Universidad ORT Uruguay

Facultad de Ingeniería

Duphly: Compositor algorítmico de música con sistema de creación de reglas para la melodía

Entregado como requisito para la obtención del título Master en Ingeniería

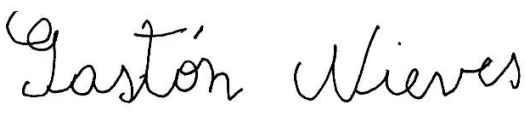
Gastón Nieves – 165379

Tutor: Álvaro Tasistro

2020

Yo, Gastón Nieves, declaro que el trabajo que se presenta en esa obra es de mi propia mano. Puedo asegurar que:

- La obra fue producida en su totalidad mientras realizaba Duphly: Compositor algorítmico de música con sistema de creación de reglas para la melodía;
- Cuando he consultado el trabajo publicado por otros, lo he atribuido con claridad;
- Cuando he citado obras de otros, he indicado las fuentes. Con excepción de estas citas, la obra es enteramente mía;
- En la obra, he acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, he explicado claramente qué fue contribuido por otros, y qué fue contribuido por mí;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.



Gastón Nieves

20/2/2020

A mi fiel compañera durante las largas noches en vela, quien incondicionalmente me brindo su compañía y comprensión durante este difícil proceso.

Agradecimientos

Estoy profundamente agradecido hacia mi tutor Álvaro Tasistro, quien dedicó una gran parte de su tiempo y paciencia durante el desarrollo de esta tesis.

A Yanella Bia, quien incondicionalmente colaboró para que esta tesis fuese posible, muchas gracias.

Agradezco a mi padre, que hizo su mejor esfuerzo por colaborar para que todo salga bien; y a toda mi familia.

Abstract

La presente tesis consiste principalmente del desarrollo de una herramienta capaz de componer música de forma automática o asistida, calificando el sistema resultante dentro de la categoría de composición algorítmica y sistema experto. El sistema provee un entorno que permite la creación de reglas por parte del usuario, las cuales son luego aplicadas para crear de forma automática una línea melódica. El entorno de trabajo planteado fue creado en colaboración con un músico y compositor profesional del medio.

La tesis incluye un análisis de la literatura realizado con el fin de relevar el estado del arte, una implementación completa de una gramática para la creación de bases armónicas de blues de doce compases y el planteo e implementación del entorno mencionado para la creación de reglas de generación de líneas melódicas. La creación de estas reglas se piensa dividida en tres grandes etapas, dos de selección de partes de la obra y una de modificación. Durante la etapa de modificación, se permite el uso de una condición de modificación y una de optimización o garantía. Dado que existen en general múltiples modificaciones válidas y el tamaño de la entrada es variable, se descarta la utilización de algoritmos tipo *backtracking* y se opta por utilizar algoritmos evolutivos, siendo que éstos permiten realizar una búsqueda de soluciones de una forma más eficiente y permiten la existencia de diversidad en los resultados.

Si bien el sistema se aplica particularmente para la composición de blues de doce compases, esto no se presenta como una limitación sino más bien como un caso particular de las capacidades del sistema.

Además de la reproducción durante el tiempo de ejecución, el sistema provee salida en pdf del resultado en una partitura estándar, así como salida mediante archivo midi.

Palabras clave: Composición automática, Composición de blues automática, improvisación de blues automática, composición semi-automática, composición de base armónica de blues asistida, música por computadora, composición algorítmica, sistema experto de composición.

Índice

1. Introducción

La presente tesis consiste de tres grandes piezas fundamentales. La primera de ellas es un análisis de la literatura y estado del arte de la composición automática o semiautomática en sistemas de composición algorítmica. El segundo pilar es la implementación completa en máquina de la gramática para la generación de bases armónicas de blues de doce compases planteada en [7]. Por último, se tiene un entorno que facilita la creación de reglas para ser utilizadas por el sistema en la generación automática de líneas melódicas, facilitando el uso a cualquier usuario. Las dos últimas partes de la tesis continúan el trabajo iniciado en [48] y apuntan entre sus metas a la superación de varias de las limitaciones planteadas en dicho sistema.

El sistema planteado para la creación de reglas permite que éstas sean aplicadas en tres grandes etapas, dos de selección y una de modificación. Durante la etapa de modificación, se permite el uso de una condición de modificación y una optimización o garantía. Siendo que existen múltiples soluciones posibles para el problema y el tamaño de la entrada es variable, se descarta la utilización de algoritmos tipo *backtracking* y se opta por utilizar algoritmos evolutivos, dado que éstos permiten realizar una búsqueda de soluciones de una forma más eficiente, así como la existencia de diversidad en los resultados.

1.1 Sobre Duphly

El sistema sobre el cual se desarrolla esta tesis se define como un sistema experto, es decir, un sistema desarrollado con conocimientos sobre el área, siendo a su vez un sistema de composición algorítmica o basado en reglas. [48] plantea para la creación de la base armónica un subconjunto de la gramática presentada en [7], y para la creación de la línea melódica un conjunto de reglas de composición que se pueden dividir en dos tipos: reglas de creación y reglas de modificación. Todas esas reglas se ven restringidas por dos reglas pasivas: la primera limita la cantidad de octavas y en particular cuáles pueden ser utilizadas por la línea melódica; esto refiere a un uso común por los músicos durante la improvisación e incluso es aconsejado por músicos profesionales. La otra regla pasiva es la creación de una línea de bajo que

recorre la escala de blues sobre los acordes de la base y por debajo de la línea melódica.

Las reglas de creación permiten la generación de una línea melódica armónicamente correcta o de notas reales (y por lo tanto simple) con respecto a la base. Las reglas planteadas en [48] generan salidas tanto aleatorias como con alguna característica o forma especificada. Principalmente el sistema se inclina a la modificación de una melodía, siendo que cuenta con un conjunto de reglas de modificación de una línea melódica que es más amplio que el de las reglas de creación. Las reglas utilizadas son basadas en conceptos musicales extraídos de la experiencia de músicos profesionales.

1.2 Limitantes de Duphly

Como se detalló en la parte anterior, el sistema permite la creación de una base armónica y una línea melódica mediante la utilización de las reglas, facilitando la salida de un archivo *midi* para escuchar el resultado, así como de una partitura que permite la lectura de la música.

Si bien el sistema cumple su cometido, cuenta con un conjunto grande de limitantes, entre ellos: Las reglas son reglas atómicas, es decir, consideran únicamente información sobre la melodía a nivel de notas y las notas siguientes, no permite evaluar y modificar a nivel de frase, compás u obra. A su vez, no es posible la creación de música de duración mayor a doce compases, ya que no cuenta con ninguna forma de extender la melodía y muy probablemente las posibles melodías resultados de aplicar dichas reglas sobre pasajes más largos no cuenten con sentido musical. Entiéndase la música como un lenguaje, y que, como tal, tiene que ser coherente en frases cortas así como en el total del discurso.

Otra limitante es el hecho de que la implementación de la gramática de Steedman [7] se limita únicamente a las reglas que aplican y generan bases armónicas utilizando acordes mayores. Y por último, el sistema permite únicamente la creación de música de blues, estando incluso las reglas limitadas a ese género.

1.3 Contribución

Un primer aporte que realiza esta tesis es una implementación completa de la gramática planteada por Steedman [7], permitiendo al prototipo representar y utilizar escalas tanto mayores como menores y las reglas extras planteadas, que no fueron anteriormente implementados en [48].

Por otra parte, el prototipo permite superar las otras limitaciones de [48] planteadas arriba, puesto que la cantidad de reglas de generación de línea melódica que se pueden crear en el sistema resultante es virtualmente infinita. En efecto, se plantea un sistema que permite la *creación* de las reglas, siendo que dicho sistema debe garantizar al menos la capacidad de crear las reglas ya planteadas en [48]. No se presentan límites en cuanto al estilo musical, siendo que el sistema se adaptó para facilitar el cambio a otros estilos, abriendo la posibilidad del uso de cualquier sistema de referencia en lugar de únicamente la escala de blues.

Para crear las reglas se plantea un sistema con tres etapas de búsqueda y refinamiento, de modificación y de optimización de un determinado parámetro.

La primera etapa consiste en la búsqueda de una determinada característica a nivel de compás; en la segunda se buscan dentro de los resultados obtenidos pasajes musicales que cuenten con alguna particularidad; por último, dentro de esos compases se selecciona un defecto, y para dicho defecto qué modificación particular se desea hacer, así como una postcondición una vez hecha la modificación.

El planteo realizado en esta tesis provee a los compositores y músicos, así como a cualquier usuario en general, la posibilidad de conceptualmente poder probar una noción musical que se haya adquirido mediante la experiencia o inducido mediante la audición, permitiendo formalizar y reconocer la calidad de dicha noción. De esta forma, el usuario se acerca de una forma más directa a la composición algorítmica, hecho que se ha dado en la práctica y de manera informal históricamente en la música.

1.4 Estructura del trabajo

Esta tesis se desarrolla en cinco etapas principales:

La primera parte de este trabajo consta de un análisis del estado del arte y trabajos relacionados con la presente tesis.

La segunda etapa consiste en una breve introducción a conceptos musicales básicos para facilitar al lector la comprensión de los planteos musicales.

En la siguiente etapa se explica la gramática planteada por Steedman [7] y se discute brevemente su implementación.

La cuarta etapa plantea el sistema teórico a implementarse con el fin de facilitar la creación de reglas para la creación de líneas melódicas, y describe el sistema implementado, decisiones de diseño y algoritmos utilizados.

Por último, se contará con una sección dedicada a las conclusiones.

