan "solaparse". Definiendo así el "sustain", hacemos que la nota se corte un instante antes de empezar la siguiente y evitemos este bug sin que se note el corte7.

Mvoice es una lista con las notas que queremos interpretar. En nuestro caso hemos definido una lista con las notas originales de la pieza, pero podrían ser cualquier otras o incluso aleatorizarse.

La amplitud (o intensidad) de la nota la hemos aleatorizado entre 0.3 y 0.5 para tratar de darle una dinámica más natural.

Para la voz tintinnabular (p2), hemos usado el método *getTvoiceFromMelody* con la misma melodía que usamos en p1 y aplicándole el Tintinnabuli 0 del relativo menor.

Esta pieza, como está escrita únicamente para piano, vamos a usar solo un plugin vst, el *Keyzone Classic* de *Bitsonic*, que es completamente gratuito. Lo desplegaremos independientemente en 2 pistas, una para la voz principal en el canal 1 y otra en el canal 2 para la voz tintinnabular.

Por último y saliéndonos un poco de la composición original, como base armónica, hemos usado kontakt para generar una rueda de acordes que ambiente la composición. Esta rueda está compuesta por La menor, Mi menor, Do mayor y Re menor.

“Cantus” en una instrucción

```
notes = [comp for i in range(19)
                  for comp in list(range(14,14-i,-1))]
```

Con esta única instrucción generamos la lista de notas que vamos a tocar con todos los instrumentos a diferentes velocidades.

Partiendo del 14, que es la nota más alta de la pieza, vamos añadiendo el siguiente número descendente en cada vuelta, así 19 veces, llegando hasta el -3.

Hemos querido extender la pieza más allá del final original de Pärt para hacer de ella una composición cíclica y con un carácter procedural

Si hacemos un print(notes) en FoxDot obtenemos lo siguiente:

```
[14, 14, 13, 14, 13, 12, 14, 13, 12, 11, 14, 13, 12, 11, 10, 14, 13, 12,
 11, 10, 9, 14, 13, 12, 11, 10, 9, 8, 14, 13, 12, 11, 10, 9, 8, 7, 14, 13,
 12, 11, 10, 9, 8, 7, 6, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 14, 13, 12, 11,
 10, 9, 8, 7, 6, 5, 4, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 14, 13, 12,
11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2,
1, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 14, 13, 12, 11, 10, 9,
 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2,
1, 0, -1, -2, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3]
```

