Escuela
Politécnica
Superior

Estudio e implementación de métodos de composición algorítmica con propósitos exploratorios

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
Laura Sirvent Collado
Tutor/es:
José Manuel Iñesta Quereda

Septiembre 2014

Universitat d'Alacant
Universidad de Alicante

Prólogo

Este trabajo comienza a partir de una propuesta de José Manuel Iñesta de crear una aplicación fácil de aprender a utilizar que contenga una implementación básica de los principales métodos de composición algorítmica, con un propósito explorativo y educativo, que permita reconocer las ventajas, inconvenientes y posibilidades de cada uno de estos métodos.

En los capítulos que se suceden a continuación se presentará un estudio teórico de cada uno de los métodos de composición algorítmica, y posteriormente su diseño e implementación a los problemas específicos que introducen cada uno de ellos dentro del marco de la aplicación. También se profundizará en el diseño general de la aplicación, en términos de usabilidad por parte del usuario. Finalmente se expondrán los resultados y las conclusiones del trabajo.

Por último comentar que todo el contenido relativo a este trabajo (código fuente, documentación, memoria, ejemplos) se encuentra alojado en el servidor del Departamento de Lenguajes y Sistemas Informáticos, accesible desde la dirección `http://grfia.dlsi.ua.es/cm/projects`.

Agradecimientos

Ya que se me da la oportunidad de hacerlo en este pequeño espacio, quiero dar las gracias a muchos que me han apoyado y ayudado desde el principio.

A mi tutor José Manuel Iñesta, por proponerme una idea suficientemente motivadora como para llevarla a cabo y acabar más que satisfecha con el resultado. A todos con los que me he cruzado, con los que he aprendido algo y de los que me llevo muy buen recuerdo de estos cuatro años en la Universidad de Alicante.

Y principalmente, a mis padres y a Patri, gracias por toda la ayuda y apoyo para conseguir guiarme en el camino, sin ellos hoy no estaría aquí; a Carmen, por estar ahí desde que tengo memoria, y hacerme sentir más que bien; y a Raúl, por estar todo este tiempo inspirándome y animándome a seguir adelante con todo en general y este trabajo en particular. Gracias a vosotros cinco por hacer atreverme con cualquier cosa que me proponga.

Índice general

Índice de figuras

Índice de cuadros

1 Introducción

1.1 Presentación

La informática ha cambiado en relativamente poco tiempo la forma de ver todo lo que rodea a las personas. Hoy en día es utilizada para casi todo, nos ayuda y asiste en tareas que antes ni imaginábamos hacer de otra forma que no fuera la manual.

Por otra parte, la composición algorítmica es una técnica y área de estudio que investiga formas de generar composiciones musicales utilizando métodos formales. Aunque siempre se haya considerado la música como algo artístico y fuertemente relacionado con las emociones, y su proceso de composición como un procedimiento enteramente creativo sólo alcanzable por las capacidades mentales del ser humano, realmente sí puede establecerse una diferencia en el mismo. Se puede distinguir entre la denominada *composición musical instantánea*, como por ejemplo el flamenco, guiada por la intuición y la capacidad de improvisación del ser humano (siempre condicionado por la educación musical que ha recibido), y la *composición musical regida por normas y reglas musicales*, como el caso de la música clásica occidental, dirigida por un conjunto de reglas teóricas (teoría musical, armonía, orquestación, etc), que deben ser estudiadas previamente a la composición.

A lo largo de la historia siempre se ha querido formalizar la técnica de composición musical, como en el *Arte de la Fuga* de Johann Sebastian Bach, o el *Dodecafonismo* ideado por Arnold Schöenberg.

Parece, y es lógico e inevitable, que estas formalizaciones de composición musical sean muy susceptibles de ser aproximadas al marco de la computación. Por esto, la idea de poder convertir las reglas teóricas que dirigen la composición musical a un conjunto de procesos sistematizados y automáticos

ha sido fuente de inspiración desde los comienzos de la computación. Uno de los primeros hitos en la composición automatizada se produce alrededor del año 1840, por Ada Lovelace, considerada pionera de la programación, al querer mecanizar la *Máquina analítica* [1], y utilizarla para componer fragmentos musicales.

El desarrollo por su parte de la computación ha ido a lo largo de los años creando sistemas y equipos de cálculo cada vez más sofisticados, veloces y cercanos a cualquier persona. Atrás han quedado los primeros ordenadores que ocupaban habitaciones enteras y que sólo estaban al alcance de instituciones con alto poder adquisitivo. Hoy en día la informática se encuentra en un punto alcanzable por cualquier persona, gracias al desarrollo de formas de comunicarse y enlazar con la misma, como el uso de las interfaces de usuario. [2]

Debido a esta evolución en el campo de la informática, existen actualmente infinidad de investigaciones que unen la composición musical con la computación, desarrollándose sistemas de composición algorítmica que reúnen las reglas musicales que se han mencionado anteriormente. Estos sistemas pueden diferenciarse por el grado de intervención humana que necesita. Puede tratarse de sistemas de **composición automática**, que generan composiciones musicales utilizando métodos formales y computables, y que casi no necesitan de intervención humana; y sistemas de **composición asistida**, que consiste en herramientas computacionales que ayudan a la persona en el proceso de composición algorítmica, generando valores parametrizables (alturas, ritmos, dinámica), o materiales básicos, como pequeños fragmentos musicales que pueden ser utilizados como punto de partida a una composición mayor.

Un sistema de composición algorítmica puede reunir las reglas teóricas que rigen la música, que pueden ser explicadas desde un punto de vista computacional como un conjunto de operaciones y reglas matemáticas aplicadas a los elementos que componen un fragmento musical. También puede construirse como una simulación del proceso creativo que sigue una persona al componer música.

Actualmente la gran mayoría de estos sistemas tienen una gran complejidad desde el punto de vista de uso, lo que hace muy complicado aprender a utilizarlo por parte de usuarios no experimentados. El presente proyecto plantea un sistema de composición asistida que pueda facilitar al usuario fragmentos musicales, ayudarle en el proceso de composición a partir de generar motivos o pequeños fragmentos de cualquier longitud, tanto melódicos como rítmicos y que sirvan como inspiración, a partir de algunos de los métodos formales que se han ido ideando a lo largo del tiempo en la investigación de la composición algorítmica. Todo ello dentro de un programa fácil de comprender y utilizar por casi cualquier persona que se le presente.

[1] Diseño de un motor de cálculo, ideado y descrito por Charles Babbage en el año 1817.

[2] Forma que tiene el usuario de comunicarse con el ordenador, que tengan la capacidad de ser fáciles de comprender y utilizar, intuitivas y amigables para con el ser humano.

El proyecto puede servir además como forma de apreciar las bondades y visualizar las desventajas que presentan cada uno de los métodos de composición algorítmica que van a estudiarse, para así comprender y saber en qué ocasiones un determinado método de composición algorítmico puede ser totalmente válido en la composición musical.

1.2 Objetivos y motivación

Este proyecto tiene como principal motivación intentar acercar la composición algorítmica al público general, sin necesidad de tener conocimientos técnicos sobre informática, y que sirva como medio de inspiración, o como sistema a través del cual pueda crear música que directamente pueda utilizar.

Otro objetivo del proyecto es unir en un único sistema los diferentes métodos de composición algorítmica que existen, para poder así aprovecharse de las bondades de cada uno de ellos, y que pueda servir a los usuarios a idear nuevas formas de componer música a través de este sistema.

Además de ayudar en el proceso de la enseñanza de la composición algorítmica, suministrando a los usuarios un sistema que sea capaz de dar ejemplos sencillos sobre los métodos de composición algorítmica que existen.

Para ello se diseñará e implementará un sistema que contenga distintas formas de composición algorítmica que se han estudiado a lo largo de los años, y que sea capaz de generar música en tiempo real, con posibilidad de guardar en un fichero la composición en formato MIDI.

1.3 Estructura de la memoria

La memoria del presente proyecto consta de 6 capítulos. El contenido de los mismos es el siguiente:

- **Capítulo 1**: Presentación del proyecto.
- **Capítulo 2**: Explicación breve de los conocimientos previos necesarios y reseña histórica.
- **Capítulo 3**: Análisis previo del sistema a implementar, solución que se propone y flujo de trabajo de la aplicación.
- **Capítulos 4**: Diseño del sistema, descripción de cada una de las partes de la aplicación, detalles de la implementación y el uso de la misma.
- **Capítulo 5**: Resultados y ejemplos obtenidos, y evaluación del sistema.
- **Capítulo 6**: Conclusiones finales.

2
Conocimientos previos

Antes de comenzar a explicar el desarrollo del proyecto es importante tener ciertos conocimientos sobre el campo en el que se desarrolla el mismo. En primer lugar, se explicará brevemente unas ideas básicas sobre teoría musical, para posteriormente explicar los apartados técnicos del proyecto.

2.1 Teoría musical

Se explicarán algunos conceptos sobre teoría musical y representación de la música que se consideran necesarios para el entendimiento del proyecto.

2.1.1 Definiciones

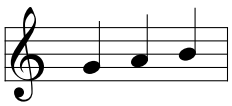
La representación de la música se realiza a través de signos gráficos, que definen a la vez su duración a través de la horizontalidad, y su altura a través de su verticalidad.

- Pentagrama: Base sobre la que se representan los signos musicales. Son cinco líneas horizontales equidistantes entre sí.
- Clave: Signo que define la altura de las notas que se escribirán en el pentagrama. Las más frecuentes son la clave de Sol y la clave de Fa en cuarta.

  En la clave de Sol, la nota ubicada en la segunda línea del pentagrama representa a la altura Sol.

En la clave de Fa en cuarta, la nota ubicada en la cuarta línea del pentagrama representa a la altura Fa.

Como ejemplo, la figura 2.1 contiene un pentagrama con tres notas representadas en clave de Sol.



Sol

Figura 2.1: Notas en clave de Sol. La nota en la segunda línea se interpreta como Sol.

- Nota: Signos que se escriben en los espacios o líneas del pentagrama y representa una determinada altura musical.
- Figura: Representa la duración de la nota, a través de un determinado signo gráfico. El uso de las figuras define el ritmo. Se toma como unidad la figura *redonda*, teniendo las figuras que le suceden una duración de la mitad de su predecesora.
- Silencio: Signo que representa la ausencia del sonido durante un lapso de tiempo. El lapso de tiempo viene dado por el tipo de figura que representa el silencio. La duración de cada silencio sigue el mismo patrón que la duración de las figuras.

  La tabla 2.1 contiene todas las figuras existentes, sus correspondientes silencios y sus valores, tomando como figura unidad la redonda.

Nombre	**Figura**	**Silencio**	**Valor**
Redonda	𝅝	𝄻	1
Blanca	𝅗𝅥	𝄼	$\frac{1}{2}$
Negra	♩	𝄽	$\frac{1}{4}$
Corchea	♪	𝄾	$\frac{1}{8}$
Semicorchea	𝅘𝅥𝅯	𝄿	$\frac{1}{16}$
Fusa	𝅘𝅥𝅰	𝅀	$\frac{1}{32}$
Semifusa	𝅘𝅥𝅱	𝅁	$\frac{1}{64}$

Cuadro 2.1: Figuras existentes en la notación musical

- Alteración: Signo cuya finalidad es modificar la altura de la nota que lo sucede durante todo un compás.

  - Sostenido ♯: Aumenta en un semitono la altura de la nota.
  - Bemol ♭: Disminuye en un semitono la altura de la nota.
  - Becuadro ♮: Devuelve la altura natural de la nota.

- Compás: Unidad de tiempo en la que se divide la obra musical. Corresponde a la mínima estructura temporal posible en términos rítmicos.

- Línea divisoria: Una línea perpendicular al pentagrama indica el final de un compás y el inicio del siguiente.

- Tiempos del compás: Divisiones temporales de igual duración contenidas dentro de un compás.

- Métrica: Se trata de una fracción que representa la medida del compás, siendo el numerador de dicha fracción la cantidad de tiempos por compás, y el denominador el valor de cada uno de los tiempos en relación a la figura unidad (la redonda). En la figura 2.2 se aprecia un fragmento musical de tres compases, de métrica $\frac{4}{4}$, donde cada uno de los compases se completa por la combinación de figuras que sumen el total de una redonda.

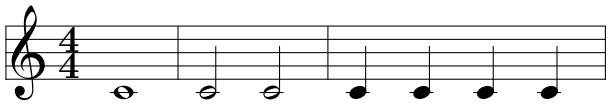


Figura 2.2: Fragmento con métrica $\frac{4}{4}$.

- Partes del compás: Cada uno de los tiempos del compás tiene su propia clasificación, que puede ser: fuerte, débil o semifuerte.

  Por norma general, el primer tiempo del compás siempre es fuerte, y los posteriores débiles, con exepción del tercer tiempo en un compás de cuatro tiempos, que se clasifica como semifuerte.

- Tempo: Es el grado de velocidad con el que se debe interpretar una obra musical. Existen dos clasificaciones del movimiento:

  - Especificación tradicional: Usando términos italianos (*andante, presto, allegro*, etc).
  - Especificación *no clásica*: A través de un número que indica el número de figuras por minuto, siendo esta figura generalmente la que representa el valor de un tiempo (ver figura 2.3). También se le conoce como *movimiento*.

- Signos de expresión: Sirven para representar matices a la hora de interpretar una obra:

Figura 2.3: Ejemplo de clasificación no clásica, o tempo.

- Dinámicos: Indican la intensidad de sonido con la que debe interpretarse un fragmento o frase musical de la obra. Se clasifican de menor a mayor intensidad con expresiones en italiano. La tabla 2.2 contiene algunas de las expresiones que indican la intensidad del sonido.

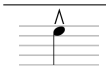
Signo	**Nombre**	**Valor**
ppp	Pianissimo	Más suave
pp	Molto piano	Muy suave
p	Piano	Suave
mp	Mezzo piano	Medio suave
mf	Mezzo forte	Media fuerte
f	Forte	Fuerte
ff	Molto forte	Muy fuerte
fff	Fortissimo	Más fuerte

Cuadro 2.2: Expresiones italianas para la dinámica de la obra o fragmento musical.

- Articulaciones: Signos colocados encima de una nota y que indican la forma en la que deben *atacarse*. La tabla 2.3 contiene las articulaciones posibles sobre una nota.

- Tonalidad: Conjunto de sonidos cuyo funcionamiento queda regido por un sonido principal llamado *tónica*. Los sonidos que componen la tonalidad son siete, conocidos como *grados*. Una tonalidad puede tener varios *modos*, que son básicamente dos, conocidos como Mayor y Menor.
- Grado: Cada uno de los sonidos que componen una tonalidad. Se corresponden con los nombres de las siete notas musicales y se identifican con números romano, además de un nombre propio (ver tabla 2.4).

  - Grados tonales: Son los grados que definen una tonalidad, y son: I, IV y V (tónica, subdominante y dominante).
  - Grados modales: Son los grados que definen el modo de una tonalidad, y son: III principalmente, y II, VI y VII (mediante, supertónica, superdominante y sensible).

Signo	Articulación
	Ataque percusivo, la nota mantiene todo su valor
	Ataque percusivo, la nota disminuye su valor $\frac{2}{3}$.
	Stacatto, la nota disminuye su valor a la mitad.
	Legato, la nota se acentúa y mantiene todo su valor.

Cuadro 2.3: Signos para la articulación de las notas.

Grado	Nombre
I	Tónica
II	Supertónica
III	Mediante
IV	Subdominante
V	Dominante
VI	Superdominante
VII	Sensible

Cuadro 2.4: Nomenclatura de los grados de una tonalidad

- Modo: Un modo o modalidad hace referencia a la manera de *ser* de una tonalidad, o lo que es lo mismo, a las distancias entre los grados de la misma. Los modos existentes son: *Jónico, Dórico, Frigio, Lidio, Mixolidio, Eolio* y *Locrio*. De entre estos, los más utilizados actualmente son el modo Mayor (figura 2.4), que se corresponde con el modo Jónico, y el modo Menor (figura 2.5), que se corresponde con el modo Eolio.

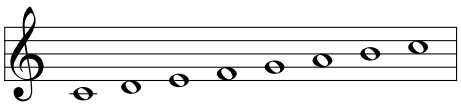


Figura 2.4: Modo Mayor o *Jónico*.

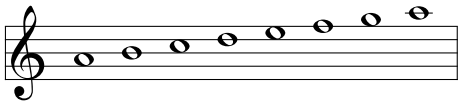


Figura 2.5: Modo Menor o *Eolio*.

2.1.2 Intervalos

La diferencia entre alturas de sonidos es lo que hace que reconozcamos una melodía o un fragmento musical, independientemente de la altura absoluta a la que se encuentren estos sonidos. Por ello, es muy importante en la música y en la composición musical el conocimiento de los intervalos.

Se conoce como intervalo a la distancia en altura entre dos sonidos musicales. En la música occidental, la unidad mínima de medida de altura es el semitono, por lo que la distancia entre alturas es medida en semitonos (o tonos, donde un tono es la suma de dos semitonos).

Clasificación de intervalos

Los intervalos se clasifican según la cantidad de grados que hay entre los dos sonidos que lo forman, incluyéndolos, y además, según la distancia en semitonos que existe entre los sonidos. Es decir, se utiliza para clasificar de igual forma la cantidad de grados y de semitonos.

De esta forma, se crea una nomenclatura la clasificación de intervalos. La tabla 2.5 contiene los nombres de cada uno de los intervalos posibles.

Además, hay que matizar la naturaleza de un intervalo a través de los siguientes términos:

- Intervalo ascendente: El segundo sonido que lo forma tiene más altura que el primero.
- Intervalo descendente: El segundo sonido que lo forma tiene menos altura que el primero.
- Intervalo conjunto: El intervalo se produce entre dos grados inmediatos.
- Intervalo disjunto: En el intervalo hay más de un grado entre los sonidos que lo forman.
- Intervalo simple: Los sonidos que forman el intervalo no guardan una distancia entre sí de más de ocho notas.
- Intervalo compuesto: Existe una distancia mayor a ocho notas en el intervalo.

Nombre	Grados	Distancia
Segunda menor	I - II	Un semitono
Segunda mayor	I - II	Un tono
Segunda aumentada	I - II	Un tono y un semitono
Tercera disminuida	I - III	Dos semitonos
Tercera menor	I - III	Un tono y un semitono
Tercera mayor	I - III	Dos tonos
Tercera aumentada	I - III	Dos tonos y un semitono
Cuarta disminuida	I - IV	Dos tonos
Cuarta justa	I - IV	Dos tonos y un semitono
Cuarta aumentada	I - IV	Tres tonos
Quinta disminuida	I - V	Tres tonos
Quinta justa	I - V	Tres tonos y un semitono
Quinta aumentada	I - V	Cuatro tonos
Sexta disminuida	I - VI	Tres tonos y un semitono
Sexta menor	I - VI	Cuatro tonos
Sexta mayor	I - VI	Cuatro tonos y un semitono
Sexta aumentada	I - VI	Cinco tonos
Séptima disminuida	I - VII	Cuatro tonos y un semitono
Séptima menor	I - VII	Cinco tonos
Séptima mayor	I - VII	Cinco tonos y un semitono
Séptima aumentada	I - VII	Seis tonos
Octava disminuida	I - VIII	Cinco tonos y un semitono
Octava justa	I - VIII	Seis tonos
Octava aumentada	I - VIII	Seis tonos y un semitono

Cuadro 2.5: Clasificación de los intervalos, en función de los grados y la distancia en semitonos entre los dos sonidos que lo componen.

2.1.3 Escalas

Se conoce como escala a un conjunto de sonidos dispuestos de forma ascendente en altura, dentro de un entorno sonoro particular. Cada uno de los sonidos de la escala se corresponde a un grado, tal y como se explicó anteriormente (ver 2.1.1, definición de tonalidad).

En ocasiones los términos tonalidad y escala suelen confundirse y tomarse como sinónimos, por lo que es necesario explicar la pequeña diferencia que los distingue:

- Tonalidad: Es un conjunto de sonidos regidos por un sonido principal (grado I o tónica).
- Escala: Se refiere a la sucesión de sonidos dispuestos (pertenecientes a

una tonalidad o no) de forma ascendente o descendente en altura.

Es decir, en una tonalidad no importa el orden en el que los sonidos estén dispuestos, mientras que en una escala sí. Además de que una escala puede ser tonal o no.

Tipos

Existen diferentes tipos de escalas musicales; aunque la escala diatónica sea la más conocida, es interesante conocer algunos tipos más de escalas, y entender sus bases matemáticas.

Escala diatónica

La escala diatónica está formada por siete sonidos, los cuales se suceden entre sí por intervalos de segunda. Dentro de este concepto se distinguen dos modos de escalas diatónicas, mayor y menor, teniendo cada una de ellas una escala natural diatónica, que se utiliza para saber a que distancia está cada grado con el siguiente. Cada uno de los modos tiene una escala natural, por la que podemos conseguir dicho modo sin aplicar alteraciones a las notas.

Modo mayor: En el modo mayor los grados se separan entre sí de forma que hay un tono de separación entre todos los grados excepto del III al IV y del VII al I, en los que hay un semitono de separación. La escala natural del modo mayor es aquella con su tónica en Do, es decir, la escala de Do.

Para poder conseguir una escala diatónica mayor con una nota diferente a Do como tónica, han de utilizarse alteraciones sobre algunas de las notas, para así conseguir las separaciones necesarias definidas en la escala mayor. Este conjunto de alteraciones se denomina **armadura**, y se sitúa inmediatamente después de la clave en el pentagrama, con efecto continuo hasta el final de la obra. La colocación de estas alteraciones se realiza siguiendo el *círculo de quintas* (figura 2.6). Por ejemplo, si se quiere la escala de Re Mayor, a través del círculo se puede ver que se necesita aplicar dos alteraciones, en Fa y Do, para poder conseguir dicha escala (figura 2.7).

Modo menor: Por otra parte, en el modo menor los grados se separan de forma que hay un tono de separación entre todos los grados excepto del II al III y del V al VI, en los que hay un semitono de separación. En este caso, la escala natural del modo mayor es aquella en la que el primer grado o tónica es la nota La.

Dentro del modo menor se distinguen dos escalas más aparte del modo menor natural, estas son la **escala menor armónica**, donde el grado VII está aumentado un semitono, y la **escala menor melódica**, que tiene el grado VI y VII aumentados en un semitono sólo cuando la escala se interpreta de forma ascendente, y al ser interpretada de forma descendente, estos grados vuelven a su altura natural.