2. Análisis del estado del arte y trabajos relacionados

Con el fin de captar la mayor cantidad de trabajos posibles relacionados con el área de trabajo fue precisa la realización de una revisión sistemática de la literatura.

2.1 Revisión de literatura

2.1.1 Motivación

Esta revisión persigue el fin de proveer un marco de referencia para situar el trabajo de investigación, así como la identificación de distintos huecos en el área de investigación.

2.1.2 Pregunta de investigación

¿Cuál es el estado del arte actual respecto a la composición automática de música con sistemas expertos?

2.1.3 Estrategia de búsqueda

La estrategia de búsqueda se dará en base al uso de palabras clave y cadenas de búsqueda en bases de datos electrónicas y servicios de indexado de publicaciones científicas. A su vez se tomarán en consideración listas de referencias.

Se considerarán únicamente las publicaciones que se puedan encontrar mediante búsqueda en timbo.org.

2.1.4 Criterios de inclusión

Se tomará en consideración únicamente artículos que traten sobre composición automática o semiautomática de música basados en conocimiento experto y composición algorítmica.

2.1.5 Criterios de exclusión

Aquellos artículos que utilicen mecanismos de aprendizaje automático de forma total o parcial para la composición.

Aquellos artículos que refieran a composición de música no armónica.

Artículos que refieran a la composición de música mediante interpretaciones de fuentes de datos diversas, que difieren del conocimiento musical occidental utilizado por los músicos para componer.

2.1.6 Estrategia de extracción de datos

De cada artículo se extraerá: Autor, título, revista o conferencia, año de publicación; método utilizado para la composición; nivel de satisfacción con el resultado del sistema reportado por los autores; y si se realizó una evaluación del resultado por parte de un músico profesional.

2.1.7 Plantilla de la estrategia de extracción de datos

Nombre del artículo	
Autor	
Revista o conferencia	
Año de publicación	
Resumen	
Método utilizado para la composición	
Nivel de satisfacción reportado	
Evaluación del resultado	

2.1.8 Palabras clave a utilizar para la búsqueda

Automatic music composition.

2.1.9 Justificación de los criterios considerados en esta revisión

Como ya fue mencionado, esta revisión tiene la finalidad de establecer un marco teórico y facilitar el conocimiento de huecos existentes en el conocimiento como etapa previa al trabajo de tesis.

La intención de la tesis es el desarrollo de un sistema de composición automático dado por conocimiento experto, de forma que sea posible establecer un conjunto de reglas abstractas que permitan la comprensión exitosa de música independientemente de cualquier sistema informático, siendo éste el principal motivo

por el que se descartan sistemas que utilicen alguna forma de aprendizaje automático, ya que no es transparente el conjunto de reglas utilizado en estos casos.

2.2 Conclusiones sobre la revisión de la literatura

De un total de aproximadamente 1500 artículos resultado de la búsqueda, bajo los criterios de exclusión, fueron analizados 45. Fue posible constatar que la mayoría de los trabajos realizados en la búsqueda de composición automática tienen una tendencia a centrarse sobre el uso de alguna técnica en particular y no tanto sobre la búsqueda desde el punto de vista musical, siendo ejemplo de ello varios trabajos enfocados en la implementación de sistemas evolutivos con diversas variaciones como método de composición. Teniendo esto en cuenta, son de menor ocurrencia los sistemas enfocados en el uso de reglas como medio para la composición y muchos de los métodos utilizados no son replicables por un ser humano, dejando desde el punto de vista del músico muy poca información utilizable.

Fue posible notar durante la revisión que casi ningún sistema permite formular reglas para ser aplicadas, siendo que, en aquellos sistemas que sí utilizan reglas, ellas simplemente fueron desarrolladas e introducidas en el sistema en cuestión, o la conformación de reglas se hace a través de la programación.

Por último, se menciona en algunos artículos la inexistencia de un prototipo que permita probar o utilizar la gramática completa de Steedman [7].

En los anexos de esta tesis se encuentran los cuadros de extracción de datos de algunos artículos de mayor relevancia.

3. Breve introducción a la música

Este capítulo tiene el fin de facilitarle al lector la comprensión de los términos y conceptos que se utilizan a lo largo de este documento. Si bien se explicará parte de la terminología utilizada, no se entrará en mayor detalle y se asumirá que el lector cuenta con algunos conocimientos básicos.

3.1 Notas musicales

Las notas musicales cuentan con un conjunto de características, de las cuales a fines de este trabajo por su importancia introduciremos dos: duración y altura.

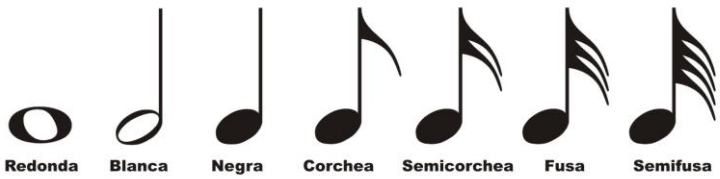
3.2 Duración

En la música la duración se identifica al igual que como en el resto de los ámbitos por un determinado lapso de tiempo. Llamamos figura a aquellos nombres o dibujos que representan una determinada duración relativa de una nota musical con respecto a otra.

Las figuras musicales más comúnmente utilizadas son: redonda, blanca, negra, corchea, semicorchea. Los valores relativos correspondientes suelen calcularse generalmente en base al valor de la negra. Comúnmente, para determinar la duración de cada una de las figuras se le asigna un tiempo a la negra y en base a ese tiempo se calcula el resto de los valores.

Figura	Valor
Redonda	4 negras
Blanca	2 negras
Corchea	½ negra
Semi-Corchea	¼ negra
Fusa	⅛ negra
Semifusa	1/16 negra

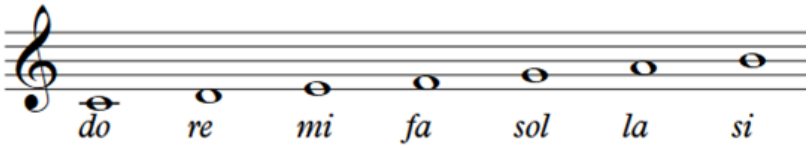
La representación correspondiente a cada figura se da en la siguiente imagen:



3.3 Altura

Las notas musicales además de estar asociadas con una duración en el tiempo, también están asociadas con una frecuencia. Con el fin de determinar la frecuencia se asocia un tono y una octava. Los tonos de las notas musicales son los ya conocidos: Do, Re, Mi, Fa, Sol, La y Si, y las octavas se representan con un número que generalmente va desde el 1 al 8. Es posible mediante una fórmula asociar un tono y una octava a una determinada frecuencia --un caso sumamente conocido es el La de la octava 4, que se suele usar como referencia en las orquestas y para la

afinación, cuya frecuencia es 440Hz. En particular nos va a interesar el conocimiento de las alturas ya que es terminología que será necesaria constantemente.



Además de los tonos existen lo que se llaman semitonos o alteraciones, que son alturas intermedias que se encuentran entre las 7 ya mencionadas. Generalmente se les suelen llamar bemol o sostenido, siendo que bemol refiere a bajar un semitono una nota y sostenido aumentar un semitono una nota; con ambos nombres se puede por lo tanto estar haciendo referencia a una misma frecuencia. En particular existen los siguientes semitonos:

Sostenido	Bemol
Do sostenido	Re bemol
Re sostenido	Mi bemol
Fa sostenido	Sol bemol
Sol sostenido	La bemol
La sostenido	Si bemol

3.4 Distancia entre notas

Se le llama distancia entre notas o intervalo a la diferencia que se encuentra entre la altura de dos notas musicales. Para calcular la distancia se suelen contar los semitonos o una simple cuenta asociando números a la escala. Por ejemplo el intervalo entre la nota do y la nota mi es de 3, asociando 1-do, 2-re, 3-mi; o utilizando semitonos un total de 4 semitonos. La noción de intervalo o distancia entre notas es sumamente importante para la teoría musical, ya que determina la forma de los acordes y escalas musicales.

3.5 Escalas musicales

Una escala musical se define como una secuencia de notas, ascendente o descendente en altura, donde a cada nota se le asigna un grado. Existen muchas formas de escalas; en particular a fines de esta tesis nos interesan las escalas mayores, menores, cromáticas, pentatónicas y la escala de blues.

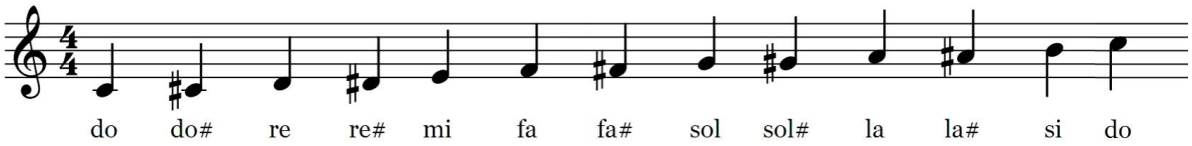
3.6 Grado de una nota dentro de una escala

Dentro de una escala musical a las distintas alturas se les asigna un grado, que determina el intervalo correspondiente entre cada nota y la primer nota de la escala (formalmente la que determina el nombre de la escala musical). La distancia entre los tonos van a estar determinados por la forma de la escala. En el marco de la teoría musical es de uso común referirse a los grados con números romanos.

3.6.1 Escala cromática

La escala cromática se caracteriza por comenzar por una nota cualquiera y la distancia existente entre cada una de sus notas es siempre de medio tono.