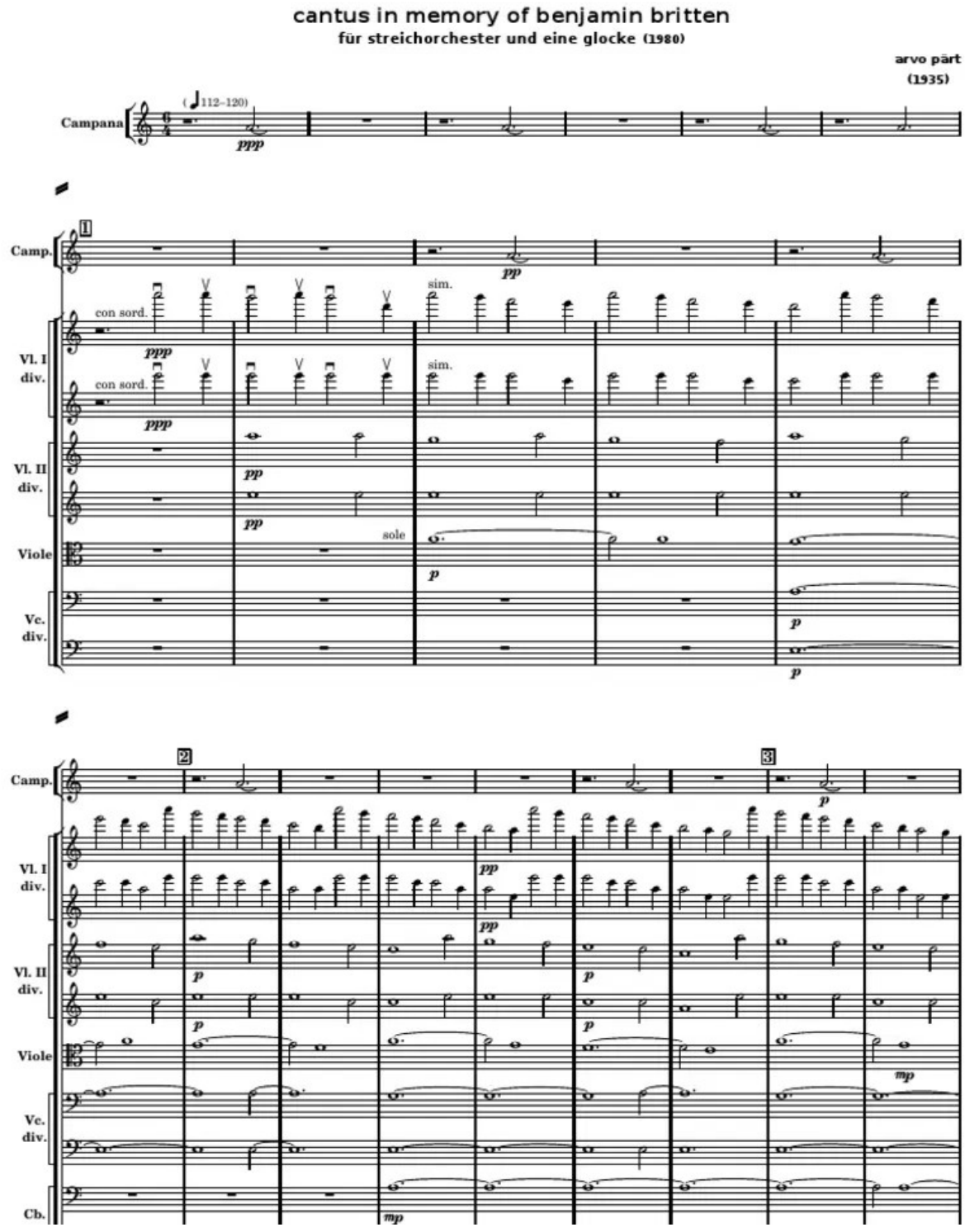


Figura 5: Primera página de partitura original de Cantus in Memoriam Benjamin Britten

```
#Violins
Clock.schedule(lambda: #Comienza en el beat 4
                p1 >> MidiOut(notes, dur=[1,2],
                              oct=5, amp=0.1, channel = 0), Clock.now()+4)
Clock.schedule(lambda: #Comienza en el beat 7
                p2 >> MidiOut(getTvoiceFromMelody(notes, 4), dur = [1,2],
                              oct=5, channel = 1, sus = p1.dur-0.1), Clock.now()+4)
#Violins II
Clock.schedule(lambda:
                v1 >> MidiOut(notes, dur=[2,4],
                              oct = 4, channel = 2), Clock.now()+7)
Clock.schedule(lambda:
                v2 >> MidiOut(getTvoiceFromMelody(notes, 4), dur = [2,4],
                              oct = 4, channel = 3, sus = p1.dur-0.01), Clock.now()+7)
#Violas
Clock.schedule(lambda:
                l3 >> MidiOut(notes, dur=[4,8],
                              oct = 4, channel = 2), Clock.now()+13)
#Cellos
Clock.schedule(lambda:
                c1 >> MidiOut(notes, dur=[8,16],
                              oct = 3, channel = 5), Clock.now()+25)
Clock.schedule(lambda:
                c2 >> MidiOut(getTvoiceFromMelody(notes, 4), dur = [8,16],
                              oct = 3, channel = 6, sus = p1.dur-0.01), Clock.now()+25)
#Contrabasses
Clock.schedule(lambda:
                b1 >> MidiOut(notes, dur=[16,32],
                              oct = 2, channel = 7, sus = p1.dur-0.01), Clock.now()+50)
Clock.schedule(lambda:
                b2 >> MidiOut(getTvoiceFromMelody(notes, 4), dur = [16,32], amp=0.1,
                              oct = 2, channel = 8, sus = p1.dur-1), Clock.now()+50)
```

De esta manera plantearíamos los instrumentos. En primer lugar, con Clock.schedule(), hacemos que cada uno de ellos entre en su pulso correspondiente, como en la obra original.