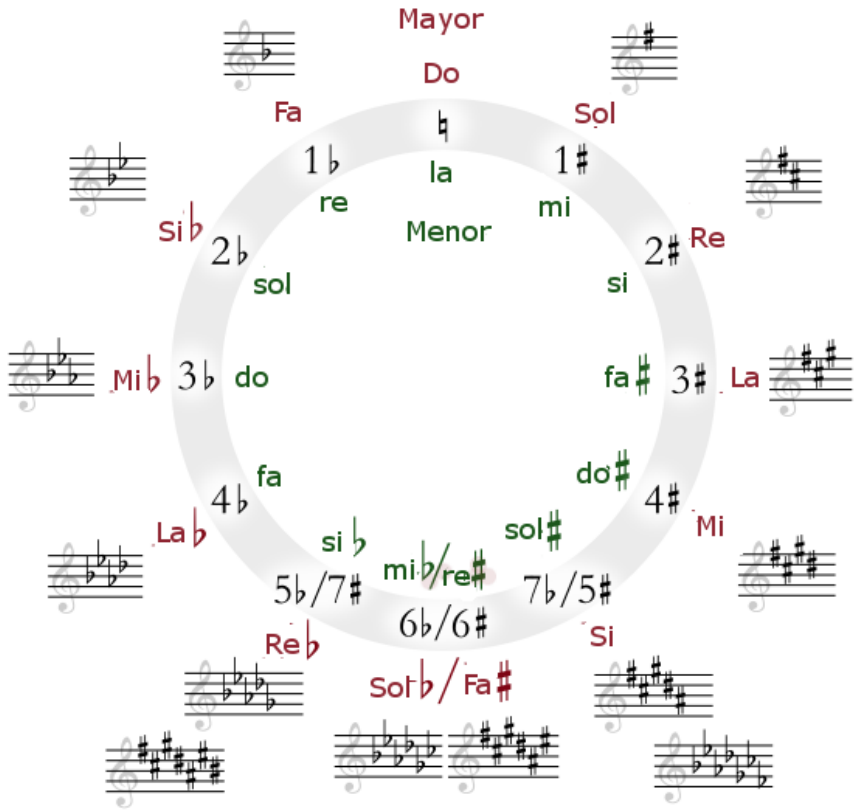


Figura 2.6: Círculo de quintas.

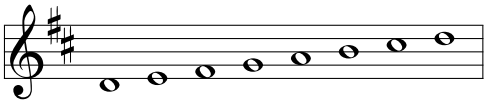


Figura 2.7: Escala de Re Mayor.

Existe un último concepto en las escalas diatónicas; cada escala mayor tiene su escala relativa menor, que utiliza la misma armadura para definirse. Esto puede verse también en el círculo de quintas (figura 2.6). Por ejemplo, el modo menor relativo de La Mayor, y que por tanto tiene tres sostenidos en su armadura es Fa# Menor.

Escala cromática

La escala cromática, o también conocida como escala *dodecáfona* es aquella que está construida por una sucesión de doce sonidos separados entre

sí por semitonos. Este tipo de escala no se rige por una nota tónica, ni se distinguen en ella grados tonales o modales, ya que en la escala cromática todos los sonidos tienen igual importancia, debido a que todos ellos están separados a igual distancia. Por lo tanto, cualquiera de los sonidos que forman esta escala puede ser tomado como tónica. Puede verse un ejemplo de escala cromática en la figura 2.8.

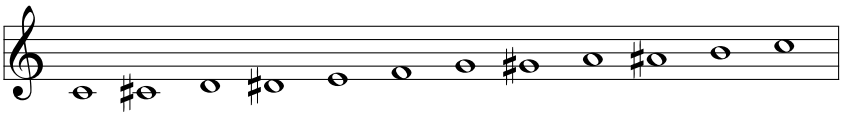


Figura 2.8: Escala cromática ascendente.

Escala pentatónica

La escala pentatónica es otro tipo de escala que está construida sobre cinco sonidos separados entre sí por dos semitonos excepto entre el grado I y II, que está separado por tres semitonos, y el grado IV y V, separado también por tres semitonos. Esta es la escala pentatónica menor. En el caso de querer construir la escala pentatónica mayor, se sigue el mismo patrón de sucesiones pero comenzando en el segundo grado, es decir, el segundo sonido de la escala pentatónica menor se corresponde con el primer sonido de su versión mayor. En la figura 2.9 se pueden apreciar dos escalas pentatónicas, mayor y menor, con tónica en Do.

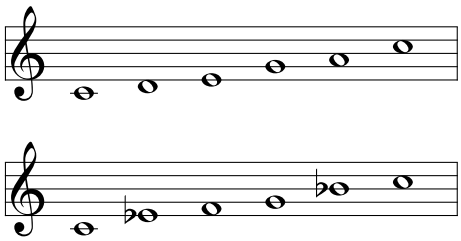


Figura 2.9: Escalas pentatónicas (mayor y menor) en Do.

Escala de blues

La escala de blues es una escala derivada de la escala pentatónica menor, a partir de añadirle un intervalo de quinta disminuida a la tónica de dicha escala. La escala de blues es, por lo tanto, una escala conformada por seis tonos (hexatónica). Esta escala es característica del estilo musical blues.

Como ejemplo, si se toma la escala pentatónica menor en Do, para convertirla en escala de blues, hay que añadir un intervalo de quinta disminuida, en este caso, Sol bemol, obteniendo así la escala de blues en Do (figura 2.10).

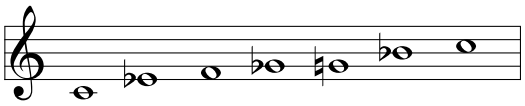


Figura 2.10: Escala de blues en Do.

Otras escalas

A continuación se enumeran algunas escalas denominadas *artificiales*, por ser producto de la introducción de sonidos alterados a una escala natural.

- Escala Hispano-árabe
- Escala Enigmática
- Escala Acústica
- Escala Locria-Mayor

2.1.4 Acordes

Se conoce como acorde a un conjunto de sonidos distintos que se interpretan simultáneamente. Generalmente, el acorde se forma a partir de intervalos sobre una nota base, conocida como **raíz, tónica o fundamental**.

Acorde perfecto o tríada: El acorde formado por tres notas separados por intervalos de tercera consecutivos se denomina acorde perfecto o tríada, mayor o menor, en función de la distancia a la que se encuentren el grado I con el III (figura 2.11).

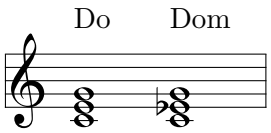


Figura 2.11: Tríadas de Do mayor y Do menor.

En función de tipo de intervalo entre el grado I y el III y el I y el V, los acordes tríadas pueden clasificarse en tres grupos:

- Mayor: Intervalo de tercera mayor y quinta justa.
- Menor: Intervalo de tercera menor y quinta justa.
- Disminuido: Intervalo de tercera menor y quinta disminuida.

Inversión de tríadas: Un mismo acorde puede presentarse en su estado fundamental (la nota raíz es la de menor altura) o invertido (cualquiera de las demás notas es la de menor altura). Aún estando invertido, el acorde conserva su clasificación.

Si el grado III es el que se encuentra más abajo en el acorde, se dice que es una **primera inversión**, mientras que si es el grado V, se trata de una **segunda inversión** (figura 2.12).

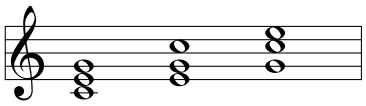


Figura 2.12: Acorde en estado fundamental, primera y segunda inversión.

Acorde cuatríada: Un acorde cuatríada se forma al añadir un nuevo intervalo de tercera a un acorde tríada. El nuevo acorde tiene cuatro notas siendo la última de ellas una séptima de la raíz del acorde. La clasificación de estos acordes se realiza mediante los intervalos que se forman en las notas que lo forman con respecto a la fundamental:

- Mayor con séptima mayor: Tríada mayor e intervalo de séptima mayor.
- Menor con séptima menor: Tríada menor e intervalo de séptima menor.
- Mayor con séptima menor: Tríada mayor e intervalo de séptima menor.
- Menor con séptima menor y quinta disminuida: Tríada disminuida e intervalo de séptima menor.

La figura 2.13 contiene un ejemplo para cada uno de los tipos de acordes cuatríadas.

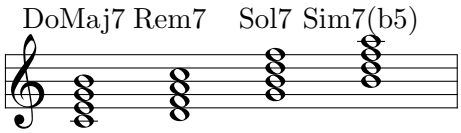


Figura 2.13: Ejemplos de acordes cuatríadas, de izquierda a derecha: mayor con séptima mayor, menor con séptima menor, mayor con séptima menor y menor con séptima menor y quinta disminuida.

2.2 Composición algorítmica

Se conoce como composición algorítmica a la técnica de crear motivos, fragmentos, o incluso obras completas musicales a partir de métodos formales. La composición algorítmica, en su forma más clásica, suele proceder de paradigmas que se separan en dos grandes grupos: estocásticos (basados en la aleatoriedad) y deterministas (el resultado viene determinado de unos valores de entrada). Además de esto, el desarrollo actual de la ciencia computacional ha hecho posible que otras técnicas informáticas sean aplicadas a la composición algorítmica, como los procesos evolutivos o de aprendizaje.

Antes de pasar a explicar cada uno de estos paradigmas, se va a realizar una breve aproximación histórica de la composición algorítmica a lo largo de los años.

2.2.1 Historia

Es evidente que la música y las matemáticas están estrechamente ligadas, tal y como se ha explicado en la sección de Teoría Musical (2.1), es fácil de ver que las reglas que rigen la música son básicamente reglas matemáticas, tanto en términos de melodía, como en ritmo y armonía.

Pitágoras (500 a.C) fue el descubridor de las relaciones aritméticas que existen en una escala musical y de la afinación pitagórica, que se basa en intervalos de quintas justas. Con esto hizo una relación directa entre música y matemáticas, como preámbulo de poder componer música mediante algoritmos y reglas matemáticas.

Ya en el Siglo XI, Guido de Arezzo, un monje italiano al que se le atribuye toda la notación musical actual, ideó un sistema de composición algorítmica a partir de texto escrito, que se recoge en el capítulo XVII su obra *Micrologus*. Este sistema describe unas reglas simples para crear fragmentos musicales a partir de texto relacionando cada una de las vocales con una altura musical (figura 2.14), para poder crear así una melodía para una frase o fragmento escrito (figura 2.15).

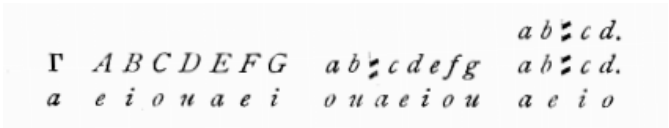


Figura 2.14: Relación de las vocales con notas musicales (nomenclatura americana) en el sistema de Guido de Arezzo.

Otra pincelada de composición que puede enumerarse como algorítmica es el motete *Nuper Rosarum Flores* de Guillaume Dufay, en el año 1436. Se trata de una obra musical que fue compuesta con motivo de la realización de la cúpula de la Catedral de Florencia, por el arquitecto italiano Filippo

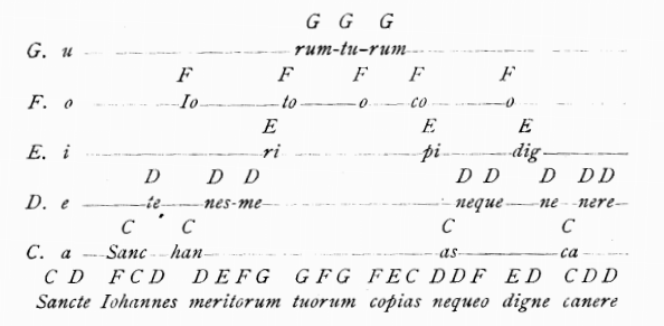


Figura 2.15: Ejemplo de composición musical siguiendo el sistema de Guido de Arezzo.

Brunelleschi, y en la que el compositor se inspira directamente en las proporciones matemáticas de la arquitectura del edificio (6:4 y 2:3) y las aplica a los patrones rítmicos de la pieza musical.

En el año 1650, el sacerdote y estudioso Athanasius Kircher publicó su trabajo musicológico *Musurgia Universalis*, en el que desarrolló un sistema de composición algorítmica, llamado *Arca Musarithmica* (figura 2.16). Se trataba de un mecanismo dentro de un arca, formado por una serie de tiras de madera *columnae* con tablas *tariffa*. En las tablas estaba la información necesaria para el método ideado por Kircher, en forma de números que hacía referencia a las alturas y patrones rítmicos de las notas.

Kircher se basó en este sistema para crear en el año 1661 un nuevo mecanismo llamado *Organum Mathematicum*, que también estaba formado por un conjunto de tiras de madera, divididas en nueve secciones, que ayudaban en la producción de diversos estudios del momento (aritmética, geometría, calendarios, movimientos planetarios) los cuales se incluía la música. Las tablas musicales del órgano eran básicamente un subconjunto de las tablas del *Arca Musarithmica*.

El siguiente hecho remarcable en la historia de la composición algorítmica se halla en el siglo XVIII, cuando Wolfgang Amadeus Mozart diseñó el sistema *Musikalisches Würfelspiel*, un juego de dados mediante el cual se podía componer un vals de forma aleatoria. Mediante el resultado al azar de los dados se seleccionan fragmentos musicales, que dispuestos de forma secuencial, componen una obra completa (figuras 2.17 y 2.18).

Ya después de la Segunda Guerra Mundial, gran cantidad de compositores siguieron la corriente del Serialismo [1], apareciendo obras como la *Suite Lírica* para cuarteto de cuerda de Alban Berg, que utilizaba permutaciones como transposición o inversión sobre las notas para obtener los doce tonos

[1] Técnica de composición musical, evolucionada a partir del *dodecafonismo* de Arnold Schönberg. El serialismo se establece a partir del principio serial, por el que cada una de las dimensiones musicales, como la altura, ritmo, dinámica o timbre se definen por doce notaciones, que deben aparecer por igual a lo largo de la obra musical, sin repetir alguna hasta que hayan aparecido las doce, evitando así cualquier apariencia tonal en la obra.

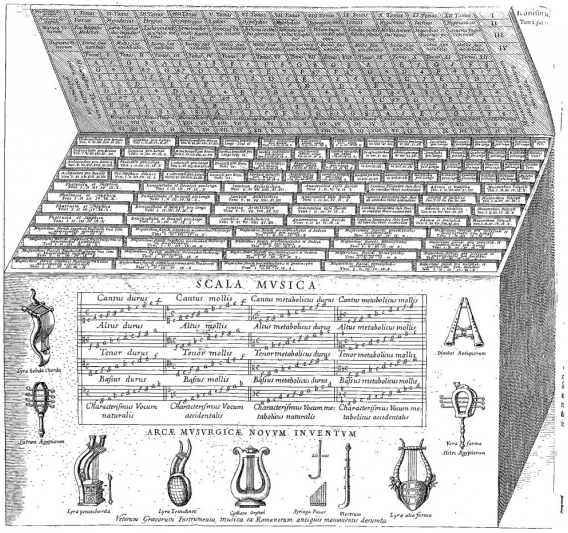


Figura 2.16: *Arca Musarithmica* de Athanasius Kircher, año 1650.

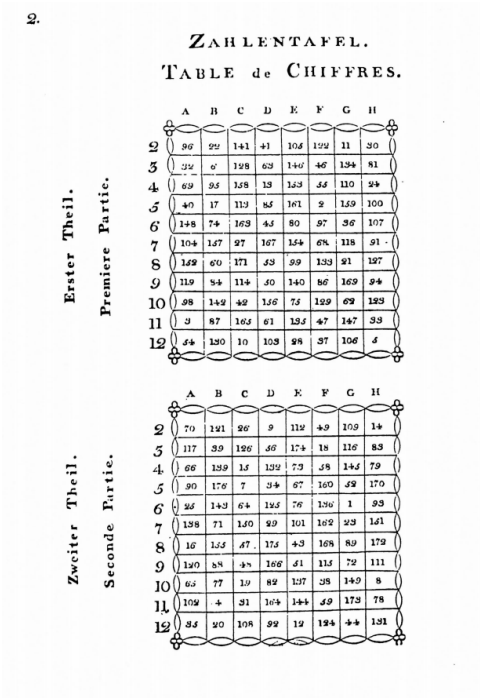


Figura 2.17: Tabla de selección del fragmento musical según el resultado de los dados en el juego de W.A. Mozart.

requeridos por el principio serial. Este proceso realizado por Berg puede entenderse como algorítmico determinista.

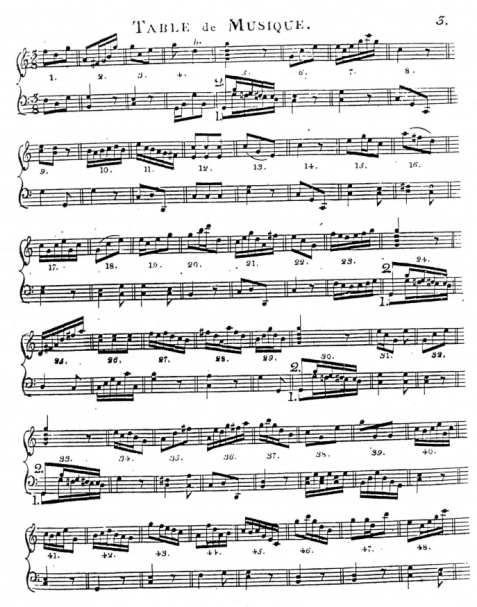


Figura 2.18: Algunos de los compases del juego de dados de W.A. Mozart.

Paralela a esta corriente musical, compositores como Iannis Xenakis utilizaron fuentes alternativas para componer música. Xenakis estaba altamente interesado en aplicar las matemáticas a la composición musical, centrándose en la teoría de probabilidad, y desarrollando trabajos de *música estocástica*, término acuñado por él mismo, como su conocida obra *Pithoprakta*, en el año 1956, compuesta a partir de métodos probabilísticos.

Fueron Lejaren Hiller y Leonard Isaacson los primeros en utilizar un ordenador para generar música de forma algorítmica. Para ello utilizaron, en el año 1957, el ordenador ILLIAC de la Universidad de Illinois, y a través de algoritmos se generó la *Suite ILLIAC* para cuarteto de cuerda, pionera en este campo.

Y finalmente, otra obra remarcable en la historia de la composición algorítmica, ya utilizando ordenadores, es *HPSCHD*, por el compositor John Cage junto con Lejaren Hiller, creada entre los años 1967 y 1969, y que trataría de *"hacernos entender el proceso de pensamiento"*.

Estos son algunos de los hechos más remarcables en la historia de la composición algorítmica, además de otros muchos estudios y sistemas ideados que beben de infinitas fuentes de inspiración para componer música. A continuación se van a conocer los tipos de sistemas de composición algorítmica que existen, en función de su enfoque inicial.

2.2.2 Procesos estocásticos

Se conoce como proceso estocástico como un proceso cuya evolución temporal es aleatoria, donde el conjunto de todos los posibles valores que puede tomar el proceso es conocido.

Históricamente en la composición algorítmica se han utilizado dos subtipos de procesos estocásticos, dependiendo de qué forma se obtienen los valores del proceso, que puede clasificarse como de sucesos independientes o de sucesos dependientes.

Puesto que el proceso estocástico va generando valores aleatorios, el papel del compositor juega un papel poco importante, limitado a seleccionar el conjunto de posibles valores, definir sus probabilidades, o decidir a qué variables de la composición musical irá destinado el proceso (alturas, duraciones, dinámica, etc).

El diseño del sistema de composición algorítmica a partir de procesos estocásticos puede ser más o menos complicado, pudiendo afectar el proceso de forma simple y directa a las variables de la composición, o formar parte de otro sistema mayor, donde la toma de decisiones se realiza de forma aleatoria.

Sucesos independientes

Un proceso estocástico de sucesos independientes se caracteriza porque los valores que se obtienen de dicho proceso no dependen de ningun valor previo.

Si el conjunto de valores es discreto, la probabilidades de cada uno de ellos serán discretas, mientras si el conjunto de valores es continuo, las probabilidades para este se definirán a partir de una función de densidad de probabilidad.

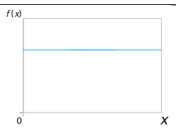
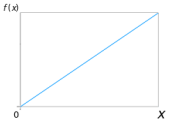
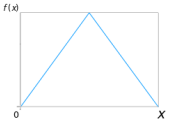
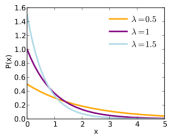
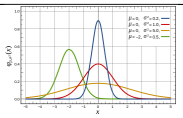
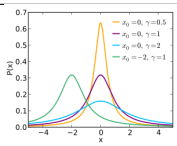
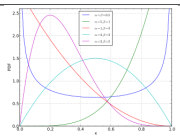
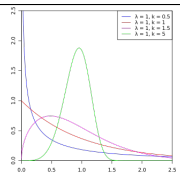
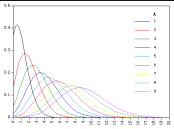
La probabilidad de un determinado valor siempre será un número entre 0 y 1. En este caso, la probabilidad de que una variable X del conjunto se expresa como:

$$P\{X\} = y, \quad 0 \leq y \leq 1$$

Una propiedad importante es que la suma de todas las probabilidades del conjunto sea igual a 1:

$$\sum_{i=0}^{n} P\{X_i\} = 1$$

En la composición algorítmica es más común utilizar una función de densidad para definir la probabilidad de un conjunto arbitrario, normalmente de valores entre 0 y 1, que serán después proporcionados al conjunto que se quiera, como por ejemplo, alturas musicales. Existen diferentes funciones de densidad (ver tabla 2.6) interesantes en la composición algorítmica, que pueden utilizarse para experimentar y observar los resultados que se obtienen.

Nombre	Función	Forma
Uniforme	$f(x) = 1$	
Lineal	$f(x) = x$	
Triangular	$f(x) = \begin{cases} x & x \leq 0.5 \\ 1 - x & x > 0.5 \end{cases}$	
Exponencial	$f(x) = \lambda^{-\lambda x}, \quad \lambda > 0$	
Gaussiana	$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	
Cauchy	$f(x) = \frac{\alpha}{\pi(\alpha^2 + x^2)}$	
Beta	$f(x) = \frac{1}{B(a,b)} x^{a-1}(1-x)^{b-1}$	
Weibull	$f(x) = \frac{tx^{t-1}}{s^t} e^{-(\frac{x}{s})^2}$	
Poisson	$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}$	

Cuadro 2.6: Algunas funciones de densidad.

Sucesos dependientes

Por el contrario, un proceso estocástico de sucesos dependientes está principalmente caracterizado porque los valores que se generan sí dependen de los valores anteriormente generados por el mismo sistema.

La base matemática de este tipo de proceso es la de la probabilidad condicionada, que se define como la probabilidad de un suceso X_{n+1} posterior en el tiempo a un suceso X_n:

$$P\{X_{n+1} = j | X_n = i\} = P_{ij} \tag{2.1}$$

Sobre esta base se han ideado diferentes formas de composición algorítmica. A continuación se explican algunas de ellas:

Modelos de Markov: Los modelos de Markov (también denominados cadenas e Markov) fueron ideados por el matemático Andrey Andreyevich Markov, para explicar procesos aleatorios dependientes del tiempo.