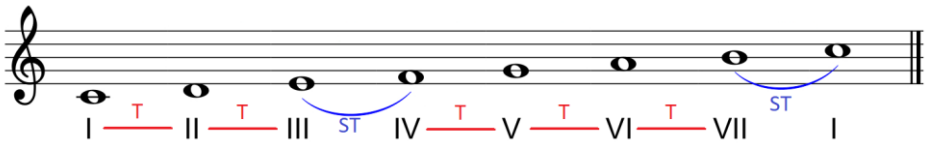
Por ejemplo a continuación podemos ver una escala cromática que comienza en Do:



3.6.2 Escala mayor

Las escalas mayores se caracterizan por tener la siguiente distancia entre las notas: Tono-Tono-Semitono-Tono-Tono-Tono-Semitono, generalmente abreviado de la siguiente forma: T-T-ST-T-T-T-ST.

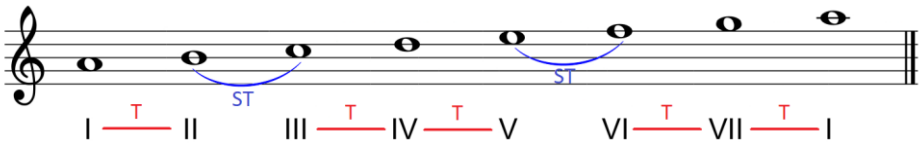
En la siguiente imagen se puede ver la escala de Do mayor:



3.6.3 Escala menor

Las escalas menores tienen la siguiente distancia entre sus notas: T-ST-T-T-ST-T-T.

Por ejemplo, la escala de La menor es como se ve a continuación:



3.6.4 Escala pentatónica

La escala pentatónica es una escala que se suele utilizar tanto en el blues como en el jazz y que cuenta únicamente con cinco notas. Se forma como una subsecuencia de las escalas mayores o menores, por lo que cuentan con forma menor y mayor. En particular la escala de pentatónica mayor cuenta con los grados: I - II - III - V - VI.

La pentatónica menor por su parte cuenta con los grados: I - III - IV - V - VII.

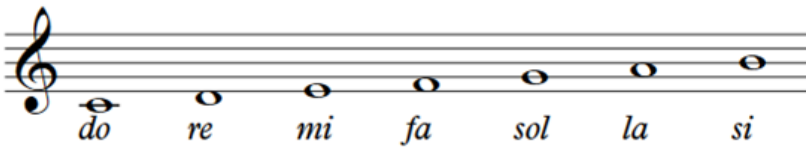
3.6.5 Escala de blues

La escala de blues es una escala hexatónica. En particular la construcción de la escala de blues proviene de la escala pentatónica menor agregando lo que se llama la nota de blues, que se corresponde con el 5o grado disminuyéndolo un semitono. Quedaría entonces la escala de blues como: I - III - IV - bV - V - VII.

3.7 Acorde

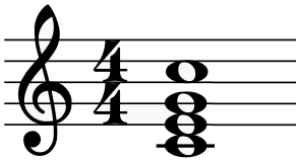
Se le llama acorde a un conjunto de al menos tres notas musicales que suenan en simultáneo y resguardan determinados intervalos entre sus notas, siendo que los intervalos definen el acorde y en base a éstos se nombra. Una forma común de formar acordes es tomando el primer, tercer, quinto y octavo grado de una determinada escala. Por ejemplo un acorde de Do mayor se puede formar tomando el 1er, 3er, 5o y 8vo grado.

Considérese por ejemplo la siguiente imagen:



El Do representa el 1er grado, ya que es la primer nota de la escala. La tercera nota es el mi, la quinta es el sol y por último la octava que no se encuentra presente (al menos no a continuación) sería el Do nuevamente. Por lo que un acorde de Do

mayor se formará con las notas Do-Mi-Sol-Do como se puede visualizar a continuación:



3.8 Progresión armónica

Se le llama progresión armónica o base armónica a una secuencia de acordes, generalmente formados sobre las frecuencias más bajas, que se utilizan para acompañar una melodía u obra.

La base armónica constituye el fundamento para la melodía que se construirá por encima. Es importante tener en cuenta que de la base armónica dependen directamente las notas que podrían considerarse válidas o correctas durante el desarrollo de la melodía.

Las progresiones armónicas se suelen nombrar por grados, considerando la primera nota del acorde (la que le da el nombre al acorde) para calcular el intervalo con respecto al grado principal (llamado tonalidad) de la música. Por ejemplo, si la obra se encuentra en Do Mayor, un acorde de Mi mayor sería un grado 3 o III. Comúnmente se utilizan los números romanos para referirse a los grados de los acordes dentro de una progresión armónica.

Otra denominación común para los acordes dentro de una progresión armónica va ligada directamente a la función que cumplen dentro de la progresión, siendo los nombres asociados de la siguiente forma:

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

3.9 Tipos de acordes

Existen muchos tipos de acordes, siendo la clasificación más significativa la de acordes mayores y acordes menores. Los acordes se identifican principalmente por el nombre de la escala de la cual forman parte, y generalmente se puede saber el nombre del acorde mirando la nota más grave. Reconocer si un acorde es mayor o menor, y la forma o caso particular, es posible contando la cantidad de semitonos que se encuentran entre sus notas, siendo que sin importar la nota en la que comienzan los acordes de un determinado tipo siempre cuentan con la misma cantidad de semitonos entre sus notas.

3.9.1 Acordes mayores

Los acordes mayores se caracterizan por contar con los grados I - III - V - VIII de la escala mayor correspondiente. En el caso que quisiéramos concretar el acorde correspondiente a la escala Do Mayor, nos quedaría el acorde compuesto por las notas DO (primer grado), Mi (tercer grado), Sol (quinto grado) y Do (octavo grado).

3.9.2 Acordes menores

Al igual que en los acordes mayores, los acordes menores se conforman con los grados I - III - V - VII, siendo esta vez los correspondientes a la escala menor. Por ejemplo, el acorde La menor estar compuesto por: La (primer grado) - Do (tercer grado) - Mi (quinto grado) - La (octavo grado)

3.9.3 Variaciones sobre los acordes

Además de contar con las versiones comunes de los acordes existen múltiples posibles variaciones sobre los mismos:

3.9.3.1 Acordes con 7ma

Los acordes con 7ma son aquello que en lugar de incluir el 8vo grado como 4a nota incluyen el 7mo grado. Por ejemplo, un acorde de Do mayor con 7a sería: Do - Mi - Sol - Si

3.9.3.2 Acordes con 9na

A este tipo de acorde se le agrega el 9o grado, además de las 4 notas propias del acorde.

3.9.3.3 Acorde con 13va

Estos acordes se les adiciona el 13vo grado por encima de las 4 notas del acorde.

3.9.3.4 Otras variaciones

Existe una gran cantidad de variaciones más, a fines de esta tesis el resto de las variaciones se irán presentando a medida que sean necesarias.

4. Creación de la base armónica

Como se puede deducir de las partes anteriores del documento, la base armónica no sólo aporta desde el punto de vista musical, sino que también limita las posibles melodías, ya que dependiendo de la base se ve limitada la cantidad de notas posibles que serían “correctas” de usar en la melodía. Es importante comprender que la existencia de notas que no son “correctas” es sumamente común, y en parte es lo que le da variedad o se podría decir que hace interesante una obra en particular.

Con el fin de aumentar la cantidad de salidas posibles del sistema, es de suma importancia mantener una variabilidad en la creación de la base, por lo que se optó por desarrollar un generador a partir de la gramática planteada por Steedman [7].

La gramática se plantea para la creación de bases armónicas de doce compases de cuatro negras por compás.

A continuación se detallarán las reglas que se plantean en [7]. Con el fin de facilitar la lectura se irán exponiendo, a medida que sea necesario, la notación que se utilizará y su significado.

4.1 Regla inicial

La primer regla que se plantea en [7] o regla 0, plantea la estructura básica de la progresión armónica. Escrita utilizando los grados dentro de la progresión queda de la siguiente manera:

Caso base: I - I:7 - IV - I - V:7 - I

Es posible notar la aparición de la notación :7, que refiere a que son acordes con séptima los que se están tratando.

Es importante también considerar que todos los acordes que no contengan ninguna notación que detalle lo contrario son acordes mayores. Por lo que el caso base está compuesto de una progresión de seis acordes, donde todos los acordes son mayores, y en particular existen dos acordes con 7a.

Otro dato interesante y no menor es el hecho de que estamos hablando de progresiones armónicas de doce compases y existen únicamente seis acordes. En este caso entonces cada acorde tiene una duración correspondiente a lo que se llama una cuadrada, la cual es equivalente a la duración de dos redondas u ocho negras, por lo que cada uno de los acordes duraría dos compases (los compases son de cuatro negras de duración)

4.2 Regla inicial menor

[7] propone a su vez la posibilidad de utilizar como caso base una variación sobre la regla anterior para poder introducir las progresiones armónicas que contengan acordes menores. En este caso los grados de los acordes se conservan quedando de la siguiente manera:

Regla 0 menor: Im - I:7 - IVm - Im - V:7 - Im

En este caso se agregó la letra **m** como distintivo de que estamos tratando sobre acordes menores, por lo que ahora la progresión cuenta con una mezcla entre acordes menores y mayores.

4.3 Regla 1

La primera regla de transformación que se presenta en [7] define que es aplicable sobre un acorde cualquiera, ya sea menor, mayor o con séptima, siendo el resultado de su aplicación dos acordes con la mitad de duración que el acorde original, con el mismo grado y mismas alteraciones, con la excepción de que el primer acorde resultado no va a conservar la séptima en caso de tenerla.

Regla 1: x(m)(:7) -> x(m) x(m)(:7)

En la expresión se utiliza una variable x para referirse a un acorde de cualquier grado dentro de la progresión. También ocurren paréntesis, significando éstos que lo contenido es opcional, pero que, en el caso de existir el contenido del paréntesis, tiene que suceder en ambos lados de la expresión.