Es importante remarcar que a cada uno de los instrumentos le hemos asignado una letra, esto es porque si definimos todos los instrumentos como “p”, al haber varios, crean conflicto entre ellos y hay solapamientos. El número 2 siempre corresponderá a la voz Tintinnabular de cada uno de ellos.

Las duraciones, según vamos bajando de registro, se van doblando como en la composición original. De esta manera creamos una composición cíclica de muy larga duración, ya que cada instrumento interpreta las mismas notas, pero a distintas duraciones y ha de pasar mucho tiempo hasta que coincidan en el inicio.

En cuanto a la parte MIDI de esta pieza, hemos utilizado un único plugin VST que tenemos desplegado en varias pistas, una para cada voz de la composición.

El plugin es BB Symphonic Orquesta, un plugin gratuito que emula una orquesta sinfónica en la que podemos elegir por separado la sección que queremos utilizar, lo cual es perfecto para esta pieza.

Figura 6: Representación de las pistas del Cantus en Reaper

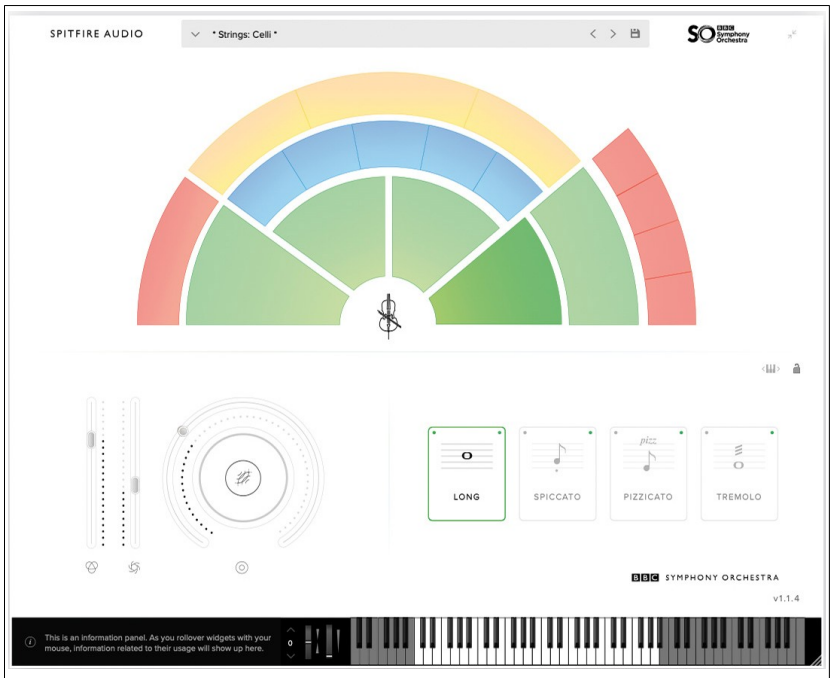


Figura 7: Interfaz del BBC Symphonic Orquestra

La algoritmia de Fratres

Esta obra no vamos a tratar de replicarla lo más fiel a la realidad, sino, que con la técnica que propone Pärt, vamos a introducir una componente aleatoria. Además acercarnos a la interpretación real es complicado, ya que hay múltiples versiones, y en cada una de ellas un instrumento solo hace una melodía diferente encima del motivo que implementaremos y que ya hemos analizado.

Esta aleatoriedad va a estar en el Tintinnabuli de la pieza, donde, en vez de elegir uno de los niveles fijos como hemos hecho con otras, vamos a generar una voz tintinnabular que seleccionará la fila de la matriz aleatoriamente entre +1, +2 y +3 para cada una de las notas de la siguiente manera:

```
def tint(mot):
    mt = []
    inc = 0
    for n in mot:
        if random.random()<0.5: inc=1
        elif random.random()<0.8: inc=2
        else: inc=3
        mt.append(getTvoice(n,inc))
    return mt
```

Donde al t+1 le damos una probabilidad de 0.5, al t+2 de 0.3 y al t+3 de 0.2.