Según este modelo, se describe un proceso estocástico como una secuencia de eventos aleatorios dependientes de un parámetro temporal. Al conjunto de eventos se le denomina *espacio de estados*, y al conjunto de parámetros espacio de parámetros. Cuando este espacio de estados es contable, el proceso estocástico también puede denominarse *cadena estocástica*.

Una cadena estocástica donde la probabilidad de un futuro estado X_{t+1} (la variable aleatoria en el tiempo t+1) depende del estado actual X_t se le conoce como *cadena de Markov*, y su probabilidad de describe como en la ecuación 2.1.

Una cadena de Markov puede representarse y quedar completamente explicada mediante un grafo (*grafo de transición*), o mediante una matriz (*matrix de transición*). En el grafo de transición los nodos representan el espacio de estados, y las aristas que unen los nodos representan las probabilidades de transición entre ellos (figura 2.19).

Por otra parte, una matriz de transición contiene las probabilidades de transición entre todos los posibles estados, denominando cada fila como un estado actual y cada columna como un estado futuro, y donde la posición de la matriz m_{ij} contiene la probabilidad de suceder el estado j tras el estado i (tabla 2.7).

Si el modelo de Markov únicamente se tiene en cuenta el evento inmediatamente anterior en el cálculo de las probabilidades, se dice que es de *primer orden*, aumentando este orden en función de cuantos sucesos anteriores se utilizan para calcular la probabilidad de un suceso futuro. Aumentar dicho orden hace que aumente exponencialmente las dimensiones de la matriz, lo que incrementa considerablemente la complejidad del sistema, hablando en términos computacionales. El número de celdas en la matriz de transición depende directamente del número de estados (S) y del orden de la cadena (N), siendo este:

$$x = S^{N+1}$$

	S1	**S2**	**S3**
S1	0.0	0.6	0.4
S2	0.2	0.3	0.5
S3	0.2	0.7	0.1

Cuadro 2.7: Matriz de transición representando una cadena de Markov.

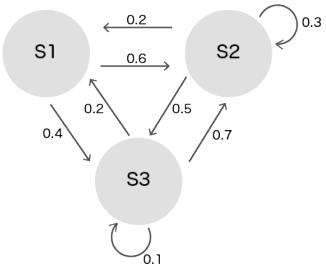


Figura 2.19: Grafo de transición representando una cadena de Markov.

Las cadenas de Markov pueden aplicarse a la composición algorítmica a partir de un análisis previos de obras musicales, para obtener las probabilidades de transición entre las alturas, duraciones, patrones rítmicos, motivos, o cualquier tipo de estado que pueda extraerse de la misma, para obtener así la cadena de Markov deseada, y poder componer otras piezas de forma aleatoria que sigan el patrón de probabilidades de la obra observada.

Random Walk: Otra forma de proceso estocástico dependiente es el denominado Random Walk (paseo aleatorio), que se define como un ejemplo de una cadena de Markov, donde el conjunto de estados se encuentra ordenado, y a partir de un estado solo se puede pasar a sus vecinos, ya sea dando saltos de una o más unidades. Es importante saber que en este proceso un estado no puede repetirse de forma sucesiva.

Matemáticamente, se define el paseo aleatorio como:

$$X(t+\tau) = X(t) + \phi(\tau)$$

Donde ϕ es la variable aleatoria que decide el siguiente paso, y τ es el intervalo entre dos pasos sucesivos.

Queda por definir el comportamiento de X(t) cuando se aproxime a los límites del conjunto de estados. Este comportamiento puede ser:

- Reflectivo: Se cambia la dirección al llegar al límite.

- Elástico: La probabilidad de llegar al límite disminuye al acercarse al mismo, y aumenta la probabilidad de aproximarse al punto medio del conjunto.
- Absorbente: Si se pide rebasar los límites se permanece en ellos.

Por último, también puede definirse la longitud del paso, de una o más unidades.

Comúnmente en composición algorítmica, el conjunto de estados de un paseo aleatorio suelen ser las alturas musicales, limitando donde se desee, para obtener melodías más amplias o más limitadas. Aunque siempre puede experimentarse y utilizar el paseo aleatorio para decidir las duraciones o velocidades de las notas de la pieza que se está componiendo.

Ruido fraccional: A partir del ruido pueden componerse fragmentos musicales, y según su distribución espectral estas composiciones tendrán unas características que pueden resultar interesante para el compositor. Concretamente, el ruido fraccional se refiere a aquel que tiene su distribución espectral (S) en función de la frecuencia de la forma:

$$S(f) = 1/f^{\lambda}, \quad 0 \leq \lambda \leq 2$$

El ruido con esta característica se presenta en la naturaleza en infinidad de formas, y ha sido observado y estudiado a lo largo de los años. El comportamiento que presenta este tipo de ruido implica correlación entre las frecuencias y la cantidad de energía, que aumenta a medida que aumenta el parámetro λ. En ciertos valores de λ, el ruido adquiere denominación (ver tabla 2.8).

$S(f) = 1/f^0 = 1$	Ruido blanco
$S(f) = 1/f^1$	Ruido rosa
$S(f) = 1/f^2$	Ruido marrón

Cuadro 2.8: Denominaciones del ruido por su distribución espectral.

En 1975, Richard Voss y John Clarke estudiaron la distribución espectral de grabaciones musicales y de voz (Primer Concierto de Bradenburgo de J.S. Bach, Rags de Scott Joplin, emisoras de radio clásicas, de rock, jazz y blues), observando que estas distribuciones se aproximaban en un grado alto a la distribución espectral del ruido rosa ($\lambda = 1$) (ver figura 2.20). Por esto, concluyeron que el ruido rosa tiene características deseables en la composición algorítmica estocástica. Los valores que genera el ruido 1/f están relacionados de forma logarítmica con el pasado, es decir, tiene más importancia e influencia los valores más cercanos al valor actual, teniendo más influencia los 10 valores últimos que los 100 valores últimos, que a su vez tienen más

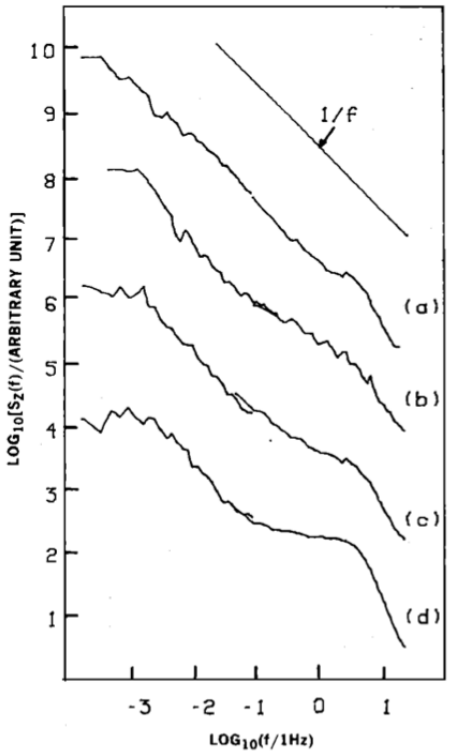


Figura 2.20: Densidad espectral de las variaciones de frecuencias, medida en escala logarítmica para una emisora de música clásica (a), jazz y blues (b), rock (c) y voz (d), con referencia de la distribución de ruido rosa.

importancia que los 1000 valores últimos.

En comparación con el resto de distribuciones, los valores generados por el ruido blanco no han sido influenciados por los sucesos pasados, por lo que las composiciones musicales que se generen por ruido blanco tendrán apariencia de ser totalmente aleatorias, sin ninguna característica o patrón musical repetitivo. En el otro extremo, los valores que se generan por ruido marrón están fuertemente, y casi exclusivamente infuenciados por los eventos pasados más cercanos al actual, por lo que los fragmentos musicales tendrán patrones totalmente repetitivos.

Por lo tanto, es lógico concluir en que el ruido rosa se encuentra en un punto intermedio entre aleatoridad y repetitividad, característica deseable en la música tradicional, donde los patrones musicales son repetitivos, aunque estos presenten ciertas variaciones entre sí. Otros valores de λ, como 0.5 y 1.5 también pueden utilizarse en la composición musical, para buscar un equilibrio diferente entre aleatoriedad y repetitividad.

Una vez se ha obtenido la forma matemática de generar valores numéricos estos se utilizarán en forma de alturas, duraciones, dinámica, o cualquier parámetro musical deseado.

2.2.3 Procesos deterministas

Un proceso determinista es aquel en el que el azar no juega ningún papel en la obtención de estados del sistema. Esto quiere decir que se producirá siempre la misma salida ante unos datos de entrada, sin importar cuántas veces se ejecute el sistema.

En un proceso determinista los estados futuros están determinados por los estados previos que se han sucedido.

Respuestas caóticas

Las respuestas caóticas provienen de procesos iterativos, donde la salida de una iteración del sistema se utiliza como realimentación en la entrada. Un proceso iterativo se describe como una operación matemática que se aplica a un valor inicial x_0. Al conjunto de puntos que se obtienen en un proceso iterativo se le conoce como *órbita* O, y para una función F con un valor inicial x_0, la órbita se define como:

$$O^F(x_0)$$

Un proceso iterativo puede producir generalmente tres tipos de órbitas:

1. Órbitas cuyos puntos tienden a un valor estable.
2. Órbitas cuyos puntos oscilan entre dos valores límite.
3. Órbitas cuyos puntos tienden al *caos*.

Las órbitas o respuestas caóticas se caracterizan por no poder distinguirse en ellas un patrón recurrente o repetitivo, y aunque no existe una regla matemática que determine si un sistema se está comportando de forma caótica si que existen algunos principios para reconocer este comportamiento:

- El valor de x_0 produce variaciones sensiblemente diferentes en la salida. Esto se conoce como *efecto mariposa*[2].
- Presencia de bifurcaciones en determinados momentos del sistema. Se dobla el número de puntos diferentes del sistema.
- Asentamientos esporádicos; se producen pequeñas trayectorias casi estables, aunque se bifurcan inmediatamente.

[2]Término acuñado por Edward Lorenz, a partir de un modelo climático diseñado por él mismo en el que observaba que pequeños cambios iniciales producían cambios climáticos extremos.

La aplicación o **ecuación logística** es un ejemplo de proceso iterativo que produce una respuesta caótica. Esta ecuación tiene la forma:

$$x_{n+1} = rx_n(1 - x_n), \quad 0 \leq \Delta \leq, \quad 0 \leq x_0 \leq 14$$

La órbita de este sistema iterativo depende del parámetro r, tal y como se indica en la tabla 2.9 para algunos valores de r, y visualizarse en el diagrama de bifurcaciones (figura 2.21).

Valor de r	**Respuesta**
$0 < r \leq 1$	La órbita tiende a 0, independientemente del valor de x_0.
$1 < r \leq 2$	La órbita tiende a $\frac{r-1}{r}$, independientemente del valor de x_0.
$2 < r \leq 3$	La órbita oscila en el entorno de $\frac{r-1}{r}$, pero acaba estabilizándose en este. Se produce una bifurcación.
$3 < r \leq 3.45$	La órbita oscila en el entorno de x_0, produciéndose dos bifurcaciones más.
$3.45 < r \leq 3.54$	La órbita oscila indefinidamente entre cuatro valores diferentes.
$3.54 < r < 3.57$	La órbita comienza a oscilar entre cada vez más valores.
$r \geq 3.57$	Comienzo de respuesta caótica.

Cuadro 2.9: Posibles órbitas de la ecuación logística en función del parámetro r.

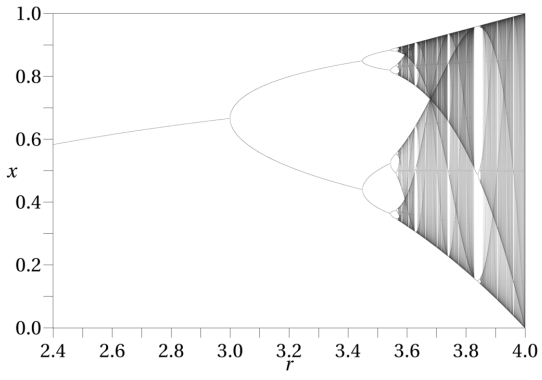


Figura 2.21: Diagrama de bifurcaciones de la aplicación logística.

La amplia variabilidad de respuestas de la ecuación logística y de otros sistemas similares, la hace interesante para componer fragmentos musicales a partir de la salida que produce, especialmente cuando ésta es caótica

(sin menospreciar las órbitas que oscilan entre varios valores, que pueden resultar interesantes para según que propósitos o estilos musicales). Para resultar agradables, las composiciones musicales tienen que presentar de una forma equilibrada cierta repetitividad y variabilidad, por lo que las respuestas caóticas son especialmente susceptibles de ser utilizadas como forma de obtener valores que usar como parámetros musicales, ya que oscilan entre un rango de valores, sin que esta oscilación sea repetitiva a lo largo del tiempo.

Desarrollo Motívico

Un método de composición algorítmica que cumple las características de sistema determinista es el desarrollo motívico. Un motivo es el fragmento musical (melódico, rítmico o ambos) más pequeño que puede extraerse de una composición, y que puede ser modificado de forma independiente (figura 2.22). El motivo tiene gran importancia en la obra musical, ya que define la temática de la misma, y es lo que suele persistir en la memoria para identificar una obra musical.

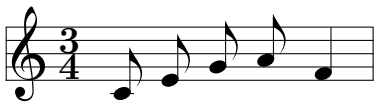


Figura 2.22: Un motivo musical.

Uno de los ejemplos más característicos de composicón musical por desarrollo motívico es el de la Quinta Sinfonía de Ludwig Van Beethoven, donde el motivo principal (figura 2.23) se repite a lo largo de toda la obra con ciertas variaciones tanto melódicas como rítmicas, estableciendo el tema principal de la sinfonía, y siendo recordado por todos como lo que la define.

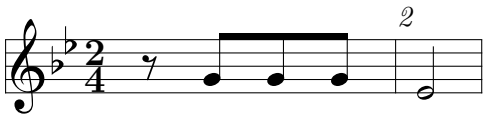


Figura 2.23: Motivo principal en la Quinta Sinfonía de Beethoven.

Un motivo se define como dos conjuntos del mismo tamaño, uno perteneciente a la parte melódica del mismo, y otro perteneciente a la parte rítmica. Para el motivo anterior (figura 2.22) se definen los siguientes conjuntos:

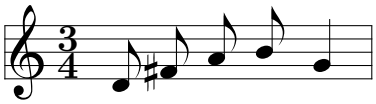
$$M_{melódico} = \{Do_4, Mi_4, Sol_4, La_4, Fa_4\}$$

$$M_{rítmico} = \{Corchea, Corchea, Corchea, Corchea, Negra\}$$

El motivo puede alterarse de diferentes formas para así obtener variaciones del mismo, pero manteniendo siempre la temática principal del mismo. En la música clásica occidental se han adoptado las siguientes operaciones que pueden realizarse sobre la melodía o el ritmo de un motivo, se explican a continuación y se ejemplifican con el motivo 2.22:

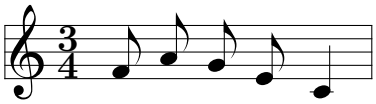
- Transposición: Consiste en aumentar o disminuir en semitonos la altura de las notas que componen el motivo. Esta operación sólo es aplicable a la parte melódica del motivo.

$$Transposición(M_{melódico}, +2) = \{Re_4, Fa_4\sharp, La_4, Si_4, Sol_4\}$$

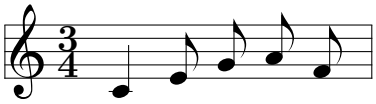


- Retrogradación: Se invierte el orden de las notas, de forma que estas quedan como vistas en espejo. Puede aplicarse tanto a la melodía como al ritmo.

$$Retrogradación(M_{melódico}) = \{Fa_4, La_4, Sol_4, Mi_4, Do_4\}$$

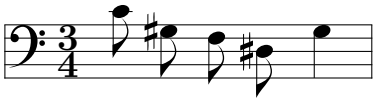


$$Retrogradación(M_{rítmico}) = \{Negra, Corchea, Corchea, Corchea, Corchea\}$$



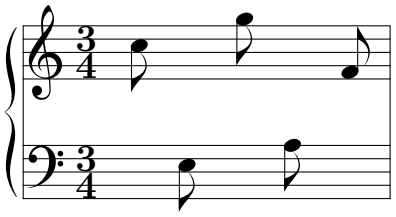
- Inversión: Para cada par de notas del motivo, se invierte la dirección del intervalo que existe entre ellas. Solo es aplicable a la melodía del motivo.

$$Inversión(M_{melódico}) = \{Do_4, Sol_3\sharp, Fa_3, Re_3\sharp, Sol_3\}$$



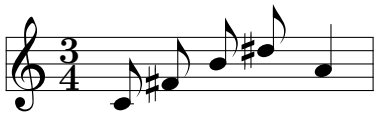
- Desplazamiento de registro: Se desplaza en octavas hacia arriba o hacia abajo cada una de las notas que conforman el motivo. Solo se aplica a la parte melódica.

$$Desplazamiento = \{+1, -1, +1, -1, 0\}$$
$$DespRegistro(M_{melódico}, Desplazamiento) = \{Do_5, Mi_3, Sol_5, La_3, Fa_4\}$$



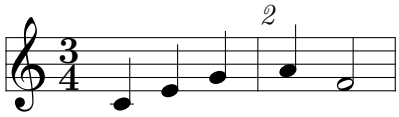
- Expansión-Contracción: Se aumenta o disminuye la distancia en semitonos entre cada par de notas. Solo aplicable a la melodía.

$$Expansión(M_{melódico}, +2) = \{Do_4, Fa_4\sharp, Si_4, Re_5\sharp, La_4\}$$



- Aumentación-Disminución: Exclusivo de la parte rítmica del motivo, se aumenta o disminuye la duración de las figuras, multiplicando o dividiendo por números factores de 2.

$$Aumentación(M_{rítmico}, +2) = \{Negra, Negra, Negra, Negra, Blanca\}$$



Combinando cada una de las operaciones anteriores puede conseguirse fragmentos musicales que posean cierta coherencia musical, ya que mantienen en todo momento el grado de repetividad necesario, a partir de repetir el motivo principal, y el grado de variación del motivo, por la presencia de las variaciones del motivo utilizando las operaciones que han sido expuestas.

Serialismo

El serialismo es una técnica de composición musical con sus orígenes datados en el siglo XIX. Proviene del *Dodecafonismo* de Arnold Schöenberg, en la que se trata a los doce tonos de la escala cromática con la misma importancia a la hora de componer, conocido como *principio serial*. La norma principal del dodecafonismo es que no se puede repetir un tono si antes no han aparecido el resto de tonos que conforman la escala cromática. Con esta

norma se evita que un tono apareza más que el resto, evitando asi que la composición carezca de característica tonal o modal.

Asimismo, esta técnica crea la composición musical aplicando operaciones como las vistas en la técnica de desarrollo motívico (2.2.3) con ciertas variaciones, ya que las notas solo pueden pertenecer al conjunto de la escala cromática en una determinada octava, por lo que se utilizan operaciones en módulo 12 para conseguir este propósito.

Las operaciones que definió Schöenberg en el Dodecafonismo son las siguientes:

- Retrógrado (R): Se invierte el orden de la serie original, quedando en espejo con respecto a ella.
- Inversión (I): Se invierte la dirección de los intervalos de la serie original.
- Retrógrado de la inversión (RI): Se invierte el orden de la serie con la inversión aplicada.
- Transposición (T): Se aumenta en semitonos cada una de las notas de la serie original. La cantidad de semitonos debe ser un número entre 0 y 11.

El proceso de composición siguiendo la técnica dodecafónica es el siguiente:

1. Se establece la serie de tonos con un orden determinado a decisión del compositor.
2. Se aplica cada una de las operaciones definidas anteriormente, para obtener variaciones de la serie original.
3. Se utilizan estas series en la composición, eligiendo el resto de parámetros (ritmo, duraciones, dinámica) a la elección del compositor.

La figura 2.24 contiene una posible serie original y algunas de sus correspondientes variaciones, que se utilizaría como punto de partida en la composición musical. En la figura 2.25 puede verse un fragmento de *Fünf Klavierstücke*, una de las primeras composiciones de Schöenberg siguiendo la técnica dodecafónica. Puede apreciarse la ausencia de tonalidad en el fragmento, por estar las alturas situadas siguiendo una serie.

El Dodecafonismo ha ido evolucionando en la técnica conocida como **Serialismo Integral**, que nace como una crítica hacia el Dodecafonismo por no aplicar el principio serial a nada más que a las alturas. Por esto, el Serialismo Integral no es más que aplicar el mismo principio serial al resto de los parámetros de una composición musical. De esta forma, cada una de las notas de una composición tendrán una altura, duración, dinámica y timbre

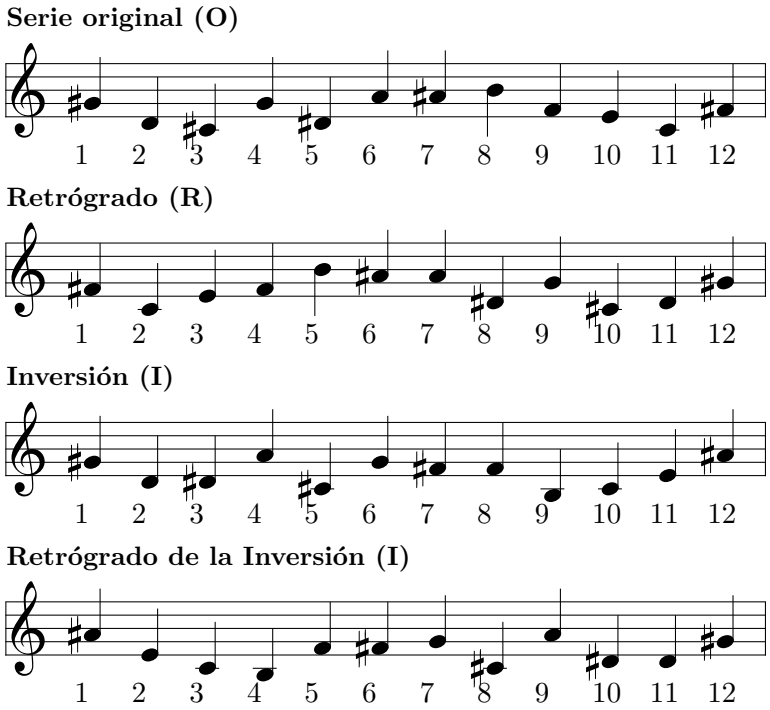


Figura 2.24: Ejemplo de serie dodecafónica original y algunas variaciones.