4.4 Regla 2

La segunda regla especificada en [7] define que, dado un acorde cualquiera, el resultado de aplicar la regla serán dos acordes de la mitad de duración cada uno, donde el primer acorde será igual al original en grado y forma, y el segundo se va a corresponder con el acorde subdominante del acorde original. Esto significa que el segundo acorde será resultado de calcular el 4o grado con respecto al acorde que se le está aplicando la regla.

Regla *2:* x(m)(:7)-> x(m)(:7) SD(x)

En este caso la única variación que existe sobre la notación en la regla anterior es la especificación de **subdominante**, siendo entonces que se debe calcular el acorde del 4o. grado a partir del acorde representado por la variable x.

4.5 Regla 3

Esta regla, a diferencia de las anteriores, cuenta con una precondición: es aplicable a dos acordes seguidos, donde el primero debe ser un acorde mayor sin ninguna modificación (es decir, el primero de los dos acordes no puede ser menor ni ser un acorde con 7ma). A su vez, el segundo acorde debe ser un acorde con 7ma, pudiendo ser menor o no. Con fines de facilitar el entendimiento de esta regla se presentará en con 3 subversiones:

Regla 3a: w x:7 -> D(x):7 x:7

Regla 3b: w x:7 -> D(x)m:7 x:7

Regla 3c: w xm:7 -> D(x):7 xm:7

En este caso la variable w representa un acorde cualquiera. A diferencia de las reglas anteriores el resultado no manipulará la duración de los acordes. Los resultados presentan una nueva especificación, en este caso la letra D, que refiere al

acorde dominante con respecto al acorde x, entendiéndose que se debe calcular el 5o grado a partir del acorde x.

Como podemos ver, las reglas 3a y 3b presentan ante una misma entrada distintas salidas; en el documento [7] define la regla entendiendo como opcional tener una salida con acorde menor o no. A fines prácticos y para facilitar el uso del prototipo, se separó dicha regla en dos, de forma que el usuario pueda elegir si quiere utilizar la salida con el acorde menor. Es importante notar que mediante el uso de esta regla el acorde correspondiente a la variable w desaparece de la progresión, dando lugar a la dominante del acorde x.

4.6 Regla 4

Como en el caso anterior, esta regla tiene como precondición que se deben recibir dos acordes para poder aplicarse, pero en particular el primero tiene que ser el 5o grado o dominante del segundo, y además tener séptima en el acorde. El segundo acorde no tiene restricciones, puede ser mayor, menor y ser acorde con séptima o no.

Regla 4: D(x):7 x(m)(:7) ->bST(x)(m):7 x(m)(:7)

La notación en este caso tiene novedoso el **b** y **ST,** refiriendo **b** a quitar un semitono del resultado de calcular la supertónica (2o grado) de x.

El resultado de aplicar esta regla transforma el primer acorde en la supertónica de x y quitando un semitono, y a este acorde lo transforma en uno con 7ma. El segundo acorde queda intacto, es decir, conserva ser menor o contener séptima, o ambas, según sea el caso.

4.7 Regla 5

La quinta regla de [7] necesita de tres acordes mayores iguales para poder ser aplicada.

Regla 5: x x x -> x ST(x)m M(x)m

La terminología nueva que se integra en esta regla es la **M**, que refiere a la función mediante o el 3er grado con respecto a un acorde dado.

En particular el resultado de esta regla será uno de los acordes intacto, el segundo se le calcula la supertónica y se torna menor, y al tercero se le calcula el mediante y se transforma a menor también.

4.8 Regla 6

Esta regla cuenta también con la precondición de recibir tres acordes. En particular se constituye por tres variantes que se expresarán por separado para facilitar la comprensión:

Regla 6a: x(m) x(m) D(x) -> x(m) #x°:7 D(x)

Regla 6b: x(m) x(m) ST(x) -> x(m) #x°:7 ST(x)m

Regla 6c: x(m) x(m) L(x)m:7 -> x(m) #x°:7 L(x)m:7

En este caso se agregan a la notación dos nuevos elementos: **#** y **°**. El **#** significa que se le debe aumentar un semitono al acorde, por lo que refiere básicamente a un sostenido. Por otra parte, el ° refiere al modo disminuido del acorde, en este caso el acorde se conforma de la siguiente forma: I - III menor - 5a disminuida y 7a disminuida. El término menor al igual que disminuida implica quitar un semitono al grado que se está calculando.

Por otra parte, también se presenta **L**, refiriendo a la función sensible de la progresión que implica calcular el 7o grado con respecto a un acorde.

Analizando las variantes de la regla es posible notar que el único cambio efectivo que sucede es con el acorde del medio, ya que a éste se le aumenta un semitono a la raíz y se le calcula la forma disminuida con séptima; el resto de los acordes quedan como estaban originalmente.

4.9 Reglas opcionales

[7] plantea al final del documento un conjunto extra de reglas que pueden ser aplicadas, pero dando a entender que tienen una menor importancia. Estas reglas fueron implementadas en el prototipo y se explicarán a continuación.

4.9.1 Regla a

La primera regla opcional que se presenta permite a partir de un acorde cualquiera transformarlo en tres posibles casos como se detalla a continuación.

Regla a1: x -> x**:**7**’**

Regla a2: x -> x:9

Regla a3*:* x -> x**:**13

Para esta regla se introducen nuevos elementos a la notación. En el primer caso podemos notar un apóstrofe luego del 7, lo que refiere a que la 7a en lugar de introducirse como la cuarta nota del acorde, se introduce en la octava inferior a la menor nota del acorde, quedando el acorde: VII- I - III - V.

En la segunda regla se utiliza el **9**, siendo que esto es un acorde mayor al cual se le agrega el 9no grado.

El caso de la última regla es análogo al anterior, sólo que se agrega el 13avo grado al acorde.

4.9.2 Regla b

La segunda regla opcional presentada en [7] tiene como precondición la existencia de un acorde con 7ma, y al igual que la regla a cuenta con 3 casos posibles:

Regla b1: x:7 -> xb:9

Regla b2: x:7 -> xb:10

Regla b3: x:7 -> x:7+5

La primera regla **b1** transforma el acorde en un acorde con 9na. Es importante notar que a diferencia de los casos anteriores la **b** se encuentra después de la variable del acorde y no antes, por lo que guarda un significado distinto. En este caso se refiere a la construcción del acorde donde el tercer grado es una tercera menor (implica agregar disminuir un semitono el tercer grado del acorde).

La regla **b2** comparte con la regla anterior el hecho de tener la 3era menor, con la diferencia de que se agrega el 10mo grado al acorde.

Por último la regla b3 introduce **+5**, que tiene como implicancia el agregar a un acorde dado la nota correspondiente con la 5a aumentada, o un semitono más que el cálculo del 5o grado.

4.9.3 Regla c

La tercera regla toma como precondición la existencia de un acorde menor cualquiera y permite las dos siguientes transformaciones:

Regla c1: xm -> xm:7’

Regla c2*:* xm -> xm:6

En este caso la notación de la primer regla ya fue introducida en su totalidad en los casos anteriores, pero se introduce **:6** que implica el agregar una 6a mayor, o lo que es lo mismo el 6o grado del acorde.

4.9.4 Regla d

La última regla opcional planteada en [7] bajo la precondición de contar con un acorde menor con 7ma permite dos salidas.

Regla d1: xm7 -> xm9

Regla d2: xm7 -> xO:7

La regla **d2** introduce a la notación **O (**a nivel musical se suele utilizar el símbolo de conjunto vacío).; en este caso refiere al acorde semidisminuido con 7ma. En particular la construcción de este acorde se hace de la siguiente forma: I - III menor - V disminuida - VII dominante.

4.10 Sobre la gramática de *Steedman*

La gramática que se implementó en el prototipo se hizo siguiendo de la forma más precisa posible el artículo referido, siendo que en algunos casos fue necesario seguir los ejemplos con el fin de discernir algunos casos dudosos; incluso de esta forma existen pequeños casos en los ejemplos que no se apegan perfectamente a la gramática. En particular, a medida que avanzan el documento de *Steedman* [7] y el planteo de las reglas, comienzan a surgir imprecisiones y falta de rigor en las expresiones de la gramática, dando lugar a ambigüedades. Durante este trabajo se hizo un análisis en profundidad para lograr replicar la idea del autor de la forma más adecuada posible.

5. Lenguaje de definición de reglas

En esta etapa del trabajo nos encontramos interesados en definir un lenguaje similar al lenguaje natural con el fin de definir nuevas reglas de composición de líneas melódicas. La finalidad es abstraer las reglas de las ya existentes en el sistema, y sólo alcanzables por un experto en la tecnología, hacia los distintos tipos de usuarios.

5.1 Reglas implementadas en Duphly

En Duphly [48] se implementó un prototipo capaz de improvisar música basado en reglas provistas por un experto en el área. Dichas reglas se aplicaban sobre la línea melódica del sistema con el fin de lograr diferentes melodías. Se procederá a presentar las reglas que fueron implementadas en Duphly [48] y sus diferentes características.

5.1.1 Tipos de las reglas

Las reglas implementadas en Duphly [48] tenían dos fines principales definidos como: “Generar” y “Verificar y Corregir”.

5.1.1.1 Generar

Las reglas de tipo “Generar” tenían la finalidad de crear una línea melódica a partir de una base dada. Dichas reglas consideraban alguna característica particular para la creación de la línea melódica, respetando las características propias del estilo del blues. Este tipo de regla tiene como finalidad principal crear alguna forma de línea melódica base sobre la cual sea posible realizar modificaciones para alcanzar alguna más compleja y de esta forma permitir un cierto grado de variabilidad en la melodía resultante.