Lo siguiente es generar las notas con sus duraciones, vamos a explicar esto paso por paso.

Lo primero que vamos a hacer es definir una primera función que recibe dos parámetros, el primero es la nota raíz sobre la que queremos generar la línea (como vimos, en la pieza original es un Do#) y el segundo es el número de notas que queremos añadir a esa nota inicial, tanto en descenso como en ascenso:

```
# expansion descendente por grados contiguos + ascendente
def expand(root,ext):
    return [root-i for i in range(ext)] + #descendentes
           [root+ext-i-1 for i in range(ext)] #ascendentes
```

A continuación, vamos a llamar a esta función, dentro de un bucle, las 3 veces que necesitamos para generar los 3 primeros compases de la pieza. Una vez generados esos tres primeros compases, los recorremos en orden inverso para generar los 3 siguientes:

```
ini = [expand(3,i) for i in range(2,5)]#primeras 3 expansiones, 3 compases
fratres = ini + [m[::-1] for m in ini]# añadimos los compases invertidos
```

Estas dos instrucciones podemos meterlas dentro de un for de 9 iteraciones que en cada vuelta reste 3 a la nota raíz. Así generaríamos las 9 repeticiones de la pieza orinal bajando una tercera cada una de ellas:

```
fratres = []
root_note = 3
for j in range(0,9):
    ini = [expand(4-3*j,i) for i in range(2,5)]
    fratres.extend(ini + [m[::-1] for m in ini])
```

Después de esto, lo único que nos falta son las duraciones de las notas, que como hemos visto, son todo negras exceptuando la nota inicial y final de cada compás, que son 2 blancas. De manera que lo que haremos será generar una lista de valores enteros que, para cada compás, su primer valor sea un 2 (el valor de una blanca) seguido de n-2 negras (valor entero 1), donde n es el número de notas del compás y como último valor, nuevamente un 2:

```
durs = [[2] + [1]*(len(c)-2) +[2] for c in fratres]
```

Como con todas estas instrucciones lo que hemos hecho es generar una lista por cada compás, como resultado tenemos una estructura de matriz, es decir, una lista de listas y los *players* de FoxDot solo admiten como parámetro listas simples, por lo tanto, necesitamos convertirlas a ese formato y por último generar la voz tintinnabular llamando a la función tint con el resultado y crear los dos *players* para las dos voces:

```
f = [x for sublist in fratres for x in sublist]
d = [x for sublist in durs for x in sublist]
t = tint(f)

p1 >> MidiOut(f, dur=d, channel = 0)
p2 >> MidiOut(t, dur=d, channel = 1)
```

5 Conclusiones y Trabajo Futuro

Este trabajo, impulsado por una idea básica como lo es el Tintinnabuli, una técnica de contrapunto, ha supuesto un descubrimiento de la obra de Arvo Pärt y un extenso aprendizaje sobre técnicas de composición algorítmica que abren un camino distinto al planteado tradicionalmente en la música clásica y demuestra como la música no siempre necesita excesivos adornos y complejidad para impresionar, a veces, solo es necesario encontrar una buena idea y ejecutarla.

Arvo Pärt elabora un estilo compositivo en el que, a partir de una idea que puede ser una secuencia de notas o un patrón rítmico, es capaz de componer piezas enteras de una riqueza asombrosa. Este es uno de los pilares fundamentales de su minimalismo y ha sido la motivación principal de este trabajo, pudiendo analizar y explicar algunas de sus obras desde un enfoque algorítmico, siendo muy fáciles de entender para un programador no músico.