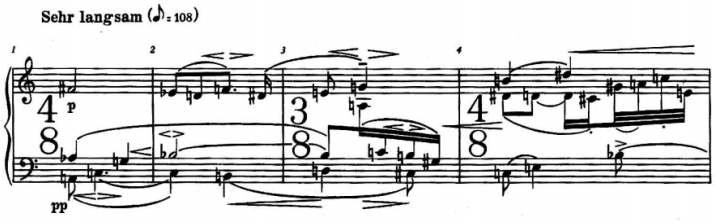


Figura 2.25: Fragmento de *Fünf Klavierstücke* de Arnold Schöengberg.

diferentes, y que están determinados por la serie definida para cada uno de los parámetros. Para obtener las series se sigue el mismo proceso explicado para la técnica dodecafónica, esta vez para cada uno de los parámetros musicales.

Puede decirse que la música compuesta bajo la técnica serial es determinista, por construirse siguiendo un proceso que tendrá siempre el mismo resultado ante unos determinados valores de entrada, y en la que no hay ningún factor aleatorio a tener en cuenta. Al poder ser definida formalmente, es fácilmente computable y utilizable como un método de composición algorítmica, ya sea en la obtención de las series, o en la composición de fragmentos musicales.

En la figura 2.26 se aprecia un ejemplo de series para varios parámetros

musicales (altura, ritmo, dinámica y articulación), creadas por Olivier Messiaen para su obra *Quatre études de rythme*, y utilizadas más adelante por Pierre Boulez en su obra *Structures I*.

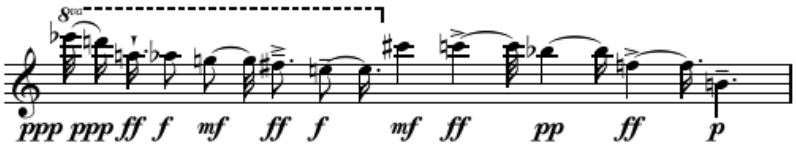


Figura 2.26: Series sin ordenar de Olivier Messiaen para la altura, ritmo, dinámica y articulación.

2.2.4 Otros métodos

Existen más técnicas, la mayoría exclusivamente computacionales, que a partir de su existencia se ha estudiado su uso como formas de composición algoritmica. Al tratarse de técnicas con mayor complejidad que las vistas anteriormente, no se han incluido como métodos a estudiar en el presente trabajo. Aún así, se van a enumerar y explicar brevemente algunas de ellas para al menos saber de su existencia.

- **Gramática generativa:** Un método utilizando tanto en análisis musical como en composición algorítmica. Tiene sus bases en el modelo lingüístico de *Noam Chomsky*. La gramática generativa fue creada como un marco sobre el que estudiar la sintaxis lingüistica. Por medio de un conjunto de reglas se puede predecir las combinaciones que son correctas dentro de un determinado lenguaje. Este mismo marco teórico puede ser utilizado para la composición algorítmica por medio de crear un conjunto de reglas que determinen que combinaciones musicales son *correctas*.

- **Algoritmos genéticos:** Un algoritmo genético es un conjunto de pasos modelados como un sistema natural, inspirándose en la teoría de la evolución de *Charles Darwin*. Los pasos de el algoritmo se generan de una forma estocástica siguiendo los pasos de un proceso evolutivo natural, como una forma de simulación computacional de la teoría de la evolución para un problema concreto que ha de resolverse. Este tipo de algoritmos son muy apropiados para resolver problemas que son dificilmente modelables dentro de un contexto matemático. Básicamente el proceso consiste en tomar una población inicial, mutarla, cruzarla, y hacerla pasar por una función que determinara si es mejor o peor que su antecesor. Es un proceso bastante aproximado al proceso humano de composición musical, donde a partir de un fragmento se generan variaciones y se prueba cómo de agradable y armonioso es para el oído.

- **Redes neuronales:** Otra forma de resolver problemas inspirado en un modelo biológico es el de las redes neuronales artificiales, donde la solución se encuentra a partir de cambiar la estructura de componentes básicos que se encuentran interconectados. Estos componentes son conocidos como *neuronas*. Dentro de la composición algorítmica, no todas las arquitecturas de redes neuronales son válidas para su uso, y estas tienen que ser adaptadas para que representen correctamente las propiedades estructurales de la música. También frecuentemente, las redes neuronales son aplicadas a un compositor algorítmico genético, como la función que determina la apropiabilidad de la variación generada.
- **Aprendizaje computacional:** El aprendizaje computacional une una gran cantidad de distintas técnicas que intentan automatizar un comportamiento inteligente. Cada una de estas técnicas se estudian por su aplicabilidad a la solución de un problema específico. Dentro de la composición algorítmica existen diferentes enfoques de aprendizaje computacional, por lo que es complicado ceñirse únicamente a uno. Existen trabajos de composición algorítmica realizados dentro del marco del aprendizaje computacional que utilizan razonamiento lógico, lógica difusa o aprendizaje automático, para por ejemplo, componer fragmentos musicales que imitan un determinado estilo o reproducir el proceso creativo que sigue una persona al componer música.

2.2.5 Sistemas actuales

Actualmente existen infinidad de sistemas de composición algorítmica implementados, algunos centrados en algún método concreto, y otros de propósito más general. En esta parte se van a enumerar algunos de ellos:

- **AC Toolbox** [3]**:** Colección de herramientas para la composición algorítmica. El sistema implementa multitud de generadores a partir de procesos estocásticos, sistemas caóticos, tablas de transición, etc.
- **CAPieces** [4]**:** Sistema de composición algorítmica por medio de autómatas celulares.
- **Maestro Genesis** [5]**:** Sistema de composición de pistas armónicas y de acompañamiento para composiciones ya existentes. Con esta aplicación, los usuarios pueden crear acompañamiento para piezas que hayan compuesto, sin necesidad de tener amplios conocimientos musicales.
- **OpenMusic** [6]**:** Entorno visual de programación para composición al-

[3] `http://kc.koncon.nl/downloads/ACToolbox/`
[4] `http://ca-pieces.sourceforge.net/index.html`
[5] `http://maestrogenesis.org/`
[6] `http://repmus.ircam.fr/openmusic/home`

gorítmica basado en el lenguaje **LISP**. El software ha sido diseñado y creado por el *IRCAM* [7].

- **Strasheela** [8]**:** Sistema de composición algorítmica que genera la música a partir de unas reglas teóricas introducidas por el usuario.

2.3 El estándar MIDI

El estándar MIDI se refiere a varios aspectos, como los cables que unen los dispositivos, los conectores de los cables, la forma de transmitir los mensajes o el tipo de mensajes y su codificación. También más adelante se añadió el estándar para los ficheros MIDI, llamado SMF (*Standard MIDI file*). Son los ficheros SMF la única parte relevante al trabajo, por lo que se explicará brevemente su contenido.

El fichero SMF se adopta como un formato de ficheros para guardar en disco los mensajes MIDI para su posterior lectura e interpretación por programas de edición musical, secuenciadores o lectores de partituras. Un fichero SMF contiene secuencias MIDI en conjunto a información de tiempos y otro tipos de eventos (meta-eventos). Los datos son almacenados en formato binario y el fichero se divide en dos bloques: cabecera y pista.

El bloque de cabecera es único en el fichero y contiene información que afecta al conjunto de pistas, como el nombre de la secuencia, autor, tonalidad, tempo, etc.

Por otra parte, los bloques de pistas contienen los eventos MIDI de sistem, de canal (mensajes de inicio y final de nota), precedidos de un valor denominado tiempo-delta, que especifica el número de pulsos de reloj (en función de la resolución de la secuencia) que deben pasar para emitir el evento MIDI desde el último evento emitido. También contiene otro tipo de mensajes denominados meta-eventos, que contienen información complementaria que acompaña a la secuencia MIDI.

En la figura 2.27 puede visualizarse la estructura tipo de un fichero SMF.

2.4 Software utilizado

El sistema se ha construido utilizando el lenguaje C++ y utilizando un conjunto de librerías llamado openFrameworks, además de librerías externas para la lectura y escritura de ficheros SMF.

[7] Institut de Recherche et Coordination Acoustique/Musique
[8] `http://strasheela.sourceforge.net/strasheela/doc/index.html`

Bloque	Información	Longitud	Contenido
Cabecera	Firma	4	"MThd"
	Longitud	4	del resto del bloque = 6
	Formato	2	0 = 1 sola pista 1 = multi-pista (1 por instrumento) 2 = multi-secuencia
	Número de pistas	2	Si Tipo = 0 ⇒ siempre 1
	Resolución temporal	2	En pulsos por cada nota"negra"
<Número de pistas> veces:			
Pista (*track*)	Firma	4	"MTrk"
	Longitud	4	N
	Datos	N	Secuencia de mensajes MIDI: de canal + de sistema + meta-eventos + *delta-times* (separación temporal entre mensajes)

Figura 2.27: Estructura de un fichero SMF.

2.4.1 openFrameworks

openFrameworks [9] es un conjunto de herramientas multiplataforma de código abierto escrito en C++. Se trata de un marco de programación que facilita algunas de las tareas que sin el uso de librerías sería tedioso. Gracias al uso de openFrameworks, el programador puede centrarse en la implementación de los algoritmos internos de la aplicación a desarrollar, y no prestar atención en otras tareas, como la gestión de audio, gráficos, soporte de plataformas, comunicaciones, o interfaz gráfica.

Algunas de las funcionalidades ya implementadas y que pueden ser utilizadas fácilmente de openFrameworks son:

- Gráficos 2D y 3D, mediante fachadas sobre OpenGL.
- Entrada, análisis y salida de audio.
- Gestión de fuentes tipográficas.
- Gestión (entrada, análisis y salida) de imagen y vídeo.
- Interfaz gráfica.

Gracias a ser de código abierto bajo licencia MIT, existe una gran comunidad de usuarios que se dedican a desarrollar o trasladar otras librerías al marco de openFrameworks. La comunidad de openFrameworks denomina a estas librerías creada por usuarios *addons*.

2.4.2 Otras liberías

El resto de librerías que serán utilizadas y que no están englobadas dentro del conjunto de openFrameworks son las siguientes:

[9] `http://openframeworks.cc`

- **Midifile**[10]: Librería de C++ para la lectura y escritura de ficheros MIDI estándar.
- **COLORED_NOISE**[11] : Librería de C++ para la generación de secuencias que simulan ruido fraccional.

[10] http://midifile.sapp.org/

[11] http://people.sc.fsu.edu/~jburkardt/cpp_src/colored_noise/colored_noise.html

3
Sistema General

3.1 Análisis de los requisitos generales

El sistema que va a implementarse debe consistir en una aplicación capaz de componer fragmentos musicales a través de los diferentes métodos de composición algorítmica explicados (sección 2.2) de una forma sencilla y cercana al usuario, con un propósito explorativo, didáctico o creativo. Se proponen los siguientes requisitos para el sistema:

- Facilidad de uso: El sistema debe ser fácil de utilizar y no presentar barreras para el usuario, de forma que pueda centrarse en la tarea, explorar, aprender y experimentar las diferentes formas de composición algorítmica estudiadas.
- Implementación de diferentes métodos de composición algorítmica: La aplicación podrá componer los fragmentos musicales de varias formas, pudiendo así apreciarse las ventajas y desventajas de cada uno de ellos, y ampliando de esta forma las posibilidades del sistema. El usuario tendrá control sobre las opciones específicas de cada uno de los métodos de composición.
- Parametrización del compositor: El usuario podrá decidir a través de la interfaz parámetros musicales relacionados con la altura, ritmo y duración.
- Visualización de la salida musical: El usuario ha de tener la posibilidad de escuchar o visualizar de alguna forma la composición musical, de forma que podrá tener información de lo que ha sucedido. Esta visualización será lo más clara y explícita posible.

- Posibilidad de exportación de los fragmentos musicales fuera del sistema: El usuario tendrá la opción de exportar la composición fuera del entorno del sistema, para poder ser utilizada en otros entornos y con diferentes propósitos.

3.2 Solución propuesta

El sistema que se propone contiene soluciones a todas las necesidades especificadas anteriormente:

- Características separadas en módulos: El sistema se encontrará separado por módulos, a partir de las funcionalidades de cada uno, para asi facilitar la implementación, y mantener separadas cada una de las partes:
  - Compositor.
  - Reproductor de fragmentos musicales.
  - Visualizador de partituras.
  - Exportador a formato MIDI.
- Interfaz gráfica de usuario: La solución más lógica ante la necesidad de la facilidad de uso es la de crear una interfaz de usuario amigable y fácil de comprender. A través de elementos gráficos el usuario podrá controlar los parámetros y acciones del compositor algorítmico. El hecho de tener interfaz gráfica acerca al usuario sin demasiados conocimientos informáticos a utilizar la aplicación, de forma que la curva de aprendizaje es menos pronunciada y con mejores resultados que utilizando una interfaz más compleja, como por ejemplo basada en comandos.
- Implementación de los métodos de composición algorítmica estudiados: Cada uno de los métodos será implementado de forma independiente y ocultas entre sí, facilitando su reaprovechamiento. Los métodos a implementar son los siguientes:
  - Compositor estocástico de sucesos independientes.
  - Compositor por cadenas de Markov.
  - Compositor por paseos aleatorios (Random Walk).
  - Compositor a partir de ruido fraccional.
  - Compositor por respuestas caóticas.
  - Compositor por desarrollo de motivos.
  - Compositor serialista.

- Parametrización del módulo compositor: La composición se encontrará parametrizada y será controlada por el usuario a través de la interfaz gráfica. El usuario podrá decidir tanto parámetros musicales generales, como específicos del compositor que esté utilizando. Las características generales parametrizables son las siguientes:

  - Rango de octavas, a través de elegir la octava mínima y máxima podrá abarcar el compositor.
  - Número de compases, para decidir la duración del fragmento.
  - Métrica, para tener control sobre el ritmo del fragmento.

- Escucha y visualización en tiempo real del fragmento compuesto: La solución más viable ante el problema de la visualización de la composición es la escucha directa. Una vez compuesto el fragmento musical, este podrá escucharse tantas veces como quiera el usuario a través de los controles clásicos de un reproductor multimedia (reproducir, pausar, parar). No se considera que el timbre del instrumento que reproducirá la composición sea decisivo, por lo que simplemente se utilizará un sintetizador a partir de muestras. Además de la escucha, se facilita al usuario una partitura animada de la composición.

- Exportación de los fragmentos musicales al formato SMF[1]: El formato más idóneo a la hora de exportar el fragmento musical compuesto es el SMF, debido a su calidad de estándar, y a su fácil manipulación por medio de otros entornos y sistemas musicales. Se da al usuario la posibilidad de exportar la totalidad del fragmento compuesto a SMF.

3.2.1 Flujo de la información

En el sistema la información principal es la composición, y la información previa a la misma (parámetros y opciones). Se describe a continuación el proceso que seguirá la información en el sistema:

1. Elección del método de composición algorítmica. Aún no existe la composición, el dato manejado es simplemente el método a utilizar.

2. Decisión de los parámetros musicales y específicos. El método elegido en el paso previo muestra los parámetros propios. La información de salida son las opciones decididas.

3. Composición del fragmento. La información del método a utilizar y las opciones específicas y musicales entran en el compositor, se obtiene como nueva información el fragmento musical.

[1] Standard MIDI File

4. Escucha y visualización. La información a tratar esta vez es el fragmento musical. Se puede considerar el fragmento musical a su vez como una entrada de realimentación hacia el punto 2.

5. Exportación del fragmento en un fichero tipo SMF. El fragmento musical se exporta fuera del entorno de la aplicación, como datos en formato SMF.

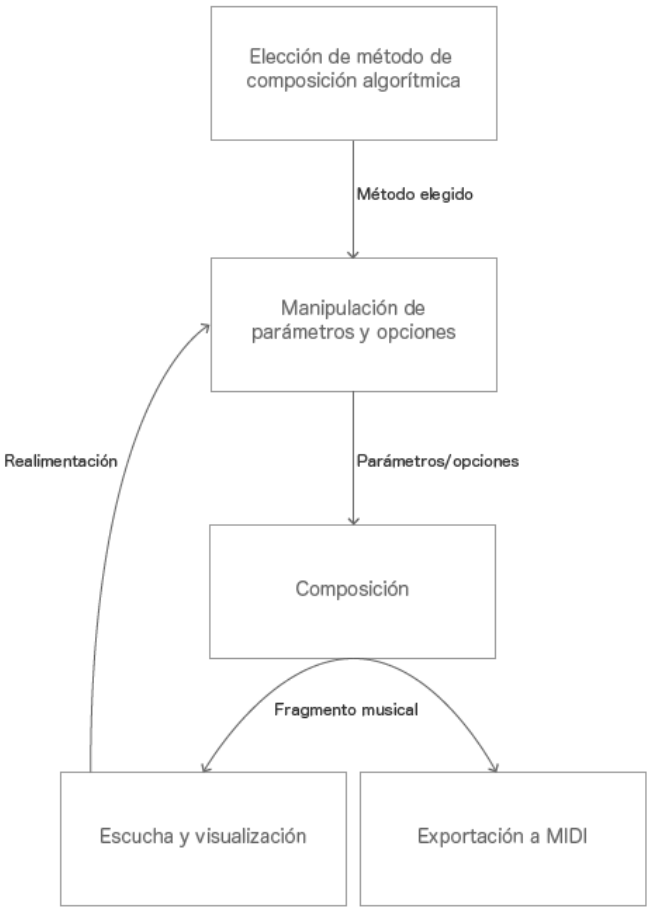


Figura 3.1: Diagrama de flujo de la información general del sistema

3.3 Descripción del diseño

Descripción general del diseño, como va a ser el sistema por dentro, diagrama de clases que va a tener, como se comunican las clases entre sí.

El sistema consiste en una aplicación que gestione tanto la composición de los fragmentos musicales, como su reproducción y visualización. Principalmente, se definen tres módulos diferentes, separados entre sí y que funcionan con independencia del resto de la aplicación.

Se define además una clase **App** que se comunica directamente con la librería openFrameworks, y mediante la implementación de las funciones requeridas por la librería, se obtiene la forma de controlar los ciclos de actualización, dibujado y eventos relacionados con la interfaz de usuario de la aplicación.

Clase Composer: Es una clase del tipo abstracta, que cada uno de los diferentes compositores que hereden de ella se encargará de implementar las funciones requeridas por la misma. Se encarga de componer un fragmento musical a partir de su duración en compases, su métrica y demás parámetros que se definirán según el tipo de compositor que se implemente. Básicamente, se invoca a una función **compose** que, implementada en las clases hijas, recibe los parámetros necesarios y compone un fragmento musical, que devolverá en forma de *array* a la clase principal (App).

Clase Player: Esta clase se encarga de reproducir un array de figuras musicales, gestionando sus duraciones, alturas y velocidades. Para ello controla un sintetizador por samples.

Clase Music Visualizer: Del mismo modo que la clase Player, el visualizador de partituras recibe un array de figuras musicales, y se encargará, utilizando openFrameworks de dibujar cada una de las figuras en un pentagrama en la ventana de la aplicación. Cada tipo de figura es una imagen, que el visualizador situará en la altura en píxeles determinada a partir de la altura musical de la figura.

La clase **App** contiene instancias de cada una de las tres clases explicadas anteriormente, y se encarga de comunicarlas entre sí y ordenar componer un fragmento y reproducirlo, todo ello a partir de los eventos de la interfaz de usuario. El diagrama UML de las clases generales puede verse en la figura 3.2.

Cabe explicar también la organización seguida en cuanto a la implementación de las figuras musicales. Se define una clase padre llamada **Figure**, que es heredada por dos clases, **Note** y **Silence**. La información principal de la figura, que es su duración, viene especificada en la clase **Figure**, mientras que información específica de nota, como su altura o velocidad vendrá especificada en la clase **Note**. Este diseño queda explicado en la figura 3.3.

Por último, conviene relativizar la información de las alturas a base 12, que representan las alturas musicales, para facilitar los cálculos y operaciones que realizará el compositor. De esta forma, en lugar de especificar la nota como frecuencia en Hz o altura MIDI, se hace con un número entre el 0 y el 11, que se corresponde como dicta la tabla 3.1, y su octava. Con este sistema

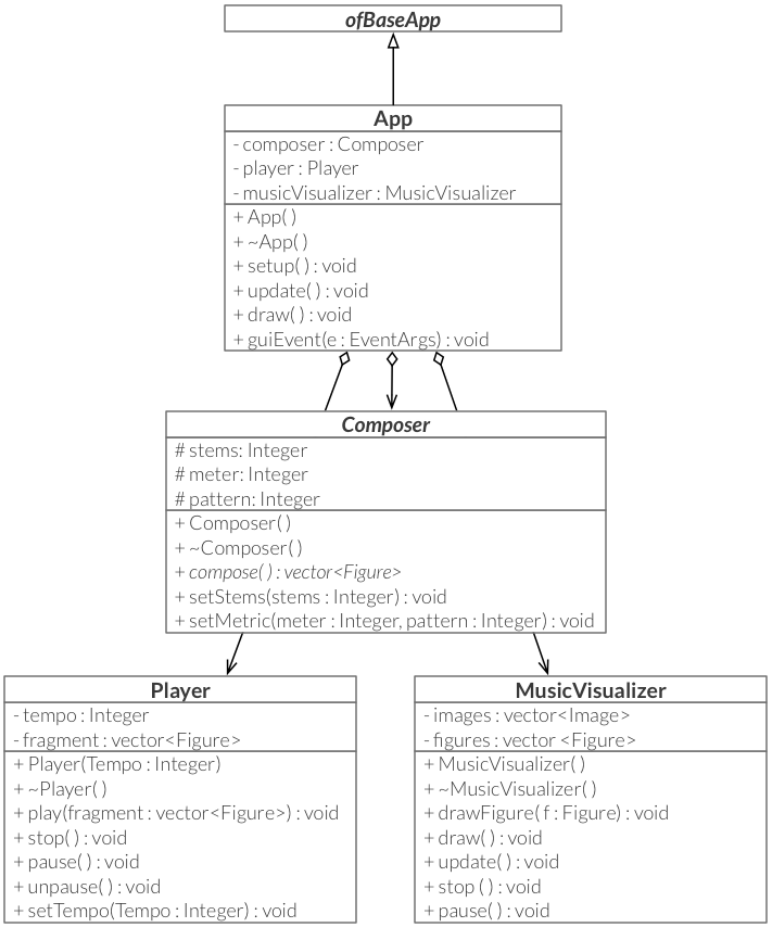


Figura 3.2: Diagrama de clases para la aplicación general.

de numeración, una escala quedará definida en su forma relativa como un vector de números enteros del 0 al 11, donde cada uno corresponde a una nota. Como ejemplo, la escala de Do Mayor es igual a:

$$DoMayor = \{0, 2, 4, 5, 7, 9, 11\}$$

Toda los vectores de las escalas se encuentran almacenados en un fichero **Scales.h** accesible desde cualquier clase que compone la aplicación.