5.1.1.2 Verificar y corregir

Las reglas de verificar y corregir fueron creadas con el fin de modificar una línea melódica buscando que se cumpla alguna característica particular y tomando en cuenta el entorno y base armónica existente en la línea mencionada.

5.1.1.3 Reglas generales

Durante la implementación del prototipo se consideraron reglas generales esenciales para garantizar que la forma de la música se apegue al estilo del blues.

5.1.2 Reglas existentes

Como se mencionó en la parte anterior, existen tres tipos de reglas, que se presentarán separadas por el tipo correspondiente.

5.1.2.1 Reglas generales

Limitar octavas utilizadas: Regla que tiene como finalidad garantizar que la melodía se encuentre por encima de la base armónica y a su vez no supere las frecuencias comúnmente utilizadas en una improvisación.

Utilización de notas dentro de la escala de blues: Esta regla se encarga de garantizar que todas las notas utilizadas se encuentren dentro de la escala de blues.

5.1.2.2 Reglas de tipo generar

Creación aleatoria: Teniendo en cuenta las reglas generales se crea de forma aleatoria una línea melódica.

Limitar saltos por compás: La regla se encarga de garantizar que exista únicamente una cantidad predefinida de notas que puedan superar la distancia de una nota en correspondiente escala utilizada para la composición (en el caso del blues la escala pentatónica).

Escala de blues sobre acordes: Regla encargada de generar la escala de blues correspondiente sobre la base (escala pentatónica). Esta escala se agrega como una línea de bajo.

5.1.3 Reglas de tipo Verificar y Corregir (modificación)

Continuidad en el fraseo: Regla con la finalidad de garantizar que la dirección del fraseo (ascendente o descendente) se mantenga de esta forma por una cantidad determinada de notas.

Limitar difusión en la dirección: Regla que itera nota a nota la melodía existente y, en el caso que la nota siguiente a la que se está evaluando se encuentre a más de una nota en la escala correspondiente de la nota actual, entonces la acerca una nota.

Continuidad en el fraseo sin permitir repetidas: Regla de funcionamiento similar a continuidad en el fraseo, sólo que no genera notas repetidas.

Continuidad en el fraseo permitiendo repetidas: Garantiza la dirección en el fraseo sin contar como notas propias de la dirección las repetidas, teniendo una tendencia a generar una mayor cantidad de notas repetidas acumuladas.

Disminución de la utilización de la nota de blues sustituyendo por la nota anterior: Se garantiza que la música contiene hasta un porcentaje dado de aparición de la nota de blues. Si se supera dicho límite, la nota se sustituye por la nota anterior creando una repetición.

Disminución de la utilización de la nota de blues sustituyendo por nota aleatoria: Mismo funcionamiento que la regla anterior, con la diferencia que la nota que se utiliza para la sustitución es una nota aleatoria dentro de la escala de blues.

Continuidad en el fraseo con variante: Regla que cumple con el mismo funcionamiento que la regla “continuidad en el fraseo” pero variando la forma de sustituir las notas, generando una cierta flexibilidad en el resultado final.

Aumentar la repetición de notas: Regla que se enfoca en el aumento de la cantidad de notas repetidas seleccionando qué notas cambiar garantizando la continuidad en el fraseo.

5.2 Forma de aplicación de las reglas

Las reglas que se utilizaron en Duphly [48] se aplican bajo el patrón Cadena de Comandos, donde la primera regla que se aplica siempre es una regla generar y luego reglas de verificar y corregir. Cada regla recibe el resultado devuelto por la regla que se aplicó anteriormente y a ese resultado le aplica su propia regla, luego devuelve el resultado y es entregado a la regla siguiente. No es de menor relevancia considerar que las reglas no garantizan que la regla que se aplicó anteriormente se siga cumpliendo en el caso que se apliquen reglas posteriores, por lo que es importante considerar que el orden de las reglas tiene influencia sobre el resultado.

Base Armónica ->Generadora -> Verificar y corregir 1 ->...-> Verificar y corregir N -> Melodia

5.2.1 Limitaciones de las reglas existentes.

A simple vista, las reglas utilizadas en Duphly [48] sufren la limitante del rango de expresión que se utiliza para garantizar el resultado, faltando reglas con la capacidad de permitir que se cumplan múltiples reglas en simultáneo (la intersección entre reglas puede ser el conjunto vacío) además de la falta de reglas a niveles mayores que niveles atómicos. En efecto, las reglas actuales actúan sobre una nota y las notas siguientes cercanas y no garantizan nada a nivel de frase o nivel de composición, por lo que las composiciones tienen sentido en pequeñas partes. A su vez no existen reglas creadas con respecto a la métrica.

5.4 Proceso para la definición de un lenguaje de creación de reglas de generación de líneas melódicas

Durante el desarrollo de este modelo, además de abstraer directamente del trabajo realizado en [48], se contó con la colaboración de un compositor profesional del medio.

En un principio se van a analizar las reglas existentes con el propósito de abstraer las características generales del proceso. La primera meta será lograr que el lenguaje creado tenga la capacidad de representar las reglas anteriormente utilizadas. La segunda meta será lograr que el lenguaje tenga la capacidad de superar las limitaciones propias de las reglas anteriormente mencionadas. Por último, las reglas creadas deberán tener la capacidad de expresión suficiente como para poder implementar las reglas planteadas por el compositor profesional.

5.4.1 Perspectivas del lenguaje

Para construir el lenguaje de la forma más completa posible se analizaron las necesidades partiendo desde la modificación de una nota hasta la consideración de la melodía completa, así como de manera inversa. Al primer proceso lo vamos a llamar *bottom-up* y al segundo *top-down*, para luego definir el lenguaje buscando un punto común entre ambas perspectivas.

5.5 Análisis *bottom-up*

Parece claro en un principio que las reglas existentes cuentan con tres formas básicas, reglas generales, reglas de creación y reglas de modificación, que en términos generales sería útil conservar para analizar el lenguaje de creación de reglas.

5.5.1 Reglas generales

Por la utilización explicada en Duphly [48] no es menor notar que las reglas generales deberán ser aplicadas a todas las reglas y por lo tanto parecería importante contar con un lenguaje particular diferente al utilizado en el resto de las reglas.

5.5.2 Reglas de generación

Las reglas de tipo *generar* deberán generar una línea melódica a partir de una base, siendo que este tipo de reglas tendrán la necesidad de tener un nivel de especificación más exacto, de forma que deberán ser más precisas que las reglas de modificación y el lenguaje debería permitirlo.

5.5.3 Reglas de modificación

A la hora de realizar una modificación, con el fin de poder identificar a gran escala y permitiendo superar la limitación del trato a nivel atómico de la melodía, es necesario que el sistema permita seleccionar del total de la música aquellas partes que cumplan con una determinada condición, que se podría identificar como macro condición. Una vez realizada la macro selección, el lenguaje deberá permitir la selección de pasajes musicales que cumplan con alguna condición, de forma que nos acerquemos al defecto que se desea cambiar. Por último, una vez dentro del pasaje, el lenguaje deberá permitir la selección y modificación del defecto, buscando bajo una condición, realizando la modificación deseada y optimizando algún parámetro, que no tiene por qué ser el mismo que implicó la modificación.

5.5.4 Características del lenguaje

En un principio podemos identificar la necesidad de sintaxis para tres etapas principales acordes a los tipos de las reglas:

1. Reglas generales que deberá cumplir toda la composición. Ninguna regla debe violar alguna regla general. Por lo tanto estas reglas deberán heredar su comportamiento a todas las reglas utilizadas en una composición.

2. Reglas de creación que deberán contar con un lenguaje suficientemente específico para determinar algunas características durante la creación. Es importante que el resultante de aplicar una regla de creación deberá tener una gran flexibilidad de forma que se permitan múltiples posibles creaciones para aumentar el abanico de resultados posibles.
3. Reglas de modificación, para las cuales se pueden inferir tres etapas. La etapa de búsqueda, de selección y por último de modificación.
4. Es importante notar que es imprescindible contar con elementos para la búsqueda que permitan contar o evaluar porcentajes en un determinado tramo, así como la selección de los elementos que se desean modificar para lograr el resultado buscado.
5. Es vital para el análisis de los resultados la posibilidad de relevar datos sobre una composición, mediante diferentes métricas que se analizarán más adelante.

Otro punto a determinar serán los distintos niveles de aplicación de las reglas para el posterior análisis, planteándose las posibles acciones en distintos niveles.

5.5.5 Orden de interés en el trabajo.

Para esta tesis es de principal importancia lograr un lenguaje que permita implementar principalmente los puntos 3 y 4 de la parte anterior, entendiendo estas partes como las más importante del lenguaje, puesto que ellas tienen el potencial de lograr cualquier resultado posible independientemente del resto de las reglas aplicadas.

5.5.6 Aplicación de las reglas

En un principio el lenguaje será creado con el fin de aplicar las reglas en un orden determinado por el usuario, y bajo el mismo funcionamiento que se usaba en Duphly [48].

5.5.7 Resumen de Funciones Consideradas.

Considerando la extensión y complejidad que implica el análisis bottom-up se plantea a continuación un resumen de las conclusiones resueltas durante el mismo. Sin embargo, en los anexos de este documento se encuentra el análisis completo bottom-up, siendo que si el lector lo desea puede referirse al mismo.

Con el fin de facilitar la escritura de las expresiones se utiliza la sintaxis de tipado de Haskell.

Durante el análisis se concluyó la necesidad de contar con dos etapas de búsqueda, siendo que la primera realiza una búsqueda en toda la melodía y la segunda una búsqueda a nivel de pasajes. Por último, se plantean modificaciones, donde las modificaciones requieren un criterio para ser aplicadas previo a realizar una sustitución. Las sustituciones se resumen en dos: modificar la altura de una nota, siendo que se modifica tanto el tono como la octava, y la facilidad de modificar la duración de una nota.