Históricamente se han aplicado multitud de técnicas algorítmicas o pseudoalgorítmicas en composición musical, aunque los autores no siempre lo hayan hecho de modo deliberado. Otros compositores más recientes sí han aplicado algoritmos para componer música con mejores o peores resultados. El caso de Arvo Pärt es paradigmático porque la algoritmia es muy evidente y la calidad de su obra es indiscutible a pesar de que su intención no parece ser la de componer obras de esta manera para ser generadas o interpretadas con ordenador. Esta es la razón principal por la que elegimos a Pärt para este trabajo, porque si eramos capaces de identificar estos algoritmos y replicarlos en código, podíamos ser capaces de recrear proceduralmente estas obras en su totalidad. Después de haber analizado algunas de sus obras más icónicas y reconocidas, hemos visto como con *Cantus in Memoriam Benjamin Britten* o *Fratres,* hemos sido capaces de generar las obras a partir de una idea algorítmica cumpliendo así el objetivo que nos marcábamos al comienzo de interpretar las piezas de Pärt con el ordenador.

Otro de los objetivos que nos habíamos planteado, era la posibilidad de aplicar las técnicas de Pärt para crear música ambiental que pueda ser utilizada en entornos

donde el oyente simplemente está de paso y no requiere de una escucha activa. Hemos explorado esta idea con las técnicas de Pärt extendiendo algunas de sus composiciones más allá de su final original y convirtiéndolas en composiciones cíclicas, como hemos hecho con el propio *Cantus* o *Für Alina*, además de poder introducir variaciones, modificar algunas de esas reglas o incluso aleatorizar algunos de sus parámetros de una manera coherente y sin alejarnos de la esencia natural de la pieza. Pero es en este punto donde nos hemos encontrado una de las mayores limitaciones al usar el entorno FoxDot.

Una de sus principales ventajas y por la cual lo hemos elegido para este trabajo, es la posibilidad de usar Python en él. Esto nos ha permitido escribir los algoritmos extraídos de las composiciones de una manera muy directa, generando listas estáticas de duraciones y notas muy fácilmente que luego hemos podido hacer sonar. Pero para conseguir una mayor riqueza, necesitábamos poder modificar las piezas en tiempo de ejecución, pudiendo añadir o modificar las notas en tiempo real y, aunque parece que FoxDot tiene algunas herramientas, no hemos sido capaces de usarlas con la precisión necesaria para este fin y realmente dudamos que sea posible de un modo simple.

Es por eso, que uno de los posibles avances futuros puede ser explorar otros entornos de programación musical que aporten una mayor riqueza a la hora de definir algoritmos que puedan variar las composiciones de una manera automatizada. El propio SuperCollider, aunque nos reste facilidad de uso, quizá sea más preciso para este fin. Otra alternativa que nos puede dar este alcance es Pyo [38], un módulo de Python que nos permite la manipulación de señales de audio y utilización de MIDI. Aunque no es tan directo ni tan fácil de usar y entender como FoxDot, puede ser más completo en cuanto a la manipulación de eventos musicales en tiempo real.

Si con estas, u otras herramientas de síntesis musical, alcanzásemos esta capacidad para generar y modificar la música en tiempo real, podríamos utilizar las técnicas de Arvo Pärt en otro tipo de composiciones, fuera de su obra, con un componente más generativo y que nos permita crear música más impredecible.

Bibliografía

1: ¿Es menos más? Artículo sobre el minimalismo. Accesible online: https://www.theguardian.com/education/2001/dec/01/arts.highereducation2

2: Minimalismo Sacro. Artículo de Lumen Learning. Accesible online: https://courses.lumenlearning.com/atd-epcc-musicappreciation/chapter/holy-minimalism/

3: Obras de Arvo Pärt. Artículo de Wikipedia. Accesible online: https://es.wikipedia.org/wiki/anexo:composiciones_de_arvo_p%c3%a4rt

4: Entrada de Arvo Pärt en Internet Movie Database Accesible online: https://www.imdb.com/name/nm0701736/

5: Encaenia y Honoris Causa 2016. Accesible online: https://www.ox.ac.uk/news-and-events/the-university-year/encaenia/2016

6: El compositor contemporáneo más interpretado 8 años seguidos. Artículo de Aleteia. Accesible online: https://es.aleteia.org/2020/02/03/el-compositor-contemporaneo-mas-interpretado-8-anos-seguidos/