Para conocer la altura MIDI correspondiente a la altura relativa y su octava basta con aplicar la siguiente fórmula:

$$Altura_{MIDI} = 12 * Octava + 24 + Altura_{relativa} \tag{3.1}$$

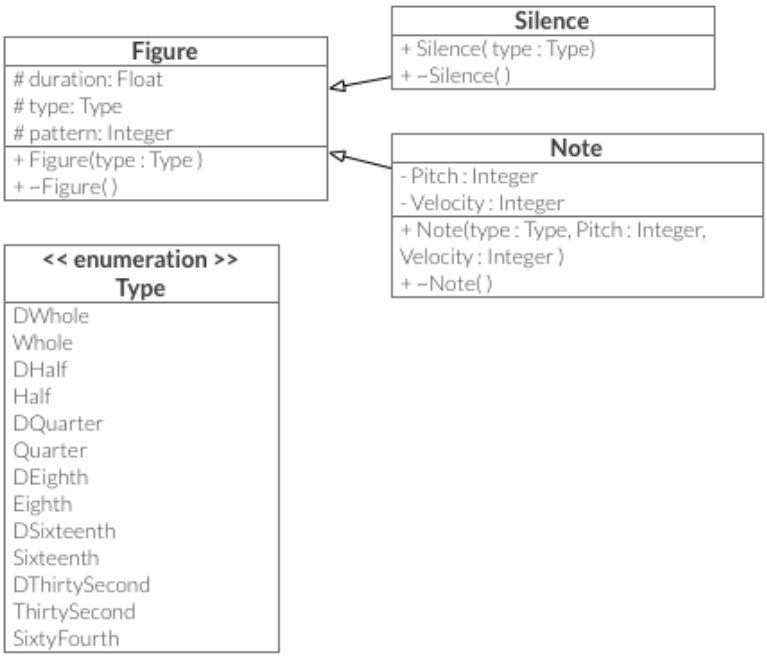


Figura 3.3: Diagrama de clases del diseño de las figuras musicales.

Do	Do♯	Re	Re♯	Mi	Fa	Fa♯	Sol	Sol♯	La	La♯	Si
0	1	2	3	4	5	6	7	8	9	10	11

Cuadro 3.1: Correspondencia entre números en base 12 y las alturas musicales.

3.3.1 Casos de uso

El único caso de uso de la aplicación es el de componer un fragmento musical. Para ello, volviendo al análisis del flujo de la información (3.2.1), es lógico que el comportamiento de la aplicación sea el mismo que el analizado, por lo que el caso de uso será el siguiente:

1. Usuario elige método de composición algorítmica de una lista.
2. A partir de las opciones desplegadas, usuario decide como mejor le convenga.
3. Usuario pulsa el botón **Compose**, generándose el fragmento musical.
4. Usuario pulsa botón **Play**, reproduciéndose el fragmento actual.
5. Usuario realiza acción de exportación a MIDI, o de borrado del fragmento.

3.3.2 Interfaz de usuario

De forma general, la interfaz de usuario debe cubrir el caso de uso principal, de modo que la tarea principal de la aplicación sea de lo más intuitiva para el usuario, y no le cause confusión. Para ello, se diferencian en la ventana de la aplicación cuatro bloques o partes, que corresponden con el flujo de trabajo (3.1). Así se define una interfaz de usuario *por pasos* de forma implícita[2], donde cada parte de la interfaz corresponde con un paso en el flujo de trabajo.

Como queda especificado en la figura 3.4, las cuatro partes que se diferencian son:

- Method: En este bloque se mostrará un listado con los métodos de composición algorítmica disponibles para su uso. El usuario sólo tendrá que pulsar en uno de ellos para poder continuar al siguiente paso.
- Options: Esta parte contiene los elementos de la interfaz de usuario que modifican parámetros del método de composición escogido, y por lo tanto, cambian al cambiar de método.
- Compose / Information: En este bloque se encuentran las acciones a realizar una vez se ha escogido el método y sus parámetros, y a su vez muestra un pequeño cuadro informativo que sirve como *feedback* sobre lo que está realizando el usuario.
- Visualization: Por último, en este bloque se muestra una vez el sistema ha compuesto el fragmento musical y se ha pulsado el botón de reproducción, una partitura animada que muestra las figuras correspondientes al fragmento, simultáneamente a su escucha.

Los bloques se han dispuesto en la ventana de forma que para el usuario sea más fácil recorrerlos con la mirada, de forma circular, comenzando en la esquina superior izquierda y finalizando en la esquina inferior izquierda, como se indica en la figura 3.5.

[2]No se trata de una interfaz guiada o tipo *wizard*, si no que el usuario sigue los pasos de una forma natural, sin ser guiado ni obligado a hacerlo.

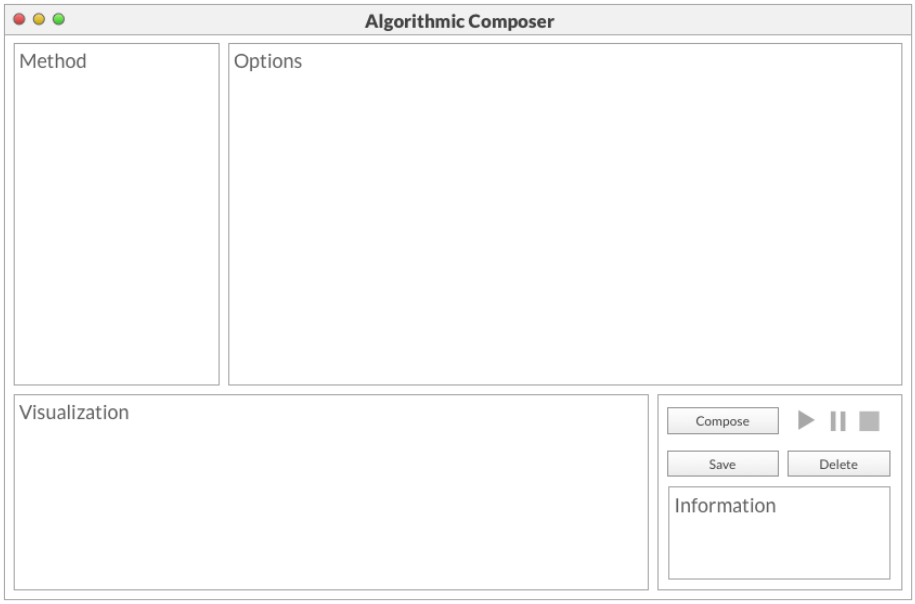


Figura 3.4: Boceto de la interfaz base de la aplicación.

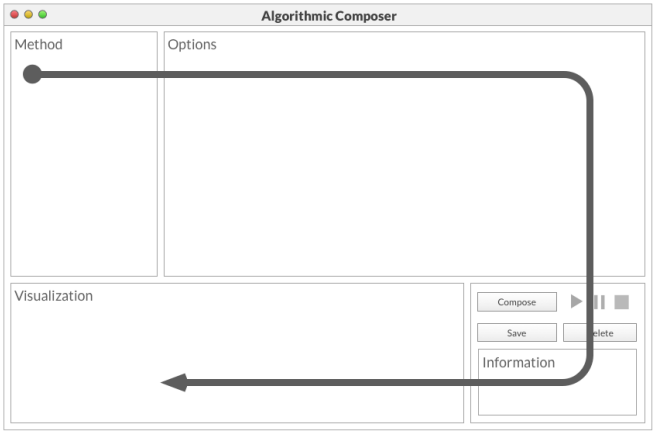


Figura 3.5: Flujo visual que se desea conseguir con la disposición de la interfaz diseñada.

4
Diseño del Sistema

4.1 Compositor estocástico de sucesos independientes

4.1.1 Análisis

El compositor estocástico de sucesos independientes es básicamente un generador de números aleatorios que serán convertidos al parámetro musical deseado. Además, se plantea tener varias distribuciones de probabilidad diferentes, para abrir más el abanico de posibilidades del compositor. La secuencia de numeros aleatorios generados puede ser utilizada sobre las alturas musicales, las duraciones de las figuras, o la relación entre el número de notas y el número de silencios en la composición, pudiéndose elegir una distribución diferente para cada uno de ellos. También resulta interesante la posibilidad de establecer alguno de estos parámetros a un valor fijo, para así obtener líneas melódicas con un ritmo fijo o líneas rítmicas con una altura fija.

Independiente de esto, también se necesita control sobre los parámetros musicales, como el rango de alturas, la cantidad de compases, o la métrica del fragmento que va a componerse. Basándose en estas premisas, el compositor estocástico de sucesos independientes necesita:

- Generador de números aleatorios según una distribución aleatoria.
- Posibilidad de elegir entre varias distribuciones aleatorias, que estarán

parametrizadas.

- Mapear la secuencia aleatoria a los parámetros musicales altura, duración o relación nota/silencio.
- Control independiente de parámetros musicales:
  - Rango de notas.
  - Número de compases y patrón rítmico.
  - Escala musical sobre la que elegir las alturas musicales.

4.1.2 Diseño

Se propone un compositor que contenga todas las características explicadas en el Análisis (4.1.1). Para continuar con el diseño del sistema general (capítulo 3) se realiza una implementación de la clase abstracta **Composer**, que se denominará **IndependentStochasticComposer**, y que por medio de la implementación del método virtual **compose**, se obtendrá el fragmento musical que el usuario desee, a partir de los parámetros de la interfaz de usuario (diagrama de clases en la figura 4.1).

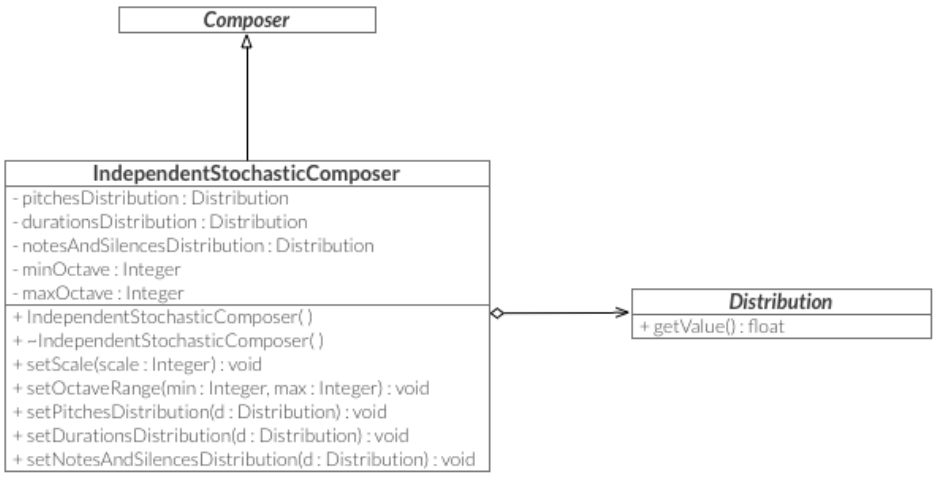


Figura 4.1: Diagrama general de clases del compositor estocástico de sucesos independientes.

A su vez, se propone un diseño para las distribuciones de probabilidad, que consiste en una clase abstracta llamada **Distribution**, y que implementarán cada una de las distribuciones vistas anteriormente (apartado 2.2.2) en una clase diferente, tal y como se puede ver en la figura 4.2.

El algoritmo simplificado en pseudocódigo para componer un fragmento musical a partir de los parámetros decididos por el usuario es el siguiente:

```
sea total : real = duracionPorCompas()
sea contador : real = 0.0
sea figura : booleano
vector fragmento
funcion componer
    para i = 0 hasta total con incremento +1
        mientras contador < total
            figura = (booleano)distribucionNotasYSilencios.obtenerValor()
            si !duracionFija
                duracion = distribucionDuraciones.obtenerValor()
                sea tests : entero = 0
                mientras duracion + contador > total
                    duracion = distribucionDuraciones.obtenerValor()
                    tests++
                    si tests = 20
                        duracion = total - contador
            si no
                duracion = duracionPorCompas()
            fin si
            sea tipo : Tipo = duracionATipo(duracion)

            contador = duracion + contador

            si figura == verdadero
                si !alturaFija
                    sea tono : entero
                    = (entero)distribucionAlturas.obtenerValor()
                    sea octava : entero
                    = (entero)distribucionAlturas.obtenerValor()

                    sea altura : entero = 12 * octava + 24 + tono
                si no
                    sea altura = alturaFija
                fin si

                sea nota : Nota = nuevo Nota(tipo, altura)
                añadir nota a fragmento
            si no
                sea silencio : Silencio = nuevo Silencio(tipo)
                añadir silencio a fragmento
            fin si
        contador = 0

devolver fragmento
```

Existe el problema a la hora de obtener el valor aleatorio y utilizarlo en cada uno de los parámetros musicales. La distribución aleatoria debe generar números en un determinado rango de valores, dependiendo de para qué se necesite el valor; si es para decidir entre nota y silencio, deberá elegir entre 2 valores, mientras que si es para decidir la altura de la nota, el rango será mayor (el rango de notas MIDI). Para resolver el problema, la distribución aleatoria siempre generará un número real entre 0 y 1, y existirá una función encargada de mapear este valor al rango de valores que se desee:

```
funcion mapearValor(valor : real, minimo : entero, maximo : entero)
    sea valorMapeado : entero = (valor * (maximo - minimo)) + minimo
    devolver valorMapeado
```

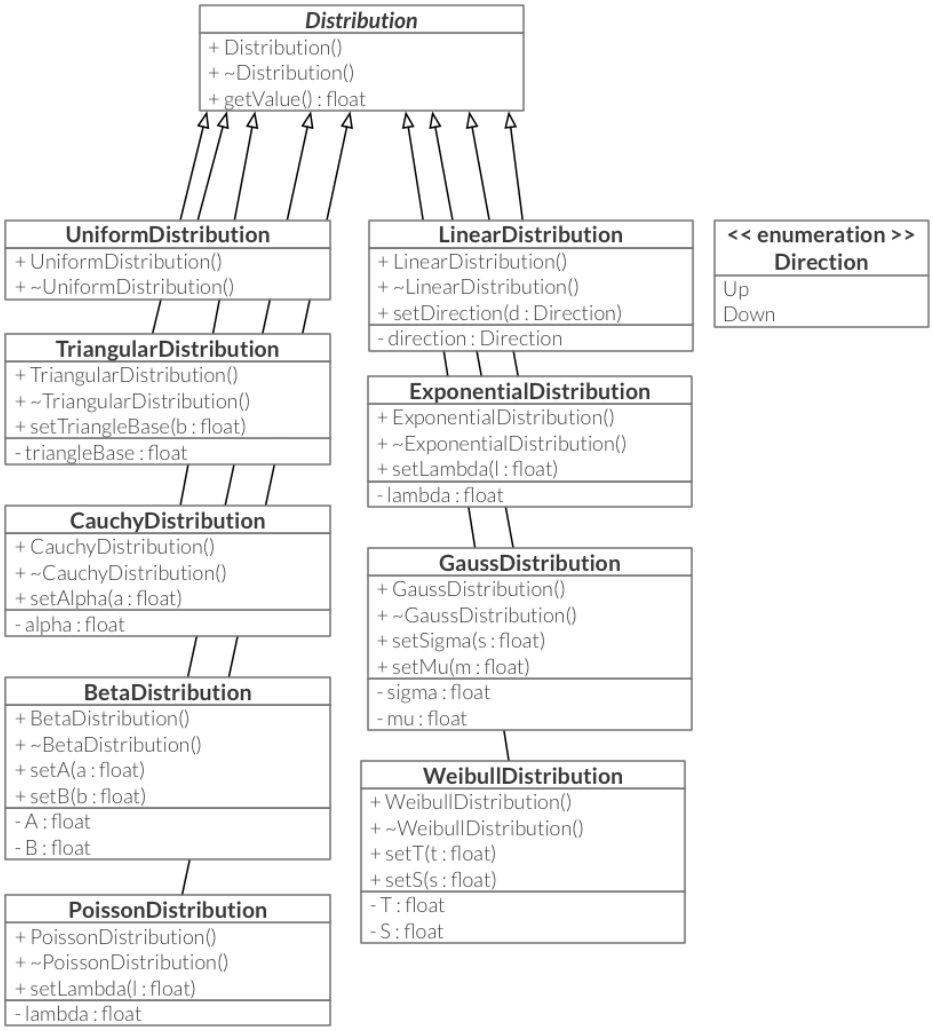


Figura 4.2: Diagrama de clases de las distribuciones de probabilidad.

4.2 Compositor por cadenas de Markov

4.2.1 Análisis

El compositor por cadenas de Markov es estocástico de sucesos dependientes, por lo que debe existir una fuente de la que tome las probabilidades

de los sucesos anteriores para obtener la secuencia musical. En este caso, la fuente será especificada por el usuario a partir de un fichero en formato SMF, y el compositor se encargará de calcular la matriz de transición entre estados para dicho fichero. Esta es la principal dificultad añadida a la hora de implementar el compositor por cadenas de Markov.

Para no aumentar la necesidad de memoria para la ejecución del programa, la matriz de transición obtenida será de primer orden (sólo se tomará el suceso inmediatamente anterior). Además, existirá una matriz de transición dedicada a las alturas y otra a las duraciones de las notas.

El compositor además tendrá parámetros musicales que el usuario modificará a su gusto, con la posibilidad de establecer a un valor fijo la altura o la duración de las notas, para aumentar las posibilidades el compositor, en el caso que el usuario sólo quiera obtener una melodía o un patrón rítmico.

A partir del análisis se obtienen los siguientes requisitos:

- Generador de números aleatorios a partir de una matriz de transición.
- Obtención de la matriz de transición a partir de un fichero SMF.
- Diferenciación entre matriz para las alturas y matriz para las duraciones.
- Control independiente de parámetros musicales:
  - Número de compases y métrica.
  - Octava y nota inicial.
  - Altura o duración fijas.

4.2.2 Diseño

Tal y como ocurre con el resto de compositores, el compositor por cadenas de Markov se implementa como una clase que hereda de la clase **Composer**, y que se denominará **MarkovChainsComposer** (figura 4.3).

El algoritmo de composición es similar al visto en el compositor estocástico de sucesos independientes, diferenciando que este compositor debe tener en cuenta la matriz de transición a la hora de escoger valores. Para escoger los valores aleatorios a partir de la matriz de transición se obtendrá un valor aleatorio real entre 0 y 1, y se comparará con los valores de la matriz de transición, como si estuvieran situados en una línea recta entre 0 y 1, y la posición del valor aleatorio abarque a una de las longitudes formadas por los valores de la matriz de transición. La altura o duración corresponderá con la posición de la matriz que abarca el valor aleatorio. Puede verse un ejemplo

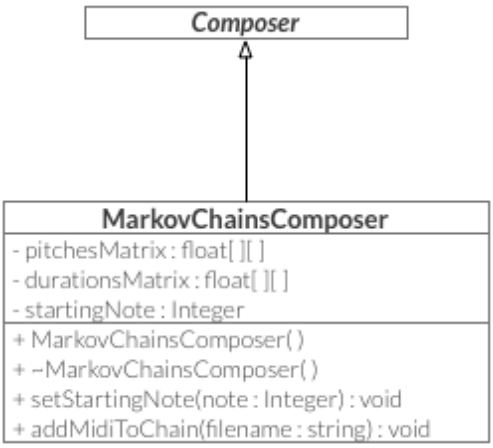


Figura 4.3: Diagrama de clases para el compositor por cadenas de Markov.

del procedimiento en la figura 4.4. El algoritmo en pseudocódigo seguido para este procedimiento es el siguiente:

```
sea tono : entero = -1
sea posicionAnterior : entero
sea posicion : entero = 0
sea suma : real = 0.0
sea valorAleatorio : real = obtenerValor()
sea encontrado : booleano = falso

mientras encontrado == falso y posicion < 11
    suma = suma + matrizAlturas[posicionAnterior][posicion]
    si valorAleatorio >= suma y
    suma + matrizAlturas[posicionAnterior][posicion + 1]
        encontrado = verdadero
        tono = posicion
        posicionAnterior = posicion
    fin si
fin mientras
```

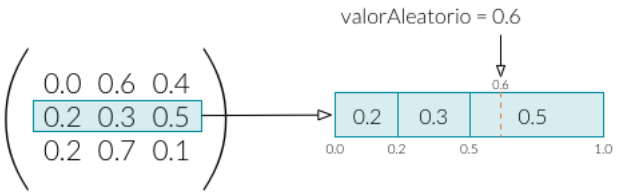


Figura 4.4: Ejemplo visual del algoritmo de obtención la posición de la matriz de transición a partir de un valor aleatorio.

Para el cálculo de matrices, simplemente se recorrerá el vector de figuras musicales y se incrementará en uno el valor de la posición de la matriz de

transición correspondiente cada vez que sea necesario (cuando se suceda una altura o duración B después de una altura o duración A, se incrementará en uno el valor en la posición (A,B) de la matriz). Finalmente, se deberán normalizar los valores de la matriz de transición, para que la suma de las filas sea siempre la unidad.

4.3 Compositor por paseos aleatorios

4.3.1 Análisis

El compositor por paseos aleatorios consiste en recorrer una cierta escala musical con pasos de una unidad hacia arriba o abajo en la escala, sin permitirse un paso de más longitud que la unidad. El usuario no tiene ningún control sobre el paseo aleatorio y las probabilidades del sistema estocástico se dividen en dos posibles estados: Dar un paso hacia arriba o hacia abajo en la escala. Una vez obtenida la dirección a partir del valor aleatorio, se obtiene la altura de la nota a partir de la nota anterior. Debe contemplarse, por tanto, la existencia de un límite superior y otro inferior, hasta donde el compositor algorítmico puede llegar.

A su vez, existe el problema de cómo actuar en el momento en el que el paseo alcance alguno de los límites. Como se explicó en el capítulo de Conocimientos Previos (en 2.2.2), se definen tres tipos de límites (reflectivo, elástico y absorbente), y el compositor actuará de una forma diferente para cada uno de ellos. Por temas de complejidad, se considera implementar los límites reflectivo y elástico.

También es necesario un control de los parámetros musicales que se estimen necesarios.

Tras el análisis, los requisitos necesarios para este compositor son:

- Generador de dirección a seguir por el paseo.
- Cálculo de la altura siguiente a partir de la anterior y de la escala musical escogida por el usuario.
- Comportamiento diferente según el tipo de límite escogido por el usuario (reflectivo y elástico).
- Control de los parámetros musicales:
  - Rango de notas (límites del paseo aleatorio).
  - Número de compases y patrón rítmico.

- Escala musical sobre la que actúa el paseo aleatorio.
- Altura inicial del paseo aleatorio.
- Duración de las notas del paseo aleatorio.

4.3.2 Diseño

El compositor por paseo aleatorio se implementará en la clase **RandomWalkComposer**, que heredará de la clase **Composer**, e implementará el algoritmo de composición en el método **compose**. El contenido de la clase puede verse en el diagrama 4.5.