Se dejan planteadas utilizando un tipado de la forma de Haskell las funciones resultado del análisis, donde sólo se explicitan los parámetros que reciben

1. BusquedaAplicacion :: Melodía-> Base -> Criterio -> Largo -> CriterioLargo

Esta función tiene la finalidad de buscar en la melodía bajo un criterio pasajes de un determinado largo. Por ejemplo, podría retornar todos los subconjuntos de dos compases de una melodía, seleccionados bajo un criterio.

2. Criterio :: Condición -> FCondicionante

Esta función representa el posible criterio de selección, donde luego de utilizada la condición, se le aplica el resultado a la función condicionante y retorna un booleano en caso que se cumpla.

3. BusquedaNotas :: Melodía-> Base -> Criterio

Esta operación se trata de una búsqueda de refinamiento, donde se utiliza nuevamente un criterio de búsqueda que permita encontrar partes de la melodía de

menores dimensiones, siendo la finalidad de la función una segunda etapa de refinamiento luego de **BusquedaAplicacion**.

4. ModificarNota:: Melodía-> Base->Nota->Criterio -> Sustitucion

Una vez aplicados los 2 pasos anteriores llega el momento de realizar la modificación correspondiente, para esto se determina sobre qué nota y bajo qué criterio se realiza la modificación

5. Sustitucion:: Melodia -> Criterio-> Nota

Sustitución es la función que queda pendiente en modificarNota, siendo la que se va a encargar de realizar efectivamente el cambio.

6. ModificarAlturaNota:: Altura -> Altura

Función atómica para modificar el tono y octava de una nota

7. ModificarDuracionNota:: Duracion->Duracion

Función atómica para modificar la duración de una nota.

5.6 Análisis Top-Down

Para realizar el análisis desde el punto de vista inverso al anterior, se partirá desde lo que se busca lograr y se irá especificando hasta llegar a la idea concreta.

En un principio se formalizan las reglas planteadas en [48] y luego con el fin de enriquecer el lenguaje se formalizarán reglas planteadas por una compositora profesional del medio.

Como se mencionó en la parte anterior, el tipo de reglas que resulta más importante a fin de esta tesis son aquellas que permiten la modificación de una melodía, por lo que consideraremos aquellas reglas de [48] de tipo “verificar y corregir”. Con el fin de facilitar la lectura se listarán nuevamente las reglas correspondientes:

1) Continuidad en el fraseo.
2) Limitar difusión en la dirección.
3) Continuidad en el fraseo sin permitir repetidas.
4) Continuidad en el fraseo permitiendo repetidas.
5) Disminución de la utilización de la nota de blues sustituyendo por la nota anterior
6) Disminución de la utilización de la nota de blues sustituyendo por nota aleatoria.
7) Continuidad en el fraseo con variante
8) Aumentar la repetición de notas

Se agregan a la lista las reglas propuestas por la compositora:

9) Buscar saltos ascendentes o descendentes y agregar escalas cromáticas o escalas diatónicas desde una nota a la otra en un 10 por ciento de los casos cada dos compases.
10) Dividir notas en posiciones aleatorias y subir un semitono o un tono la primer o la segunda nota un 10 por ciento de los casos.

11) En el caso de encontrar dos notas iguales corridas, dividir la primera en dos y aumentar la segunda mitad de la nota en un semitono o un tono.
12) Si se encuentra un intervalo ascendente dividir la segunda nota del intervalo y aumentar un tono la primer nota de la división, sobre los compases más al medio de la melodía.
13) Si se encuentra un intervalo descendente dividir la segunda nota del intervalo y disminuir un tono la primer nota de la división, sobre los compases más al medio de la melodía.
14) Aumentar una octava la melodía en algún compás aleatorio.
15) Disminuir una octava la melodía en algún compás aleatorio.
16) Dividir algunas notas en dos y aumentar una octava la primera de la división
17) Dividir notas y la segunda nota de la división hacer un salto de 7a, 9a o 13ava.
18) Sustituir un porcentaje de notas por silencios.
19) Transponer melodía en un compás dado n semitonos hacia arriba o abajo.

5.6.1 Análisis de cada regla

Con el fin de extraer las abstracciones necesarias para el lenguaje, se analizará cada una de las reglas planteadas y se determinarán las funciones necesarias para poder implementar dichas reglas en un lenguaje.

5.6.1.1 Continuidad en el fraseo

Recordando brevemente, la regla garantiza que cada grupo de tres notas conserve la dirección ascendente o descendente. Por lo que para que esta regla sea posible el lenguaje debe permitir seleccionar todos los pasajes donde haya grupos de menos de tres notas que no conserven la dirección, así como modificar dicho grupo para que mantenga alguna dirección o al menos repita la nota.

5.6.1.2 Limitar difusión en la dirección

Para que esta regla sea posible es necesario poder identificar distancia entre notas, siendo que aquellas notas que excedan un tono de distancia deberán bajarse o subirse a la siguiente nota válida o real.

5.6.1.3 Continuidad en el fraseo sin permitir repetidas

Esta regla al similar a continuidad en el fraseo, necesita únicamente el operador mayor en lugar del operador mayor o igual, como es el caso de arriba.

5.6.1.4 Continuidad en el fraseo permitiendo repetidas

Para este caso la cantidad de notas que se van a tomar es indefinido, por lo que se necesitan dos etapas de filtrado. La primera para determinar un conjunto de notas repetidas a las cuales agregarles una nota de principio y fin, mientras que en la segunda búsqueda se deberá identificar y modificar aquellos casos donde no se cumpla que se conserve la dirección. Por último, aplicar la regla **continuidad en el fraseo permitiendo repetidas**.

5.6.1.5 Disminución de la utilización de la nota de blues sustituyendo por la nota anterior

Para esta regla es necesario poder identificar la nota de blues y porcentaje de apariciones, y por último modificar cada aparición por la nota anterior.

5.6.1.6 Disminución de la utilización de la nota de blues sustituyendo por nota aleatoria

Caso equivalente al de arriba, con la diferencia que debe permitir sustituir la nota por una nota aleatoria válida.

5.6.1.7 Continuidad en el fraseo con variante

Este caso es similar al caso de continuidad en el fraseo, sólo que se agrega un factor de aleatoriedad en la aplicación de la regla.

5.6.1.8 Aumentar la repetición de notas

El lenguaje deberá permitir la selección aleatoria de porciones de la melodía y modificarlos por la nota anterior o la nota siguiente.

5.6.1.9 Buscar saltos ascendentes o descendentes y agregar escalas cromáticas o escalas diatónicas desde una nota a la otra en un 10 por ciento de los casos cada dos compases

Para la definición de esta regla es necesario poder identificar intervalos, subdividir la duración de una nota, y permitir el ingreso de la forma de escala que se desea utilizar. Si bien parecería un concepto simple, se debe tener en cuenta que la subdivisión de la nota depende de la escala ingresada y la cantidad de notas; a su vez, la escala debe ingresarse de forma que sea independiente de la base. Por último, hay que considerar que se debe poder seleccionar los compases y asignar un porcentaje de casos de acción.

5.6.1.10 Dividir notas en posiciones aleatorias y subir un semitono o un tono la primera o la segunda nota un 10 por ciento de los casos.

Sobre lo ya incluido en la regla anterior, para este caso se agrega la manipulación directa de las notas de forma absoluta (independiente de la base), así como la posibilidad de dividir una nota en dos.

5.6.1.11 En el caso de encontrar dos notas iguales seguidas, dividir la primera en dos y aumentar la segunda mitad de la nota en un semitono o un tono.

Para lograr esta regla se puede utilizar un concepto ya incluido anteriormente en las reglas: medir la diferencia entre dos notas; en este caso si son iguales, al igual que en la anterior, dividir la nota en dos y aumentar una de las dos notas un semitono.

5.6.1.12 Si se encuentra un intervalo ascendente dividir la segunda nota del intervalo y aumentar un tono la primer nota de la división, sobre los compases sobre la mitad de la melodía

Esta regla introduce dos nuevos conceptos al lenguaje, el concepto de **intervalo ascendente**, el cual se puede resolver calculando la diferencia entre dos notas, y además, el concepto de posición relativa dentro de la música, cuando refiere a la mitad de la melodía.

5.6.1.13 Si se encuentra un intervalo descendente dividir la segunda nota del intervalo y disminuir un tono la primer nota de la división, sobre los compases más al medio de la melodía

Este caso queda totalmente cubierto con los conceptos del caso anterior.

5.6.1.14 Aumentar una octava la melodía en algún compás aleatorio

El concepto de aumentar una nota una cantidad determinada de posiciones ya se introdujo en reglas anteriores, sin embargo, el concepto de una selección aleatoria es diferente a lo ya visto.

5.6.1.15 Disminuir una octava la melodía en algún compás aleatorio.

Esta regla se puede considerar incluida dentro de los casos de la anterior.

5.6.1.16 Dividir algunas notas en 2 y aumentar una octava la primera de la división

Ambos conceptos de esta regla están contemplados en reglas anteriores.

5.6.1.17 Dividir notas en posiciones aleatorias y la segunda nota de la división hacer un salto de 7a, 9a o 13ava.

A pesar de parecer diferente a las reglas anteriores, no se introduce ningún concepto nuevo, ya que dividir notas o realizar saltos (aumentar o disminuir semitonos) ya fueron contemplados en reglas anteriores

5.6.1.18 Sustituir un porcentaje de notas por silencios.

Si bien los silencios son un concepto sumamente importante para la música, ni a lo largo de esta tesis ni en [48] fue contemplado en ningún momento. Lo cual hace sumamente importante integrarlo al conjunto de reglas.