7: Arvo Pärt, creador del 'tintinnabuli' y autor de bandas sonoras para Sorrentino o Erice, premio Fundación BBVA. Artículo de europapress. Sección epCultura. Accesible online: https://www.europapress.es/cultura/musica-00129/noticia-arvo-part-creador-tintinnabuli-autor-bandas-sonoras-sorrentino-erice-premio-fundacion-bbva-20200331131758.html

8: Biografía de Arvo Pärt. Artículo en Britannica. Accesible online: https://www.britannica.com/biography/arvo-part

9: Tintinnabuli. Entrada de la Wikipedia. Accesible online: https://en.wikipedia.org/wiki/tintinnabuli

10: Für Alina. Enlace de YouTube a la pieza Accesible online: https://www.youtube.com/watch?v=jvxy69ef__y&ab_channel=shiraishinal

11: Capítulo sobre Arvo Pärt. Programa de Música y Significado en RNE. Accesible online: https://www.rtve.es/play/audios/musica-y-significado/musica-significado-arvo-prt-fratres-13-03-15/3042738/

12: Juego musical de dados. Artículo de Wikipedia. Accesible online: https://es.wikipedia.org/wiki/musikalisches_w%c3%bcrfelspiel

13: Karen Collins (2009) An Introduction to Procedural Music in Video Games, Contemporary Music Review, 28:1, 5-15, DOI: 10.1080/0749446080266398. .

14: Brian Eno y la Música Generativa. Artículo. Accesible online: https://ilpojauhiainen.com/writings/generative-music/

15: Collins, N. (2008). The Analysis of Generative Music Programs. Organised Sound, 13(3), 237-248. Accesible online: https://doi.org/10.1017/s1355771808000332

16: Brian Eno y la música ambient. Artículo en Sulponticello. Accesible online: https://sulponticello.com/iii-epoca/brian-eno-y-la-musica-ambient-1/

17: Für Alina en Sonic Pi. Repositorio de GitHub Accesible online: https://github.com/christleijtens/sonic_pi_fur_alina

18: Tintinnabulator. Enlace a la web del autor. Accesible online: https://www.miltonline.com/2020/03/30/tintinnabulator/

19: Tintinnabuli Mathematica. Artículo web del autor. Accesible online: https://aestheticcomplexity.wordpress.com/2011/11/11/programming-arvo-part/

20: Sinfonía No1 - Arvo Pärt. Enlace a la pieza en Youtube. Accesible online: https://www.youtube.com/watch?v=dah4cbw389k&t=369s&ab_channel=funkedupeast

21: Ennsayo White Light por Hermann Conen, traducido al inglés por Eileen Walliser-Schwarzbart (encontrado en las notas del álbum de ECM Alina). .

22: Extraído de una entrevista con Anthony Pitt grabada para BBC Radio 3 en la Real Academia de Música en Londres el 29 de marzo de 2000

23: Cantus in Memoriam Benjamin Britten. Enlace de Youtube a la obra. Accesible online: https://www.youtube.com/watch?v=kimkbj1jqfu&ab_channel=musicnetmaterials

24: Fratres. Enlace de Youtube a la obra. Accesible online: https://www.youtube.com/watch?v=7vdgzajvnes&ab_channel=pelodelperro

25: Fratres. Entrada en la Wikipedia donde se detallan sus distintas versiones. Accesible online: https://en.wikipedia.org/wiki/fratres

26: Spiegel im Spiegel. Artículo de la Wikipedia donde aparecen algunos de sus usos en cine y televisión. Accesible online: https://en.wikipedia.org/wiki/spiegel_im_spiegel

27: Spiegel im Spiegel. Enlace de Youtube a la obra. Accesible online: https://www.youtube.com/watch?v=tj6mzvh3xcc&ab_channel=playingmusiconmars

28: Partitura de Spiegel im Spiegel. Partitura interactiva en Musescore. Accesible online: https://musescore.com/user/28369848/scores/7070765

29: Lista de Software Musical. Entrada en la Wikipedia. Accesible online: https://en.wikipedia.org/wiki/list_of_music_software