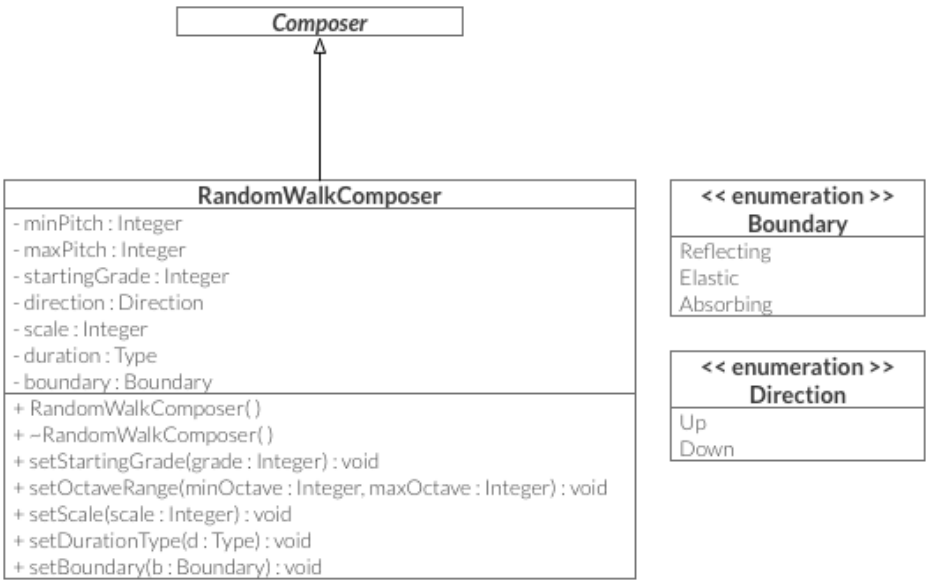


Figura 4.5: Diagrama de clases para el compositor por paseos aleatorios.

Al tener el sistema general una notación de alturas relativas (ver diseño del Sistema General, en sección 3.3), en este método de composición la obtención de la altura MIDI correspondiente a la altura relativa de un paso es relativamente complejo, ya que hay que tener en cuenta cuando se sucede un cambio de octava. Para facilitar este proceso y restar complejidad al algoritmo, se propone almacenar en un vector las alturas MIDI correspondientes a cierta escala musical y rango de notas, para que el paseo sea simplemente recorrer dicho vector.

La estructura del algoritmo es similar a la vista en los anteriores compositores, variando simplemente la forma de obtener las alturas musicales. Esa parte del algoritmo puede verse a continuación:

```
sea posicionAnterior : entero = posicionActual
```

```
sea direccion : Direction = (Direction) obtenerValor()

si direccion == Arriba
    posicionActual++
si direccion == Abajo
    posicionActual--
fin si

si limite == Reflectivo
    si posicionActual == vectorAlturas.tamaño
        posicionActual = posicionActual - 2
    si posicionActual == -1
        posicionActual = posicionActual + 2
    fin si
si limite == Absorbente
    si posicionActual == vectorAlturas.tamaño
        posicionActual = vectorAlturas.tamaño - 1
    si posicionActual == -1
        posicionActual = 0
    fin si
fin si

altura = vectorAlturas[posicionActual]
```

El resto de operaciones del algoritmo son similares a las vistas en los anteriores métodos de composición, donde entran en juego los parámetros escogidos por el usuario (cantidad de compases, métrica y rango de notas). No obstante, es importante recalcar que la nota inicial del paseo es escogida por el usuario a través de elegir la octava y la nota relativa inicial. Si la nota es más pequeña que el límite por debajo del paseo, la nota inicial será el límite por debajo, mientras que si la nota escogida por el usuario sobrepasa el límite por encima del paseo, la nota inicial será el límite superior.

4.4 Compositor a partir de ruido fraccional

4.4.1 Análisis

El compositor a partir de ruido fraccional consiste en obtener valores aleatorios del ruido fraccional, y utilizarlos para crear fragmentos musicales. Al utilizarse valores aleatorios, es un compositor del tipo estocástico, y el usuario juega un pequeño papel en él. Para poder tener un amplio abanico de posibilidades, el ruido fraccional está parametrizado. Como se explicó en 2.2.2, dependiendo del parámetro λ, se obtendrá una secuencia de números aleatorios más o menos dependiente de sí misma, por lo que hay que darle suficiente importancia a este parámetro en la implementación del compositor.

A su vez, la secuencia de números afectará a los dos parámetros musicales más determinantes de un fragmento, ritmo y altura. La secuencia aleatoria deberá poder afectarlas a las dos, o decidir a cual de ellas afecta. En el caso que el ritmo o la altura sean fijas, podrá decidirse la duración o altura fijas, donde proceda.

Igual que en los anteriores compositores, el usuario tiene que poder decidir el resto de parámetros musicales, como son la duración en compases y su métrica, el rango de octavas, y la escala musical sobre la que trabajará el compositor.

Los requisitos para el compositor son los siguientes:

- Generación de secuencias aleatorias a partir de ruido fraccional.
- Parametrización del ruido a partir de λ en la interfaz de usuario.
- Elección por parte del usuario de a qué afecta la secuencia aleatoria (ritmo y/o altura). Posibilidad de establecer a un valor fijo el ritmo o la altura del fragmento musical.
- Control de parámetros musicales:
  - Rango de alturas musicales.
  - Número de compases y patrón rítmico.
  - Escala musical sobre la que actúa el compositor.

4.4.2 Diseño

El compositor algorítmico a partir de ruido fraccional será implementado en la clase **FractionalNoiseComposer**, para tal y como ocurre con el resto de compositores implementados, mantener el paradigma de herencia de la clase **Composer**. El algoritmo de composición se encuentra en el método heredado **compose**. El diagrama de clases de la figura 4.6 contiene el diseño para el compositor.

La función **getFractionalNoiseSequence** será la encargada de obtener una secuencia de valores aleatorios de un tamaño determinado, comunicándose con la librería **COLORED_NOISE**, que devuelve una secuencia de números aleatorios de un determinado tamaño y con un valor de λ entre 0 y 2. Para poder utilizar los valores y mapearlos a las alturas musicales y/o duraciones, hay que normalizar la secuencia obtenida por la librería a un rango de valores entre 0 y 1. Una vez normalizada, el compositor puede utilizarla y obtener las alturas y duraciones deseadas, utilizando una versión muy similar del algoritmo visto en el compositor estocástico de sucesos independientes (4.1.2), sustituyendo el valor aleatorio por el valor de la secuencia normalizada que corresponda.

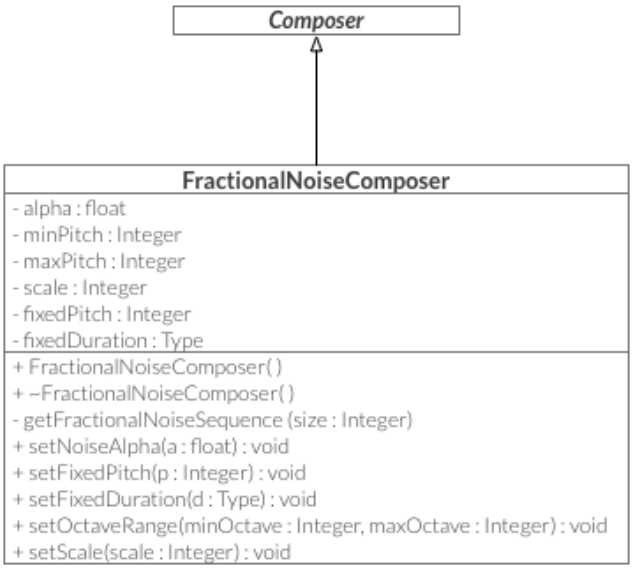


Figura 4.6: Diagrama de clases para el compositor a partir de ruido fraccional.

4.5 Compositor por respuestas caóticas

4.5.1 Análisis

El compositor algorítmico por respuestas caóticas crea fragmentos musicales a partir de la respuesta a procesos iterativos. Tal y como se explicó en 2.2.3, un sistema iterativo es aquel en el que la salida es utilizada como realimentación en la entrada del sistema. Existen multitud de sistemas iterativos, y una parte de ellos que generen una respuesta caótica, interesante para la composición algorítmica.

El compositor por respuestas caóticas deberá tener implementados varios sistemas iterativos que den una respuesta caótica, que será utilizada en la generación de alturas y/o duraciones musicales. El usuario podrá modificar los parámetros del sistema dinámico escogido, para así poder visualizar las diferentes respuestas posibles ante un único sistema.

El usuario podrá controlar también parámetros musicales necesarios en el compositor. En este caso no se incluye la cantidad de compases y el patrón rítmico, ya que resulta más útil para la evolución del sistema dinámico establecer el número de iteraciones del mismo en lugar que el número de compases.

Cabe decir que si el sistema dinámico es de una dimensión, esta afectará a la altura musical, y en el caso de ser bidimensional, afectará a la altura musical y a la duración de las notas. La duración en el caso de no verse afectada por el sistema, podrá seleccionarse de forma manual para todo el

fragmento. Por lo tanto, los requisitos que debe tener este compositor son los siguientes:

- Implementación de varios sistemas dinámicos que den respuestas caóticas.
- Parametrización manual de los sistemas a través de la interfaz de usuario.
- Medida del fragmento musical a partir del número de iteraciones del sistema.
- Posibilidad de selección de duración fijada a un valor en el caso de no verse afectada por el sistema.
- Control de parámetros musicales:
  - Rango de alturas musicales.
  - Escala musical sobre la que trabaja el compositor.

4.5.2 Diseño

La implementación del compositor por respuestas caóticas tiene algunas dificultades añadidas, ya que habrá que implementar algunas funciones dinámicas diferentes. La clase **ChaoticComposer** heredada de la clase **Composer** contiene la implementación del compositor. El algoritmo principal de composición es heredado e implementado de la clase **Composer**, (función **compose**). Se propone implementar las siguientes funciones dinámicas:

- **Función logística**

$$x_{n+1} = rx_n(1 - x_n)$$

- **Función de Mandelbrot**

$$x_{n+1} = x_n^2 + c$$

- **Función Tienda**

$$x_{n+1} = \begin{cases} rx & 0 \leq x < \frac{1}{2} \\ \\ r(1-x) & \frac{1}{2} \leq x \leq 1 \end{cases}$$

- **Función de Hénon**

$$x_{n+1} = -ax_n^2 + y_n + 1$$
$$y_{n+1} = bx_n$$

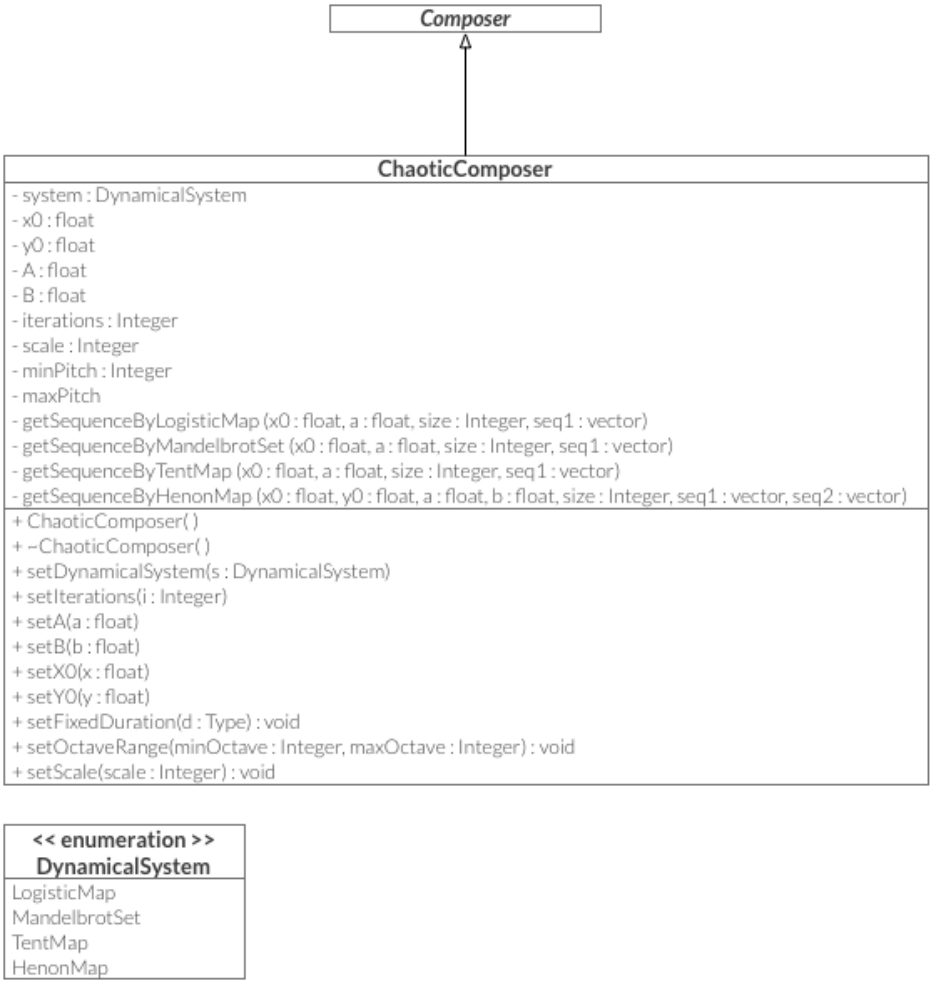


Figura 4.7: Diagrama de clases para el compositor por respuestas caóticas.

El diagrama de clases para el compositor por respuestas caóticas se encuentra en la figura 4.7.

Como las funciones se controlan con un único parámetro si son de una dimensión y con dos si es de dos dimensiones, se propone utilizar dos parámetros genéricos **A** y **B**, para restar complejidad y uso de variables con nombre diferentes pero con mismo uso en la implementación del compositor.

Otra cuestión de la implementación es mantener controlados los límites de la respuesta del compositor, para que está pueda ser mapeada a las alturas o duraciones musicales sin obtener un error de valor fuera de rango. Es por esto por lo que se pide un número exacto de iteraciones, para así obtener un vector todas las respuestas iteradas del sistema y que este pueda ser normalizado a valores entre 0 y 1.

La algoritmo principal **compose** es similar a los vistos en los anteriores compositores, con la peculiaridad de que en lugar de obtener notas musicales

hasta completar un determinado número de compases, simplemente se itera en bucle y se obtiene notas musicales hasta completar el número de iteraciones solicitadas por el usuario, por lo que se obtendrán tantas notas como iteraciones tiene el sistema dinámico.

4.6 Compositor por desarrollo motívico

4.6.1 Análisis

El compositor por desarrollo motívico consiste en aplicarle una serie de transformaciones a un motivo musical. El fragmento musical de salida del compositor es la secuencia de transformaciones sobre el motivo. Este compositor sigue un patrón diferente de los anteriores compositores; en los anteriores compositores se genera una salida en la que el usuario apenas tiene control, mientras que este compositor se caracteriza por tener el usuario mayor grado de intervención sobre los parámetros que afectan a la composición.

En el desarrollo motívico primeramente es necesario un motivo musical sobre el que realizar las transformaciones. Este pequeño fragmento puede provenir de varias fuentes. Se propone que el usuario pueda introducirlo de forma manual, a partir de un fichero MIDI, o incluso que sea generado aleatoriamente. De esta forma, se da multitud de posibilidades al usuario de composición, ya sea a partir de un motivo generado aleatoriamente, o a partir de un motivo decidido por él mismo, melódico, rítmico o ambos. La longitud del motivo no deberá excederse de diez figuras musicales, ya que al ser mayor, se pierde la característica de motivo.

Por otra parte, es necesaria la implementación de todas las posibles transformaciones estudiadas en 2.2.3 que pueden aplicarse a un motivo. Para dar mayor implicación al usuario en el compositor, este decidirá el orden de las transformaciones sobre el motivo musical.

Por lo tanto, tras el estudio, los requerimientos del compositor son:

- Compositor musical a partir de unir transformaciones sobre un motivo musical.
- Implementación de las transformaciones aplicables a un motivo musical, con los parámetros que exigen cada una de ellas.
- Obtención de un motivo de no más de diez figuras por diferentes medios:
  - Generado aleatoriamente.
  - A partir de un fichero MIDI estándar.

    - Introducido manualmente por el usuario a través de la interfaz gráfica.

- Aplicación manual de las transformaciones sobre el motivo. El usuario escoge las transformaciones a aplicar y el orden de la secuencia de las mismas.

4.6.2 Diseño

Este compositor dista en su diseño de los anteriores, donde se generaba un fragmento musical hasta completar cierta cantidad de compases o cierto número de iteraciones. En este caso, el fragmento musical se crea a partir de transformaciones sobre un motivo musical, por lo que el algoritmo principal de composición será diferente de los anteriores. La implementación del compositor queda en la clase **MotivicDevelopmentComposer**, clase que, de igual forma que en los anteriores compositores, hereda de la clase **Composer**, y el algoritmo principal de composición se encuentra en el método **compose**.

Existen ciertos problemas a solucionar en la implementación de este compositor. El primero de ellos reside en cómo diseñar e implementar el concepto de transformaciones sobre un motivo. La solución más lógica y sencilla es crear una clase **Method**, con un método **performTransformation** a implementar en las clases que hereden de esta. De esta forma, las clases que hereden de **Method** tendrán obligación de implementar **performTransformation**, cada una con la transformación deseada.

El diagrama de clases de este diseño puede encontrarse en la figura 4.8.

La implementación de cada una de las transformaciones es muy sencilla, ya que consiste básicamente en aplicar sumas y restas sobre la altura, el intervalo o la duración de las figuras que componen el motivo, por lo que no se va a explicar en profundidad.

Por otra parte, ahora el algoritmo principal **compose** es un tanto diferente al visto anteriormente. El algoritmo tiene que ir aplicando las transformaciones una por una en el orden que el usuario ha solicitado, e ir almacenando la respuesta en un único vector de figuras, que será el fragmento musical creado. Este proceso se puede ver en pseudocódigo a continuación:

```
vector<Figura> fragmento
fragmento.insertar(motivo)
funcion componer
    para i = 0 hasta sequencia.tamaño con incremento +1
        sea vector<Figura> transformacion =
        secuencia[i].aplicarTransformacion(motivo)
        fragmento.insertar(transformacion)
    fin para
devolver fragmento
```

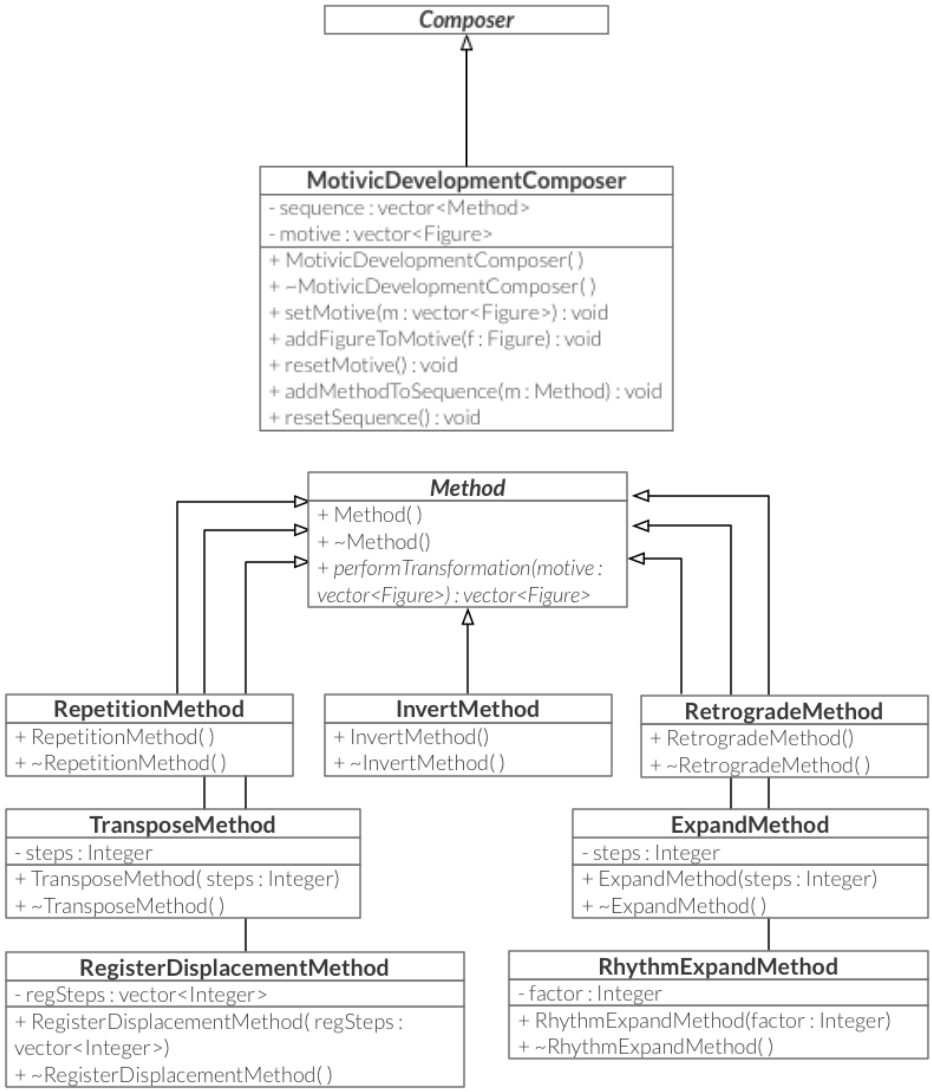


Figura 4.8: Diagrama de clases para el compositor por desarrollo motívico.

Por último, el diseño de la interfaz gráfica para este compositor es un tanto más complejo que en los anteriores compositores. Hay que distinguir dos partes en la interfaz; la primera referida a la creación del motivo principal, en la que podrá ser elegido aleatoriamente [1], a partir de un fichero SMF, tomando las diez primeras figuras del mismo, o introduciendo manualmente una por una las figuras del motivo. En este último caso, el usuario tendrá que elegir entre nota o silencio, la duración de la figura y su altura si es una nota.

[1]Para la generación aleatoria del motivo se utiliza el compositor estocástico de sucesos independientes con una distribución uniforme. No se considera dotarle de más opciones y parámetros a esta parte por ser sólo uno de los tres posibles métodos de generación de motivos.

Puede verse el detalle de la interfaz en la figura 4.9.

Figura 4.9: Detalles de la interfaz gráfica referidos a las formas de creación de motivos musicales.

La segunda parte de la interfaz es la referida a las transformaciones aplicables al motivo. Estas se mostrarán con sus parámetros en un listado. El usuario escogerá las que desee a través de un botón de añadir a la secuencia. La secuencia de transformaciones que se vaya creando también se muestra, para dar información al usuario de lo que está haciendo. El detalle de la interfaz referida a la elección de las transformaciones y su visualización puede verse en la figura 4.10.