5.6.1.19 Transponer melodía en un compás dado n semitonos hacia arriba o abajo

Concepto contemplado anteriormente bajo el concepto de mover la altura de una nota n semitonos hacia arriba o abajo.

5.6.2 Extracción de los conceptos analizados

A continuación se van a listar todos los conceptos que formaron parte del análisis de las reglas de [48] y las reglas propuestas por la compositora.

1) Filtrado por dirección ascendente/descendente por una cantidad de n notas
2) Modificación de la altura de las notas considerando la nota anterior
3) Filtrado por distancia entre notas
4) Modificación de altura de una nota una cantidad dada de semitonos.
5) Filtrado por dirección ascendente/descendente por una cantidad n de notas aceptando repetidos (<=, >=)
6) Búsqueda de notas por notas repetidas
7) Agregar notas al principio y fin de una selección
8) Múltiples etapas de búsqueda
9) Identificar un determinado grado con respecto a la base
10) Conteo de una determinada propiedad
11) Sustituir una nota por la anterior
12) Sustituir una nota por una nota aleatoria válida
13) Asignar una probabilidad de aplicación de la regla
14) Selección de la melodía por tramos
15) Búsqueda por intervalos
16) Dividir la duración de una nota
17) Asignar una secuencia relativa de notas con respecto a la base
18) Asignar una secuencia absoluta de notas
19) Asignar una secuencia de notas relativa con respecto a otra
20) Selección basada en compases
21) Sustituir una nota por un valor absoluto
22) Búsqueda en la melodía por posición relativa
23) Selección de compases de forma aleatoria.
24) Sustituir notas por silencios

Con fin de organizar la información extraída y analizarla se separan los conceptos en asignación y selección.

5.6.2.1.1 Asignación:

1) Modificación de la altura de las notas considerando la nota anterior
2) Modificación de altura de una nota una cantidad dada de semitonos.
3) Sustituir una nota por la anterior
4) Sustituir una nota por una nota aleatoria válida
5) Dividir la duración de una nota
6) Asignar una secuencia relativa de notas con respecto a la base
7) Asignar una secuencia absoluta de notas
8) Asignar una secuencia de notas relativa con respecto a otra
9) Sustituir una nota por un valor absoluto
10) Sustituir notas por silencios
11) Asignar una probabilidad de aplicación de la regla

Este conjunto de reglas se puede simplificar ideando conceptos un poco más generales y facilitando el juego de reglas.

La primera regla que se puede considerar es: Sustituir una nota cambiando n semitonos con respecto a la anterior o siendo mayor, menor, siguiente o anterior válida con respecto a la anterior. Esta regla podría cubrir conceptualmente las reglas: 1 y 3.

La segunda regla sería sustituir una nota por un valor aleatorio, mayor o menor válido, o aumentar/disminuir n semitonos una nota o una determinada nota o silencio. Esta regla presenta la capacidad de representar las reglas 2,4, 9, 10.

La siguiente regla podría plantearse: sustituir una nota por una secuencia dada de notas, o un conjunto de grados con respecto a una nota o una secuencia relativa con respecto a la base. Esta regla agrupará las reglas: 6,7,8.

Por último dividir nota en x partes.

Para expresar las reglas se utilizará el símbolo “**|**” con el fin de explicitar una opción o la otra de forma excluyente y **[ ]** para expresar un conjunto. Se expresan las reglas a continuación:

1) Sustituir nota con respecto a la anterior ((aumentando | disminuyendo) n semitonos) | ((mayor | menor | siguiente mayor | siguiente menor) válida)
2) Sustituir nota (por valor (mayor|menor|aleatorio) válido) | ((aumentar|disminuir) n semitonos) | (por nota|silencio)
3) Sustituir notas por secuencia (**absoluta** [**notas**]) | ((respecto a (**nota**| **base**)) [**grados**])
4) Dividir una nota en **n** partes

Por último, a todas las reglas se les puede agregar un parámetro que determine la probabilidad de que se apliquen las reglas.

5.6.2.2 Búsqueda y selección:

1) Filtrado por dirección ascendente/descendente por una cantidad de n notas
2) Filtrado por distancia entre notas
3) Filtrado por dirección ascendente/descendente por una cantidad n de notas aceptando repetidos (<= , >=)
4) Búsqueda de notas por notas repetidas
5) Agregar notas al principio y fin de una selección
6) Múltiples etapas de búsqueda
7) Identificar un determinado grado con respecto a la base
8) Conteo de una determinada propiedad
9) Selección de la melodía por tramos
10) Búsqueda por intervalos
11) Selección basada en compases
12) Búsqueda en la melodía por posición relativa
13) Selección de compases de forma aleatoria.

Uno de los puntos interesantes a notar es la necesidad de contar con más de una etapa de búsqueda, siendo que cada búsqueda puede arrojar una selección diferente bajo criterios diferentes. Por lo que en un principio consideraremos la posibilidad de tener la búsqueda en una etapa y la selección para la modificación en otra.

Durante el análisis de las reglas se puede notar la existencia de más de una etapa durante la búsqueda. En una primera etapa se determina el criterio por el cual se va a seleccionar, y en una segunda etapa se agregan elementos al resultado.

Las reglas 1 y 3 pueden condensarse en una única regla que permita ambas selecciones.

Identificar un grado con respecto a la base o una diferencia con respecto a la nota siguiente se definen como la segunda regla, considerando las reglas 2, 7,10.

El lenguaje deberá contar con una regla para tratar con los compases y por lo tanto abarcar las reglas 11 y 13.

A su vez, el lenguaje deberá permitir el conteo de métricas o porcentajes de aparición de una determinada nota, para cumplir con la regla 8.

El resto de las reglas serán consideradas como un agregado opcional para cada una de las anteriores.

Al igual que en la parte anterior, se resumieron las reglas de forma general y compacta:

1) Filtrado por dirección (**ascendente**|**descendente | repetidas**) (contando repetidas (**si**|**no**))
2) Filtro por ((intervalo|grado) (diferencia (mayor|menor| igual) numero)))
3) Selección de compases (**aleatoria |** (posicion (**medio|fin|principio**)) (cantidad de compases))
4) Aparición de (Nota | altura) (mayor|igual|menor) (Cantidad | Porcentaje)

Por último, a cada una de las reglas se le debe agregar la posibilidad de agregar notas al principio y fin de la selección, así como determinar la posición sobre la cual se quiere filtrar (principio, medio o fin).

5.7 Unión de ambas perspectivas

Habiendo hecho un análisis desde ambas perspectivas del lenguaje es tiempo de cotejar ambos resultados para buscar una opción óptima. En ambos casos se destacó la necesidad de contar con múltiples etapas de búsqueda previo a la modificación; en particular contar con dos etapas de selección generales, una etapa específica y una etapa de modificación parecería ser la solución óptima. Con el fin de poder seleccionar un resultado específico, y contemplando la totalidad de la música, las etapas se dividirán en una búsqueda más amplia a nivel de compás, una búsqueda a nivel de pasaje y por último una búsqueda al detalle para seleccionar lo que se desea modificar en particular. Las reglas contarán con todos estos niveles --- sin embargo, no necesariamente todas las reglas harán uso de todos ellos: al momento de crear la regla; dependiendo de la complejidad de la misma, el usuario deberá determinar qué niveles desea utilizar.

Las reglas entonces quedarán determinadas por la composición de los distintos niveles de planteados, siendo que el resultado de cada parte se pasará a la siguiente.

Con el fin de enriquecer el lenguaje, aumentar el nivel de precisión del mismo, así como la variabilidad de los resultados, se plantea el uso de métricas.

5.7.1 Probabilidad

Es de interés en particular que una regla no siempre sea aplicada, por lo que al momento de crear la regla se determina la probabilidad de que se aplique sobre un pasaje.

5.7.2 Porcentajes

Al momento de la creación de una regla es útil una métrica con respecto al total de la melodía de una determinada incidencia o selección, por lo que se utilizan

porcentajes tanto para delimitar la cantidad de selecciones, así como para limitar la efectividad de la regla.

5.7.3 Cuentas

Para constatar casos particulares en los que se vaya a aplicar una regla se puede contar diferentes tipos de resultados, por ejemplo cuántas veces aparece la nota Do, y actuar en casos donde la nota aparezca más de n veces.

5.7.4 Distribución

Todos los casos anteriores parten de la base de que exista un cierto caso o que se aplique hasta lograr un cierto caso, pero la distribución de la aplicación permite la obtención de diferentes resultados, siendo que el punto de aplicación podrá variar.

5.8 Lenguaje resultado

Como se mencionaba en la parte anterior, el lenguaje cuenta con tres etapas para formar reglas completas: FiltradoCompases, FiltradoNotas, Modificación. Algunos parámetros deberán ser obligatorios, siendo que la mayoría son opcionales. Con el fin de facilitar la selección se agrega un parámetro complemento, que retorna los intervalos invertidos durante las selecciones.

Cada una de las operaciones que se permiten en el lenguaje cuentan con múltiples parámetros que pueden actuar en simultáneo, siendo que aquellos que se encuentran múltiples bajo el mismo número parámetro son excluyentes entre sí. Se utilizan los “|” para representar una disyunción excluyente de los parámetros, siendo que sólo se puede seleccionar alguno de todos los que se encuentren separados por dicho símbolo.