30: Documentación de FoxDot. Web oficial. Accesible online: https://foxdot.org/

31: TopLap. Foro especializado en LiveCoding. Accesible online: https://forum.toplap.org/

32: Pagina web oficial de SuperCollider. Accesible online: https://scsynth.org/

33: Client vs Server. Explicación sobre la arquitectura interna de SuperCollider. Accesible online: https://doc.sccode.org/guides/clientvsserver.html

34: Definición de MIDI. Entrada de la Wikipedia. Accesible online: https://es.wikipedia.org/wiki/midi

35: Reaper. Página oficial con documentación y tutoriales. Accesible online: https://www.reaper.fm/

36: LoopMIDI. Web oficial del autor. Accesible online: https://www.tobias-erichsen.de/software/loopmidi.html

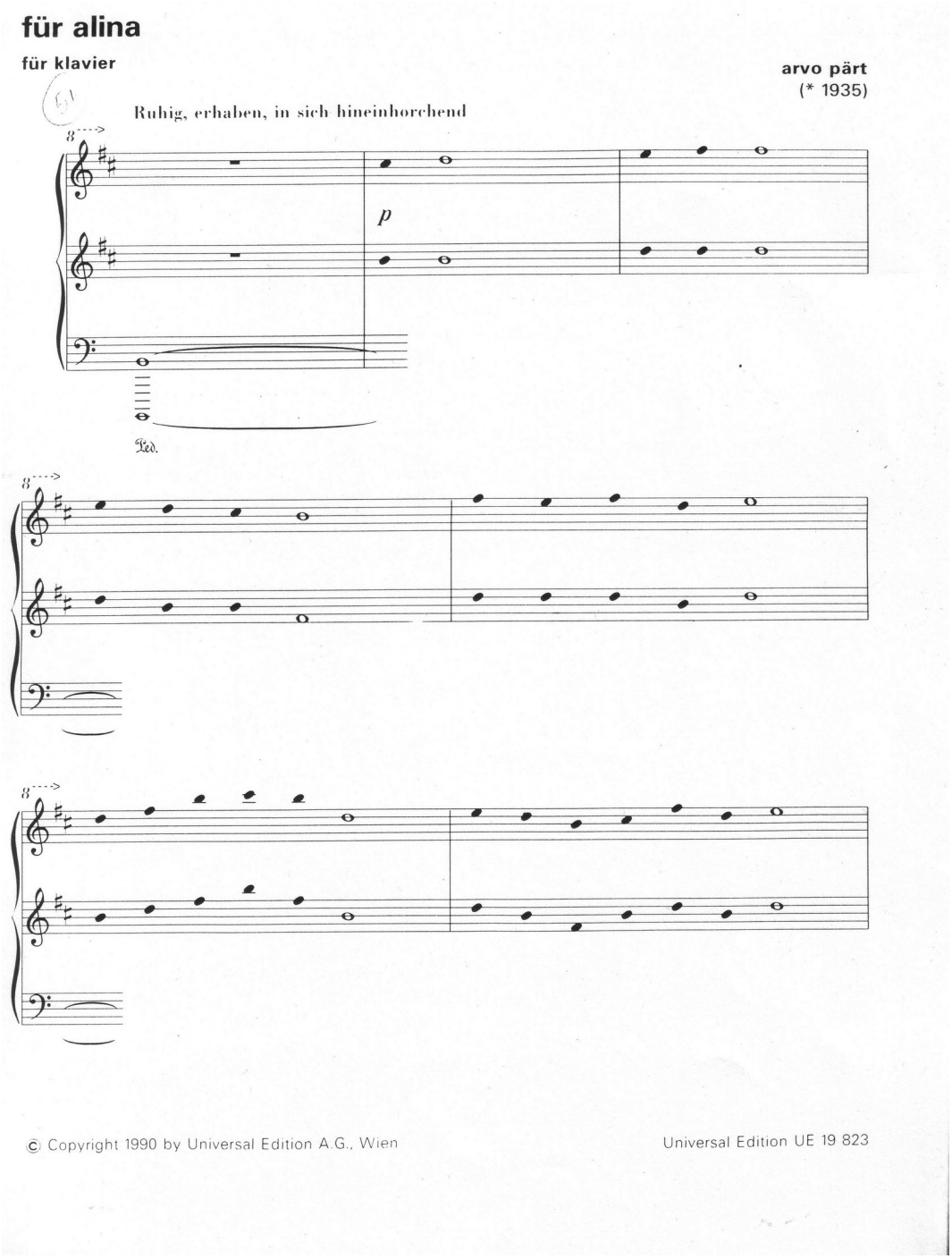
37: Documentación de FoxDot. Página con la referencia al bug mencionado. Accesible online: https://foxdot.org/docs/setting-up-midi/