4.7 Compositor Serialista

4.7.1 Análisis

El compositor serialista es bastante similar al compositor por desarrollo motívico. Cabe decir que este compositor se concibe como un generador de

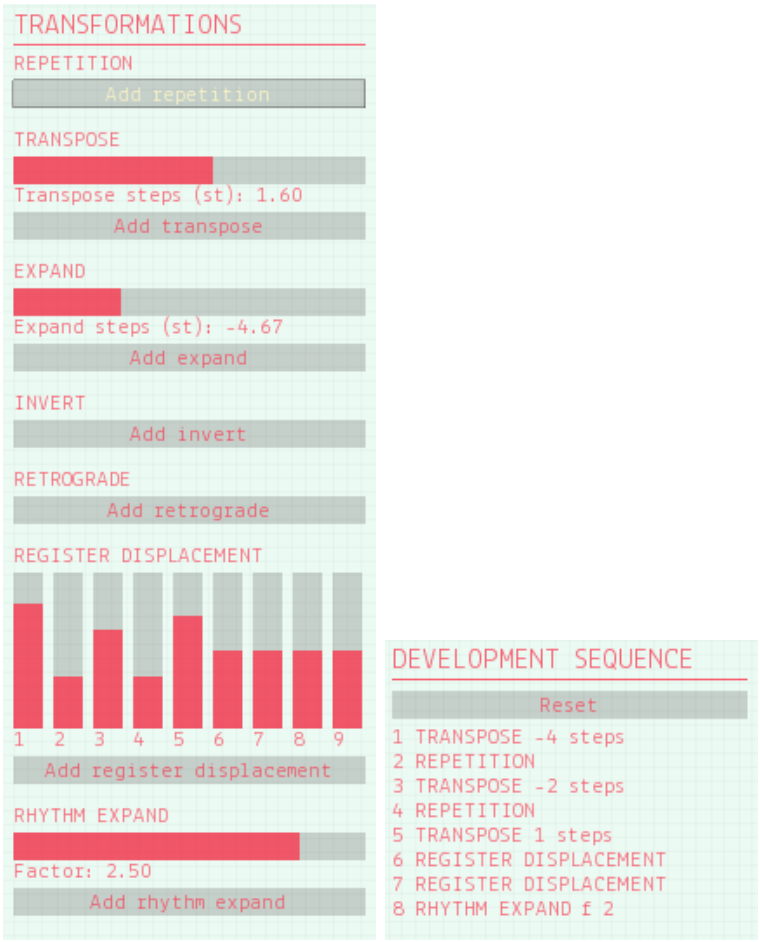


Figura 4.10: Detalles de la interfaz gráfica referidos a la elección de las transformaciones que afectarán al motivo, y el listado de las que ya han sido añadidas a la secuencia.

series que serán utilizadas por el compositor. El compositor serialista es también compositor dodecafónico, ya que como se vió en 2.2.3, el dodecafonismo es aplicar el principio serial sólo a las alturas musicales.

En este compositor va a limitarse a obtener series melódicas y rítmicas. No se incluyen el resto de parámetros musicales, como la dinámica o la articulación por considerarse suficiente para el propósito del sistema con incluir sólo el principio serial en la altura y el ritmo.

Es importante que el compositor pueda actuar únicamente sobre las alturas o el ritmo, para dar al usuario más libertad compositiva. Para ello es necesaria una opción de mantener a un valor fijo las alturas musicales o el ritmo.

Las necesidades del compositor serialista se van a basar en el proceso de composición serialista visto en 2.2.3. En primer lugar, el usuario ha de decidir el orden que quiere que tenga la serie de tonos y de figuras rítmicas.

A continuación, el usuario elegirá que operaciones aplicar a la serie que ha creado, y que el compositor le genere las series con estas transformaciones.

Por lo tanto, el compositor serialista debe incluir:

- Generador de series rítmicas y melódicas.
- Capacidad de dar al usuario la decisión de elegir el orden de las series originales.
- Implementación de todas las transformaciones seriales definidas.
- Posibilidad de elegir un valor fijo en las alturas musicales o ritmo.

4.7.2 Diseño

El diseño de este compositor también es bastante similar al del compositor por desarrollo motívico, en el que cambiaba drásticamente la forma de componer un fragmento musical. Este compositor generará series mediante transformaciones elegidas por el usuario, por esto se va a seguir la misma forma de componer que en el compositor por desarrollo motívico. La clase que contiene el compositor se denomina **SerialistComposer**, que igualmente que los anteriores compositores, hereda de la clase **Composer**. El método principal de composición lo hereda de la clase padre y se llama **compose**.

En este compositor se extraen dos fases principales: la primera es generar las series originales, y la segunda aplicar las transformaciones sobre estas series para obtener las series transformadas.

Para generar las series originales, en primer lugar el usuario deberá decidir el orden de las doce alturas musicales y/o las doce figuras rítmicas, a las que se llamarán **clases melódicas** y **clases rítmicas**. Si el usuario decide omitir este paso, las clases melódicas seguirán el orden de la escala cromática y las clases rítmicas se sucederán de mayor a menor duración. Es en este momento en el que se podrá establecer a una altura o a una figura rítmica fija. A continuación, el usuario debe elegir el orden de la serie original, que es el orden en el que aparecerán las clases melódicas y rítmicas, y es el mismo para ambas.

La clase melódica es expresada como un vector de números enteros que especifican la altura MIDI que ha escogido el usuario, mientras que la clase rítmica se entiende como un vector de tipos de figuras rítmicas. La serie original es simplemente un vector de enteros del 0 al 12. En ninguno de estos vectores puede haber un objeto repetido, y el vector siempre tiene que ser de tamaño 12.

Por otra parte, se deben implementar las cuatro transformaciones seriales estudiadas en 2.2.3. Para ello se seguirá el mismo patrón de diseño utilizado en el compositor por desarrollo motívico, donde las transformaciones heredan de una clase principal un método que deberán implementar. La clase principal se llama **SMethod**, que contiene un método **performTransformation**;

las clases que hereden de **SMethod** tendrán obligación de implementarlo. Las transformaciones seriales son simplemente transformaciones matemáticas en módulo 12, por lo que pueden ser aplicadas de forma indiferente a la serie melódica y a la serie rítmica. Por ello, se le debe dar al usuario la capacidad de decidir si una transformación es aplicada a la serie melódica, rítmica o a ambas a la vez. El contenido de las clases explicadas puede verse en la figura 4.11.

Por último, destacar el diseño de la interfaz de usuario. En este compositor el usuario juega un papel más importante que en el resto, por lo que hay que facilitarle una interfaz sencilla dentro de la complejidad del método. Para ello, la interfaz se divide en dos partes: la parte de elección de la serie original, y la de elección de la secuencia de transformaciones sobre la serie. En la primera parte queda definido mediante botones cómo el usuario debe elegir las clases melódica y rítmica y la posterior serie original a partir de estas clases, como puede verse en la figura 4.12.

En cuanto a la parte de elegir la secuencia de transformaciones que afectarán a la serie original, esta parte de la interfaz de usuario es bastante similar a la del compositor por desarrollo motívico (4.10), donde se muestra las posibles transformaciones, a qué serie afecta, y un listado de la secuencia que será aplicada, como puede verse en la figura 4.13.

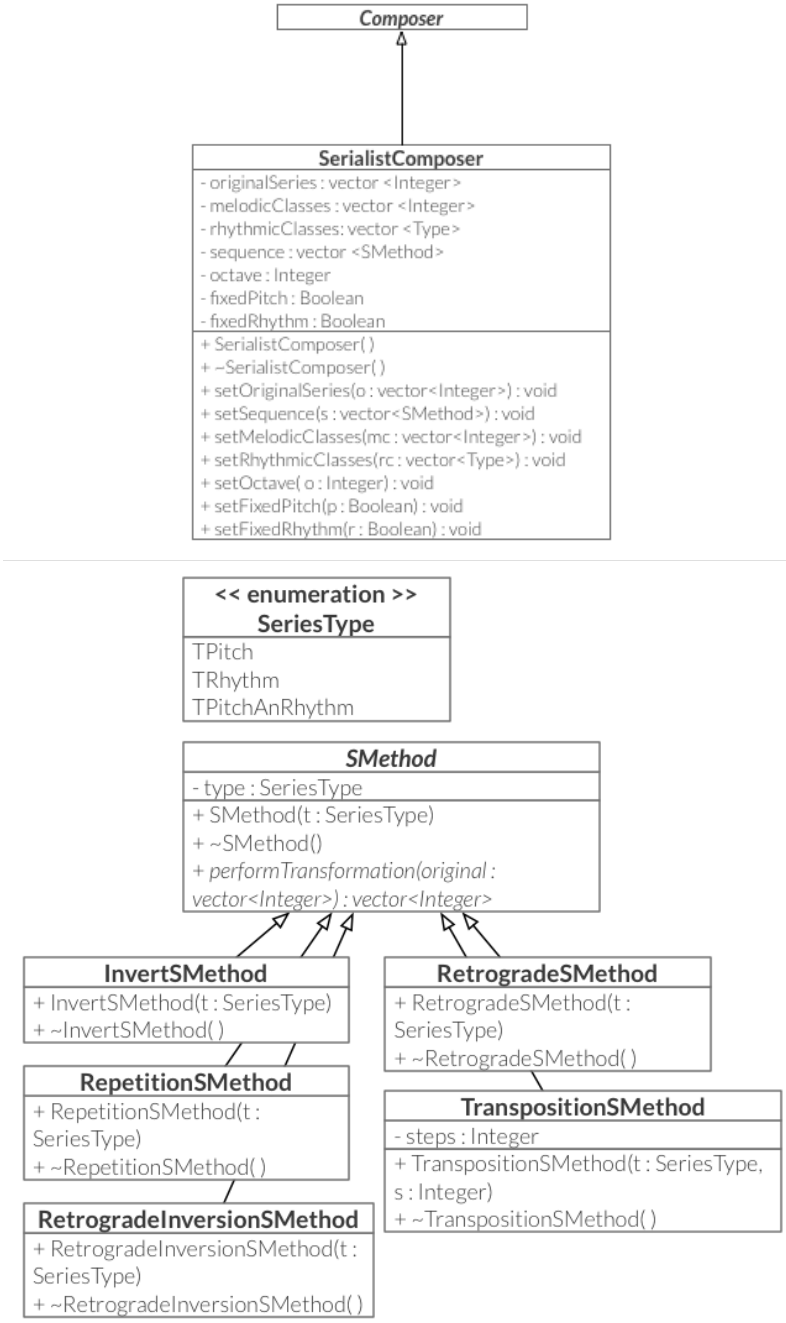


Figura 4.11: Diagrama de clases para el compositor serialista.

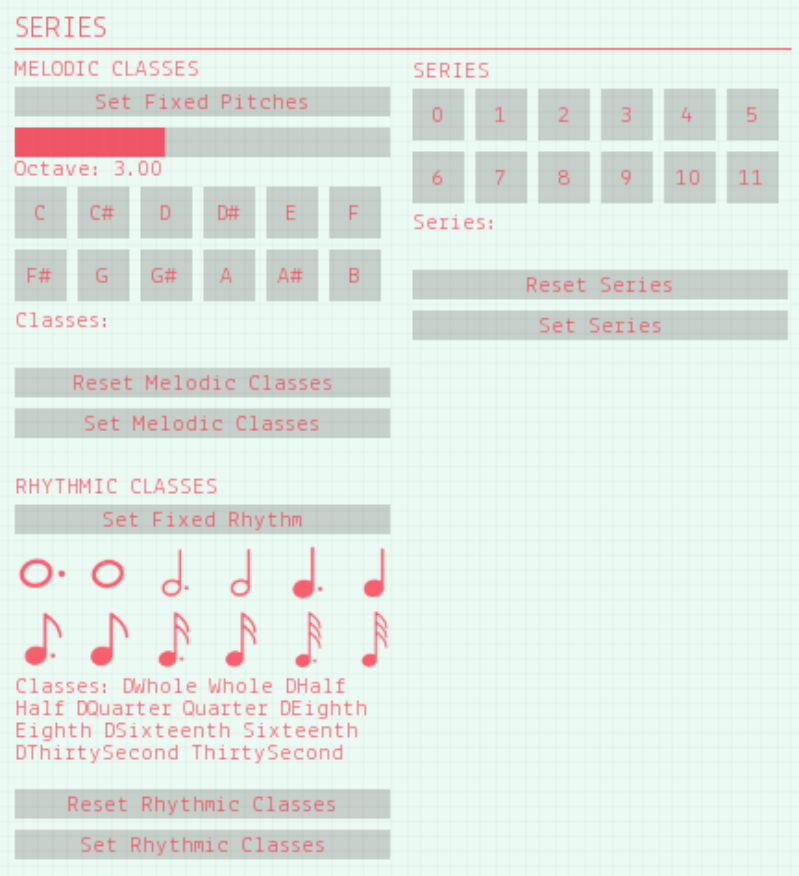


Figura 4.12: Detalle de la interfaz referido a la selección de las clases melódica y rítmica, y de la serie original.

Figura 4.13: Detalle de la interfaz referido a la selección de la secuencia de transformaciones sobre la serie original.

5 Resultados

5.1 Ejemplos

A continuación se van a enumerar algunos ejemplos para cada uno de los compositores implementados. Se variarán los parámetros de los compositores para ver las diferente salidas que pueden generar. El sistema tiene aún más posibilidades de composición que las que se van a ver a continuación, aunque no se considera necesario abarcarlas todas, por esto únicamente se incluyen algunos fragmentos muy básicos y obvios que pueden obtenerse a partir de manejar muy poco el compositor.

5.1.1 Compositor estocástico de sucesos independientes

El compositor estocástico de sucesos independientes produce muy variados fragmentos musicales según que distribuciones y parámetros se utilicen. En el ejemplo 5.1 se utiliza una distribución triangular para las alturas, obteniéndose valores intermedios de entre dos octavas, distribución exponencial para las duraciones, favoreciendo la aparición de figuras de larga duración (valores más bajos producen notas más largas) y distribución exponencial para la cantidad de notas frente a silencios, produciéndose más notas que silencios (las notas aparecen con valores bajos, los silencios con valores altos).

En el ejemplo 5.2 se ha utilizado una distribución gaussiana centrada en 0.8 y con dispersión de 0.2, produciéndose valores intermedios para todos los parámetros musicales.

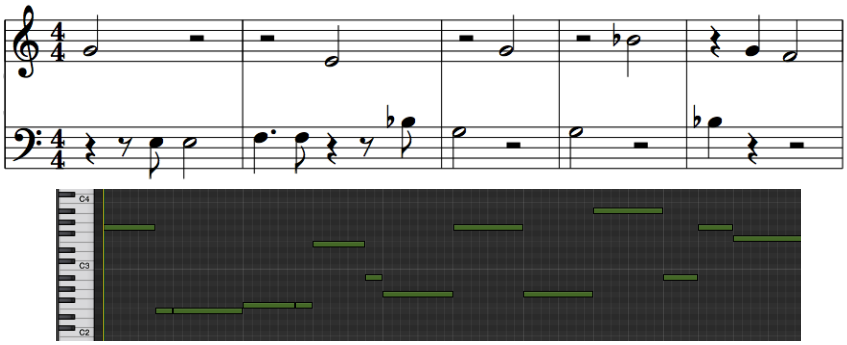


Figura 5.1: Fragmento musical generado por el compositor estocástico de sucesos independientes.

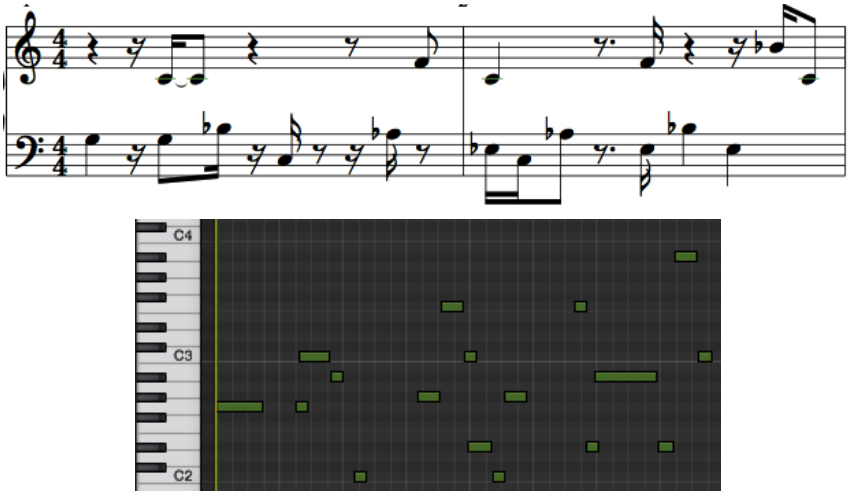


Figura 5.2: Fragmento musical generado por el compositor estocástico de sucesos independientes.

5.1.2 Compositor por cadenas de Markov

En el compositor por cadenas de Markov pueden apreciarse diferencias en los fragmentos generados a partir de según qué obras. El ejemplo 5.3 genera una salida a partir de *La Primavera de Las Cuatro Estaciones* de *Antonio Vivaldi*.

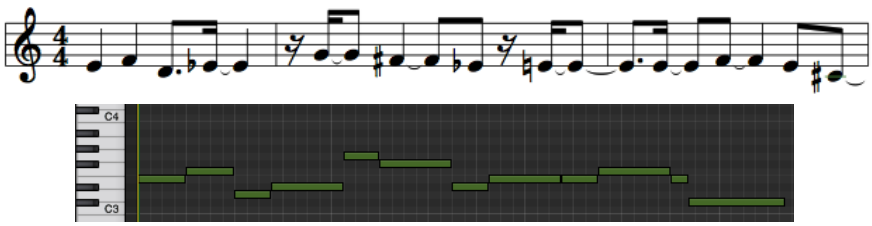


Figura 5.3: Fragmento musical compuesto por cadenas de Markov de primer orden.

En el ejemplo 5.4 se ha utilizado como obra de la que formar la cadena de Markov el *Ragtime The Entertainer* de *Scott Joplin*. Nótese la presencia de síncopas en el fragmento, como característica principal del ritmo Ragtime.

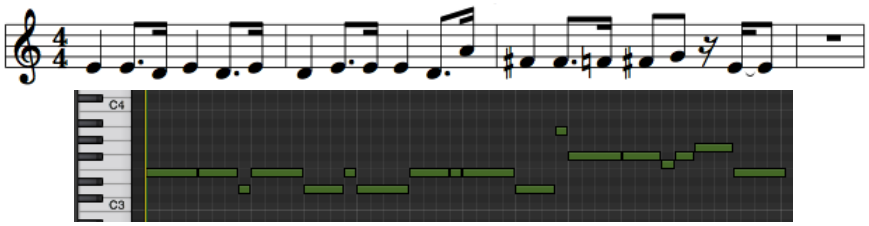


Figura 5.4: Fragmento musical compuesto por cadenas de Markov de primer orden.

El ejemplo 5.5 está compuesto a partir del *Nocturno No.2 Op.9* de *Frederic Chopin*. Se destaca de este fragmento la presencia de notas de corta duración en consonancia con notas de larga duración, como ocurre en la obra de la que se ha aprendido.

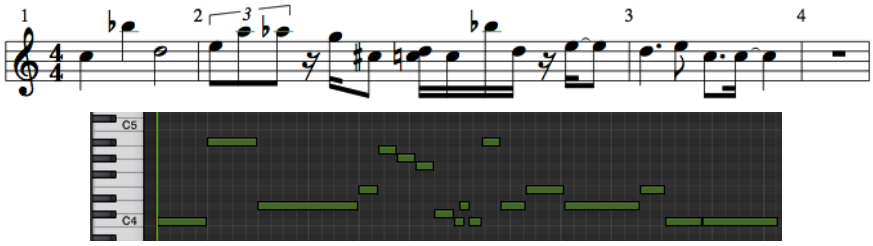


Figura 5.5: Fragmento musical compuesto por cadenas de Markov de primer orden.

5.1.3 Compositor por paseo aleatorio

Los fragmentos 5.6, 5.7 y 5.8 se han generado a partir de un paseo aleatorio utilizando diferentes escalas y duraciones de notas.

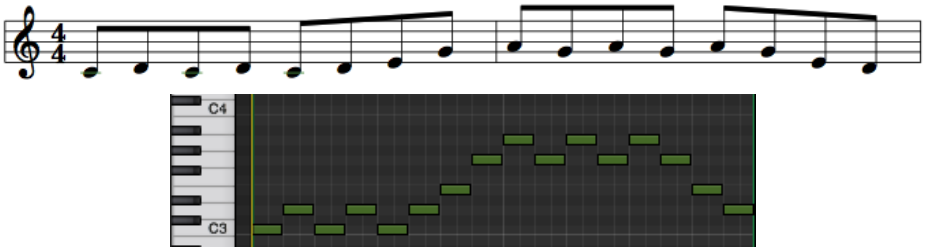


Figura 5.6: Fragmento musical compuesto por paseo aleatorio sobre la escala pentatónica y con figuras de duración corchea.

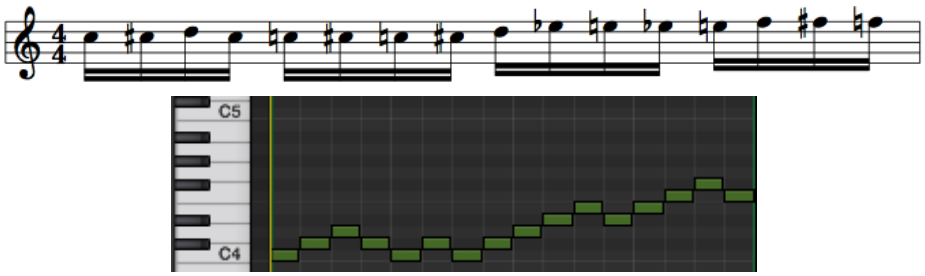


Figura 5.7: Fragmento musical compuesto por paseo aleatorio sobre la escala cromática y con figuras de duración semicorchea.

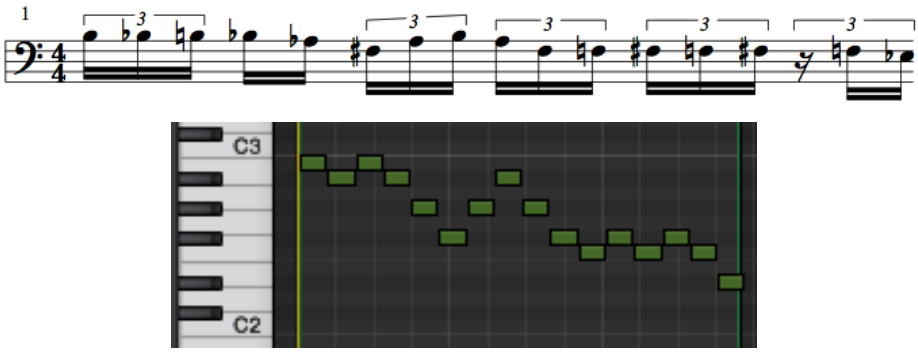


Figura 5.8: Fragmento musical compuesto por paseo aleatorio sobre la escala de Re Menor Armónica y con figuras de duración fusa con puntillo.

5.1.4 Compositor a partir de ruido fraccional

El compositor a partir de ruido fraccional genera unos fragmentos interesantes. Los ejemplos que van a verse a continuación han sido generados con diferentes valores de λ afectando únicamente a las alturas musicales y utilizando la escala de Do Mayor. El primer fragmento (5.9) se ha generado utilizando ruido fraccional con $\lambda = 0$, por lo que se obtiene una composición en la que no se aprecia ningún patrón o coherencia musical.

El fragmento 5.10 ha sido compuesto con ruido fraccional con $\lambda = 1$, y puede apreciarse cierta coherencia musical, ya que las notas anteriores influyen en la generación de las notas posteriores.

Por último, el ejemplo 5.11 ha sido generado con ruido fraccional con $\lambda = 2$. Puede verse que existe una gran influencia de la nota inmediatamente anterior sobre la posterior, por lo que el patrón es muy similar y repetitivo.

5.1.5 Compositor por respuestas caóticas

El compositor por respuestas caóticas tiene un amplio abanico de salidas en función de los parámetros de la ecuación utilizada. Los fragmentos 5.12 y 5.13 se han generado a partir de la respuesta de la ecuación logística. El

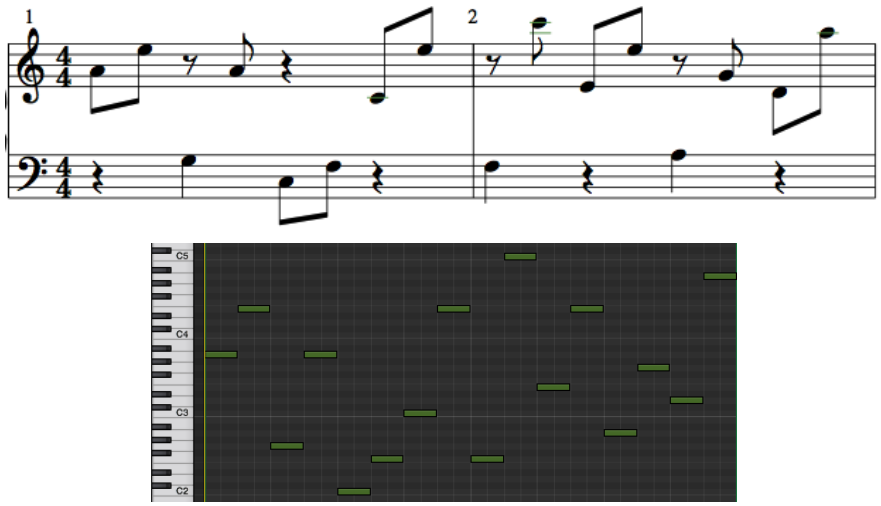


Figura 5.9: Fragmento musical compuesto a partir de ruido fraccional con $\lambda = 0$.

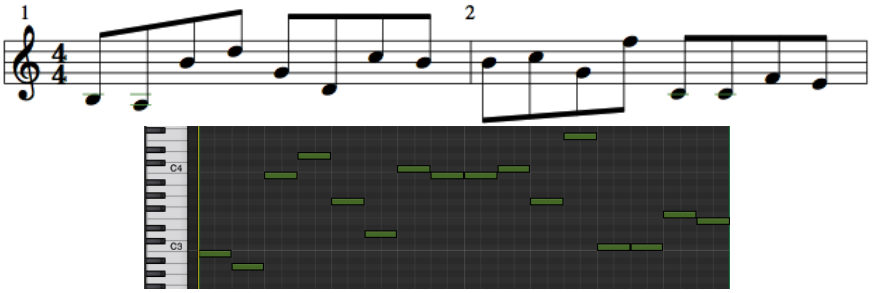


Figura 5.10: Fragmento musical compuesto a partir de ruido fraccional con $\lambda = 1$.

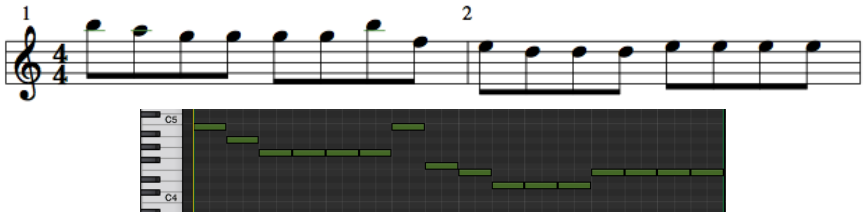


Figura 5.11: Fragmento musical compuesto a partir de ruido fraccional con $\lambda = 2$.

primer fragmento tiene como $r = 3.37$, y la salida que genera oscila entre pocos valores, no llega a ser caótica. El segundo fragmento, por el contrario, genera una salida que fluctúa entre más valores y de la que apenas se puede apreciar un patrón repetitivo, por lo que se puede decir que es caótica.

El ejemplo 5.14 se ha compuesto a partir de la respuesta a la función de Hénon con los parámetros $a = 1.4b = 0.3$. Al ser un sistema de dos dimensiones, el compositor toma la primera dimensión para afectar a las alturas y la segunda para las duraciones de las notas.

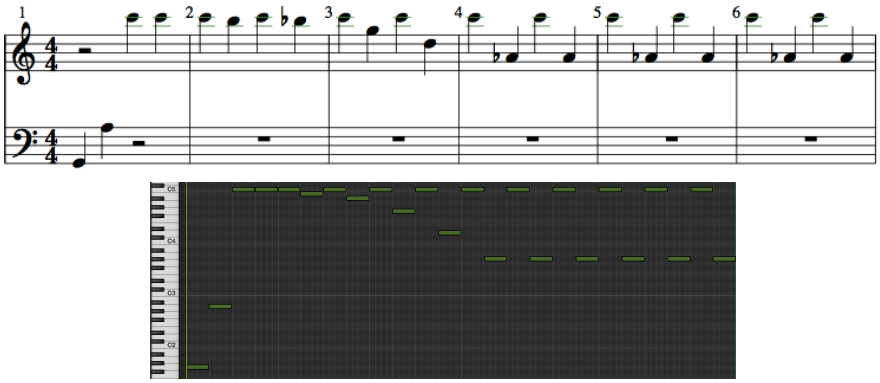


Figura 5.12: Fragmento musical compuesto a partir de la respuesta de la ecuación logística para $r = 3.37$.

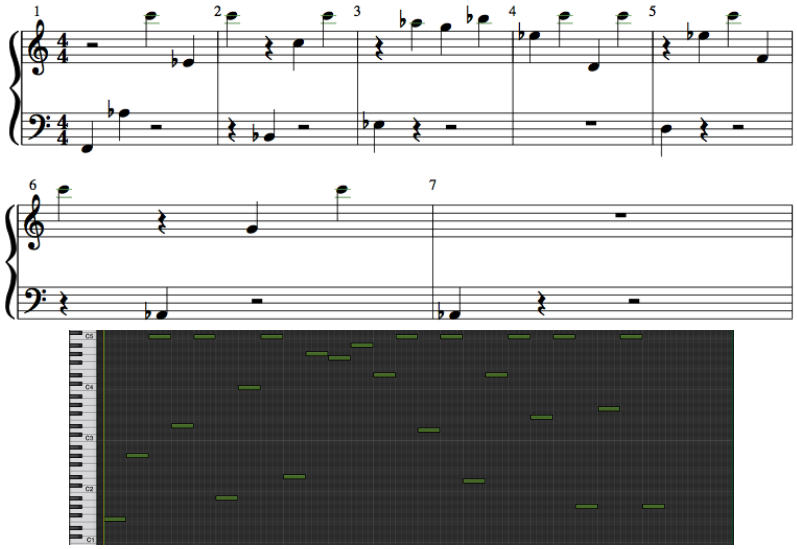


Figura 5.13: Fragmento musical compuesto a partir de la respuesta de la ecuación logística para $r = 3.82$.

5.1.6 Compositor por desarrollo motívico

Los fragmentos que genera el compositor por desarrollo motívico tienen algo más de coherencia por ser todo variaciones sobre un motivo musical. En el fragmento generado 5.15 puede verse destacadas las diferentes transformaciones hechas sobre un motivo introducido manualmente. Es un compositor con un gran potencial aunque requiere de algo más de conocimientos musicales que los anteriores compositores.

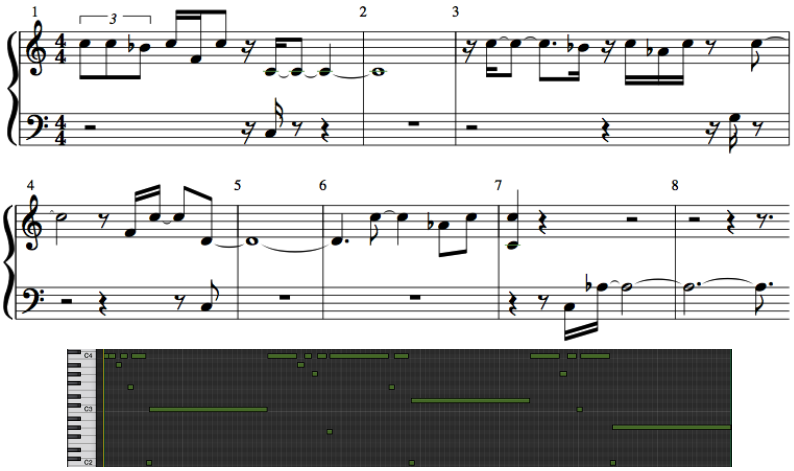


Figura 5.14: Fragmento musical compuesto a partir de la respuesta de la función de Hénon para $a = 1.4b = 0.3$.

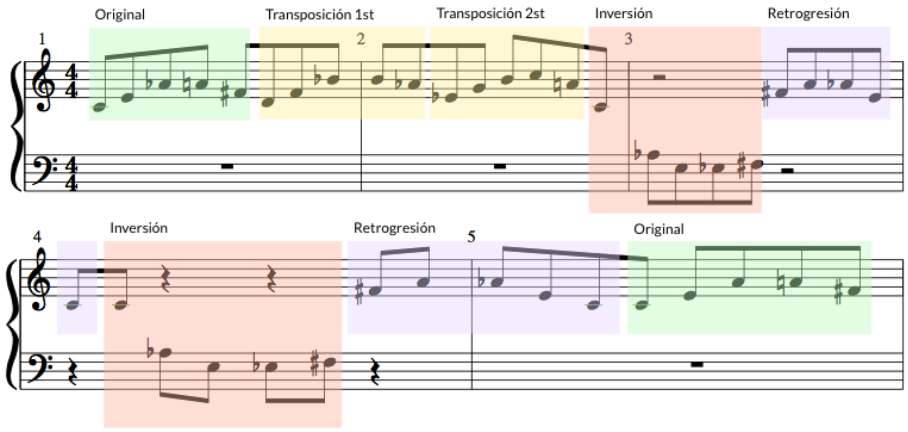


Figura 5.15: Fragmento musical compuesto por desarrollo motívico. Quedan destacadas las transformaciones hechas sobre el motivo original.

5.1.7 Compositor serialista

El compositor serialista crea series a partir de una serie original que decide el usuario. En el ejemplo 5.16 puede verse destacadas algunas de las múltiples series que pueden generarse a partir transformaciones sobre la serie original. Estas series pueden ser utilizadas tal y como salen del compositor algorítmico, o ser usadas como base para una composición serialista posterior.

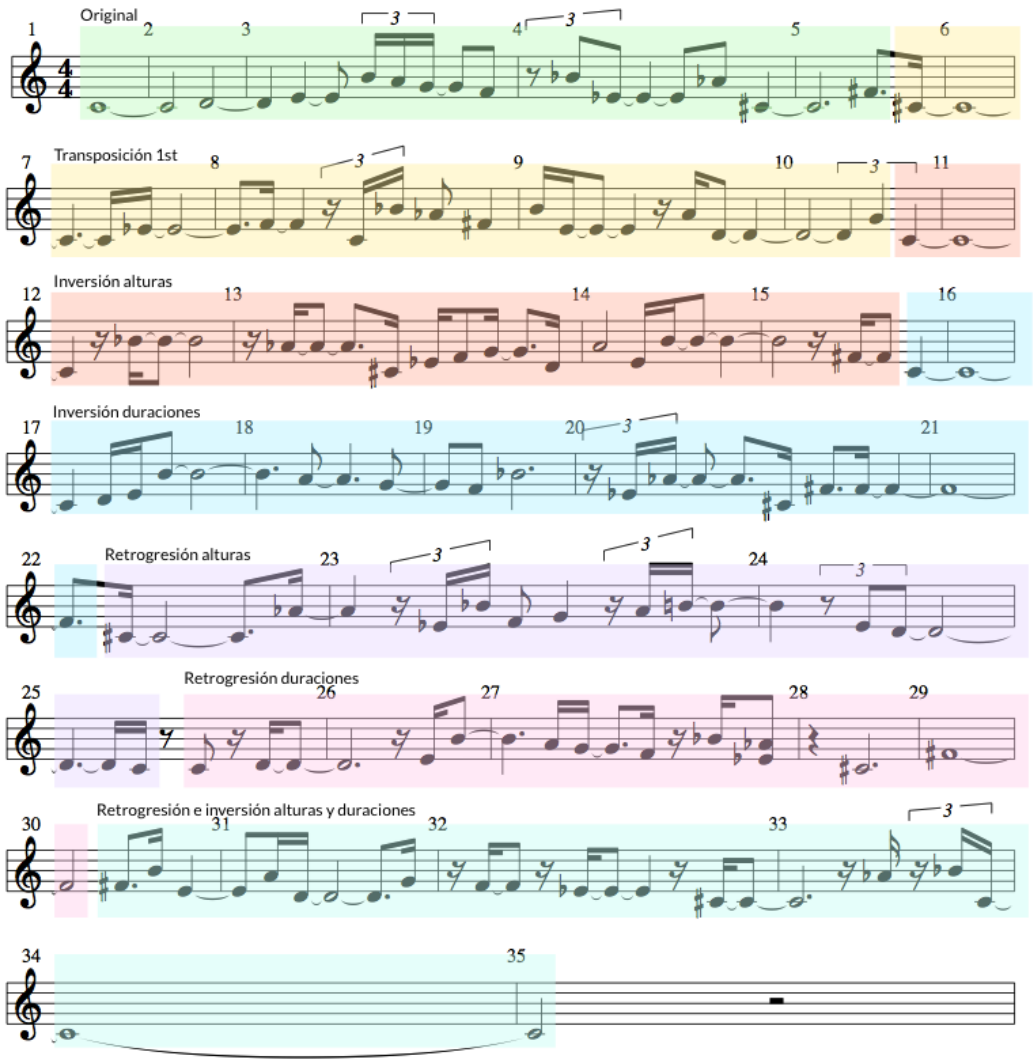


Figura 5.16: Ejemplo de obtención de series mediante el compositor serialista.

6 Conclusiones

6.1 Evaluación del sistema

Para evaluar el resultado del desarrollo de este trabajo nos basaremos en la correcta implementación de los requisitos generales que se enumeraron en el capítulo 3:

- Sistema con propósitos exploratívos y facilidad de uso: a partir de utilizar el conjunto de librerías **openFrameworks** sobre C++ se ha conseguido abstraer las tareas más básicas de implementar en una aplicación (creación de ventanas, visualización de gráficos, generación de audio), de las tareas propias de implementación de este trabajo. Por esto se ha podido ir más allá en cuanto al diseño del sistema y no encontrarse limitado por el uso de librerías más básicas en cuanto a elementos visuales y de audio. Además, el uso de la interfaz gráfica facilita enormemente el uso de la aplicación por casi cualquier usuario. El sistema está preparado para ser utilizado con propósitos exploratívos y educativos.

- Implementación de diferentes métodos de composición algorítmica: se ha conseguido implementar siete métodos diferentes de composición algorítmica de forma básica. Estas implementaciones son suficientes para ser utilizadas con un propósito explorativo de las mismas, sin necesidad de complicar la tarea con el estudio en el método, para aprender las bases del mismo.

- Parametrización de los compositores: el sistema requería de cierto grado de *personalización* de los compositores. Se ha implementado que,

por medio de la interfaz gráfica, el usuario pueda controlar los métodos de composición algorítmica de forma sencilla, y poder ver las diferentes salidas que producen los cambios en los parámetros del compositor.

- Visualización de la salida musical: se ha implementado que el fragmento musical compuesto pueda ser escuchado y visualizado simultáneamente. La escucha existe por medio de un sintetizador por muestreo, y la visualización mediante una partitura dinámica que se muestra a la vez que se escuchan las notas. Con esto se consigue que el compositor sea muy explícito en cuanto a mostrar el resultado, sin necesidad de utilizar un programa externo para ello.

- Posibilidad de exportación de los fragmentos musicales: el sistema contiene un exportador a formato SMF de los fragmentos musicales. Gracias al exportador se pueden obtener las composiciones que se han generado para ser utilizadas por cualquier sistema de edición o reproducción de ficheros MIDI de forma externa al compositor.

Tras ver que la totalidad de los objetivos planteados han sido implementados, puede concluirse el sistema con un resultado satisfactorio.

6.2 Desarrollo del trabajo

El desarrollo del trabajo puede verse como un proceso por iteraciones dividido en fases, correspondientes a las fases clásicas de desarrollo de software (estudio, análisis, diseño, implementación y depuración), en el que cada iteración corresponde a cada uno de los métodos de composición algorítmica que han sido estudiados y posteriormente implementados. Las fases que se han seguido para cada método son las siguientes:

1. Estudio y documentación del método: en esta fase se ha estudiado el método a implementar, consultando fuentes bibliográficas y extrayendo las bases del método.

2. Análisis de los requisitos del método para el sistema: a continuación, tras el estudio teórico del método, se ha analizado los requisitos que necesita implementarse para dicho método, adaptándolos a las necesidades generales y recursos disponibles en el proyecto.

3. Diseño e implementación del método: se ha hecho un diseño conceptual del método de forma sencilla, de forma que la implementación sea lo más fácil posible y sin aparentes cambios sobre la marcha. En primer lugar se hace una implementación sencilla del compositor en línea de comandos, para a continuación pasar a diseñar e implementar la interfaz gráfica que controlará el método.

4. Pruebas y depuración del método: el último paso es el de depurar el sistema a partir de pequeñas pruebas sobre el compositor y comprobaciones de la salida resultante. Se realizan pruebas tanto del compositor como de su interfaz de usuario correspondiente.

6.3 Trabajos futuros

Durante el desarrollo del sistema se han ido sucediendo nuevas ideas sobre el sistema actual que podrían resultar interesantes como futuros trabajos. Estas ideas se enumeran a continuación:

- Añadir nuevos métodos de composición algorítmica: existen aún más métodos de composición algorítmica que los que han sido implementados, como el uso de gramática generativa, algoritmos genéticos o de aprendizaje. Debido a su complejidad no se consideraron implementar en un principio, pero resultaría interesantes como futuros añadidos al sistema.

- Composición algorítmica multipista: actualmente el sistema se ha ideado como un compositor de fragmentos monofónicos. Podría ser interesante implementar un compositor de pistas armónicas y bases rítmicas sobre la melodía que se ha generado, para obtener cierto grado de polifonía sobre las composiciones.

- Más parámetros musicales afectados por los algoritmos de composición: actualmente solo las alturas y las duraciones musicales se ven afectadas por los métodos de composición. Se plantea que en un futuro puedan afectar al resto de parámetros (dinámica, articulación, timbre).

- Utilización simultánea de los métodos de composición: se plantea que puedan utilizarse a la vez varios de los métodos de composición, afectando cada uno de ellos a un parámetro musical.

6.4 Conclusiones y reflexiones finales

Este trabajo se planteó en su presentación como una aplicación que consiguiera aproximar la composición algorítmica a un público que no tuviera necesariamente conocimientos musicales o informáticos. Usuarios que quisieran obtener composiciones musicales que pudieran servirles como fuente de inspiración o que puedan utilizar directamente. Otra de las motivaciones principales era obtener un programa que sirviera de ejemplo de algoritmos de composición a estudiantes de música, informática, o cualquiera que quisiera

acercarse a las técnicas de composición algorítmica, y ayudarles a entender su funcionamiento y bases teóricas.

Con este planteamiento inicial se pensó en idear una aplicación que tras seguir unos pasos simples, el usuario obtuviera sus primeros fragmentos musicales compuestos algorítmicamente, y que tras una más profunda inmersión en el programa, obtener resultados que se acerquen más a lo que él desea.

La implementación de los métodos no fue demasiado complicada gracias a todas las fuentes de información que se pudo consultar. Al tratarse de una implementación básica de cada uno de los métodos, el mayor reto fue el conseguir una aplicación usable en cierto grado con la que poder obtener buenos resultados de composición.

Una de las posibilidades que puede considerarse más productivas de la aplicación es la de poder exportar las composiciones a un marco externo del sistema. Utilizando la exportación a un estándar como SMF se ha conseguido llevar los fragmentos a aplicaciones externas con las que poder seguir trabajando en la composición.

Durante el desarrollo del proyecto se fueron probando los métodos ya implementados obtuviendo fragmentos musicales bastante gratos en base a las expectativas que se tenían al comienzo del proyecto. Dejando aparte las posibles mejoras que se han explicado en la sección anterior, el trabajo realizado puede considerarse de satisfactorio en función de las amplias posibilidades de composición que puede llegar a abarcar.

Por último destacar que la totalidad del trabajo (aplicación compilada, código fuente, memoria, documentación, demostraciones en vídeo) se encuentra a disponibilidad del que desee alojada en el servidor del Departamento de Lenguajes y Sistemas Informáticos en la dirección `http://grfia.dlsi.ua.es/cm/projects`.

Bibliografía

[1] Jim Bumgardner. Kircher's mechanical composer: A software implementation.

[2] John Covach. Twelve-tone theory. In Thomas Christensen, editor, *The Cambridge History of Western Music Theory*, pages 603–627. Cambridge University Press, 2002.

[3] Charles Dodge and Thomas A. Jerse. *Computer Music: Synthesis, Composition and Performance*. Cengage Learning, 1997.

[4] Enric Herrera. *Teoría Musical y Armonía Moderna Vol.1*. Antoni Bosh Editor S.A., 1990.

[5] José Manuel Iñesta. *Apuntes de Sonido y Música por Computador*, 2012.

[6] José Manuel Iñesta. *Apuntes de Técnicas de Diseño Sonoro*, 2013.

[7] John A. Maurer. A brief history of algorithmic composition. 1999.

[8] Eduardo Reck Miranda. *Composing Music with Computers*. Focal Press, 2001.

[9] Gerhard Nierhaus. *Algorithmic Composition: Paradigms of Automated Music Generation*. Springer WienNewYork, 2009.

[10] José Ortiga and José Luis Barceló. *Teoría de la Música, Nivel 4*. Piles, 1993.

[11] H. Owen Reed and Paul O. Harder. *Basic Contrapuntal Techniques*. Belwin, 1964.

[12] Richard F. Voss and John Clarke. '1/f noise' in music: Music from 1/f noise. 1977.