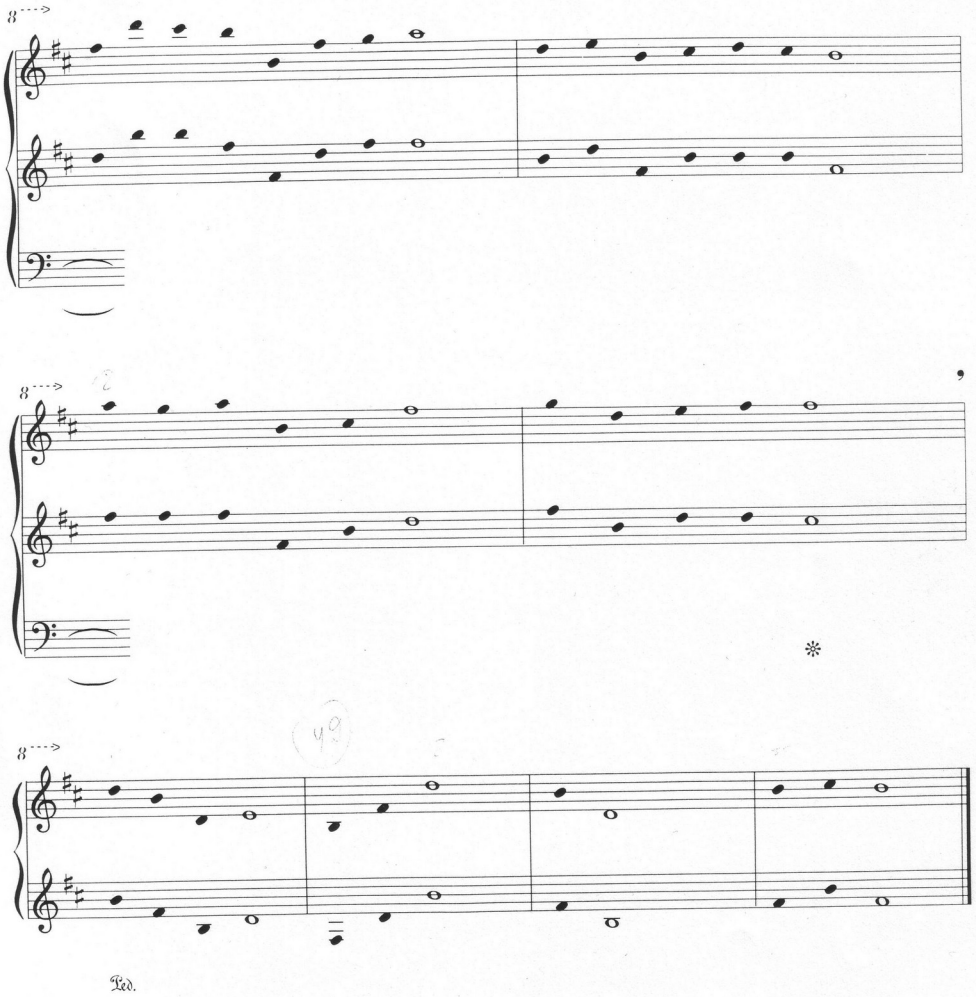
38: Pyo. Módulo de Python Accesible online: https://pypi.org/project/pyo/

Anexos

Anexo A

Aquí incluimos la partitura original de Für Alina.




8
8
8
Ped.