5.8.1 Operación de Filtrado (selección de la zona que quiero modificar):

FiltrarCompases:

Parámetro 1: (cantidadDeCompases (compases))

Parámetro 2:

- CantidadTotalDeNotas (operador numero)
- CantidadNota (Nota | altura | grado) ((Mayor| Igual |Menor) numero)
- PorcentajeNota (Nota | altura| grado) ((Mayor| Igual |Menor) numero)
- Secuencia ([Nota] | [Altura])
- Forma (asc|desc) empezando en (Nota|altura) por (numero) notas

Parámetro 3: ubicación en la música (Porcentaje de la música | (mayor | menor numero) | (Principio | Medio | Fin) | Aleatorio)

Parámetro 4: Probabilidad de que se seleccione (porcentaje)

Parámetro 5: cantidadDeSelecciones (numero)

Parámetro 6: PorcentajeDeCompases (porcentaje)

Parametro 7: Complemento de la selección (si|no)

5.8.1.1 Detalle

El filtro de compases permite hacer una selección a gran escala dentro de la música facilitando la búsqueda de determinadas características que se deseen manipular y retornando intervalos de compases completos.

La operación de filtrado se compone de 7 parámetros. El primero determina la cantidad de compases que se quiere incluir en la selección.

El segundo parámetro es la operación condición de selección, siendo que si se cumple la condición, se elegirán intervalos de la cantidad dada de compases que contengan elementos que cumplan dicha condición. Respectivamente las condiciones pueden ser:

1. Los compases contengan una cantidad determinada de notas
2. Los compases contengan una cantidad mayor, menor o igual a un número dado de tonos, notas o grados.
3. Se encuentre un porcentaje mayor, menor o igual a un número dado de ocurrencias de una nota, altura o grado
4. Exista una determinada secuencia de tonos o notas.
5. Exista una secuencia ascendente o descendente que comience por una determinada nota y se continúe por una cantidad dada de notas.

El tercer parámetro determina sobre qué parte de la música se quiere realizar la selección; habitualmente es necesario poder trabajar sobre una determinada parte de la música y no sobre la totalidad.

El parámetro cuatro determina una probabilidad de que se seleccione un determinado intervalo, esto permite generar un factor de incertidumbre en la aplicación de una regla.

El quinto parámetro permite limitar la cantidad de intervalos que se quieren obtener; de la misma forma el 6o intervalo facilita la selección de un porcentaje de intervalos dentro de lo ya seleccionado. Por último, se facilita la posibilidad de negar la selección, de forma de poder negar cada uno de los resultados de los filtros realizados, logrando en consecuencia obtener el inverso de los intervalos.

5.8.2 Filtrado de pasajes específicos a modificar:

FiltrarNotas:

Parámetro 1: (CantidadDeNotas numero)

Parámetro 2:

- Mayores (Nota|Altura |Nota altura |grado) (CantidadDeNotas numero)
- Menores (Nota|Altura | Nota altura |grado) (CantidadDeNotas numero)
- Secuencia ([Notas] | [Alturas] | [Grados]) | (Asc | Desc) (mayor|menor |igual) cantidad (conRepetidos | sinRepetidos)
- DiferenciaEntreNotas (Mayor | Menor | Igual) (numero) (cantidadIntervalosSeguidos numero)
- Grado ((Mayor|Menor|Igual) (numero)) (cantidad (Mayor|Menor|Igual) numero)
- Posicion (numero)
- NotasContinuas (numero)
- Nota (Nota | Altura)
- Aleatorio (cantidad de pasajes Numero)

Parámetro 3: CantidadNotasSiguientes (numero)

Parámetro 4: CantidadNotasAnteriores (numero)

Parámetro 5: Posicion (numero)

Parámetro 6: PorcentajeDePasajes (porcentaje)

Parámetro 7: CantidadDePasajes (numero)

Parámetro 8: Excluyentes (Si | No)

Parámetro 9 : Probabilidad de que se seleccione (Porcentaje)

Parámetro 10 : ubicación en el pasaje (Porcentaje de la música | (mayor | menor numero) | (Principio | Medio | Fin) | Aleatorio)

Parametro 11: Complemento de la selección (Si|No)

5.8.2.1 Detalle

El filtrado de pasajes consiste en lograr un nivel mayor de detalle en la selección realizada con el filtro de compases, siendo que esta operación trabajara dentro de los intervalos devueltos por la operación anterior. El resultado de un filtro de pasajes son conjuntos de notas que cumplen con una determinada condición, siendo que el resultado serán intervalos también, pero en particular más específicos.

El primer parámetro determina la cantidad de notas que se van a incluir dentro de cada intervalo creado. La segunda definición son los criterios específicos que se van a aplicar durante el filtrado, siendo respectivamente:

1. Obtiene intervalos que contengan una cantidad consecutiva de tonos, octavas, notas o grados, mayores que una cantidad dada.
2. Análoga a la anterior, sólo que agrupa si la cantidad consecutiva es menor a un número dado.
3. Verifica la existencia de una secuencia dada de notas, alturas o grados en la línea melódica. También permite, utilizando los otros parámetros, encontrar secuencias ascendentes o descendentes, con una cantidad de notas mayores o menores a un número dado, facilitando la opción de considerar o no las notas repetidas como parte de la secuencia.
4. Permite la búsqueda de intervalos musicales, retornando aquellos segmentos de la melodía donde se encuentren saltos mayores, menores o iguales a un determinado número de semitonos, por una cantidad dada de notas.
5. Facilita la búsqueda de grados mayores, menores o iguales a uno dado, en segmentos donde exista una cantidad de notas que cumplan mayor, menor o igual a una cantidad dada.
6. Devuelve un intervalo que contiene una determinada posición dentro del segmento que se esté analizando.
7. Consiste en la selección de segmentos donde las notas mantienen una dirección durante una cantidad dada de elementos.
8. Retorna todos los intervalos que contengan una determinada nota o tono.
9. Retorna una cantidad dada de pasajes aleatorios.

El tercer y cuarto parámetro permiten seleccionar el entorno al resultado de la operación de filtrado anterior, de forma que se pueden seleccionar más notas además del defecto para agregarlas al intervalo.

La quinta variable facilita la selección de un determinado intervalo del resultado, pudiendo elegir el tercer intervalo de todos los encontrados por ejemplo.

Los parámetros seis y siete permiten recortar la cantidad de resultados a una cantidad dada o un dejar un porcentaje dado.

La octava entrada determina si los intervalos resultado son excluyentes entre sí o pueden compartir partes de la melodía.

El siguiente elemento facilita la selección de la zona sobre la cual se quiere trabajar, análogo a cómo funciona en el filtrado de compases.

La última regla es la misma que la de filtrado de compases.

5.8.3 Modificación de secuencia de notas

Modificar:

Parametro 1: Desde ((Principio | Medio | Fin) | Numero)

Parametro 2: SustituirPor

- Igualar (anterior | siguiente)
- Aumentar|disminuir Semitonos (numero)
- NotaValida (mayor | Menor | igual | siguiente mayor|siguiente menor) (distancia numero | anterior | siguiente)
- Nota (altura | Nota) | Silencio
- AleatorioValida | AleatorioValida (Mayor|Menor | igual) (anterior|siguiente|distancia numero)
- Secuencia ([Notas]| [Alturas])
- AgregarSecuencia (antes|despues) ([notas] | respecto a (nota|base) [grados] | entre (anterior|siguiente) con saltos de (numero) semitonos)
- Dividir en (Numero) partes una nota

Parametro 3: Garantizando

- DiferenciaEntreNotas (Mayor | Menor | Igual) (numero)
- AparicionDeNota (Altura | Nota | Grado) (numero) (Mayor|menor|igual)
- Porcentaje (Altura|Nota|Grado) (numero) (Mayor|menor|igual)
- DistanciaEntreNotas (Mayor|Menor|Igual) (numero)
- Octava (Mayor|Menor|Igual) (numero | nota (Anterior | Siguiente))
- Secuencia (Asc|Desc) (conRepetidos Si|No) (numero)

Parametro 4: EnCasoDe

- DiferenciaEntreNotas (Mayor | Menor | Igual) (numero)
- AparicionDeNota (Altura | Nota | Grado) (numero) (Mayor|menor|igual)
- Porcentaje (Altura | Nota | Grado) (numero) (Mayor|menor|igual)
- DistanciaEntreNotas (Mayor|Menor|Igual) (numero)
- Octava (Mayor|Menor|Igual) (numero | nota (Anterior | Siguiente))

- (Si|no) sea secuencia (asc|desc) (numero) notas (conRepetidos si|no)

Parametro 5: Probabilidad (Porcentaje)

5.8.3.1 Detalle

La regla de modificación presenta el mayor nivel de complejidad, porque debe ser suficientemente exacta para permitir hacer algo bien particular, y a su vez debe permitir trabajar de forma relativa para poder referirse a funciones de más alto nivel. En particular cuenta con cinco parámetros únicamente, pero varios de ellos variables.

El primer parámetro determina la zona sobre la cual se va a realizar la modificación, ya que la modificación se realiza sobre el resultado de las operaciones de filtrado anteriores, pero eso no limita el tamaño de la melodía que puede alcanzar la operación de modificación.

El segundo parámetro determina cuál va a ser el cambio que se va a hacer, siendo que es responsabilidad del resto de las partes determinar la nota particular sobre la que se trabaja.

Respectivamente existen las modificaciones:

1. Igualar la nota seleccionada a la anterior o siguiente
2. Aumentar o disminuir una cantidad dada de semitonos la nota.
3. Cambiar la nota por una nota válida mayor, menor o igual, siguiente mayor en la escala, siguiente menor en la escala, con respecto a una nota que se encuentre a una distancia determinada dentro de la línea melódica, siendo casos particulares la siguiente o la anterior.
4. Cambiar el tono de la nota o tono y octava, o cambiarla por un silencio
5. Sustituye la nota por una nota aleatoria válida, o por una nota aleatoria mayor, menor o igual a otra nota dentro de la línea melódica que esté a una distancia dada.
6. Sustituye la nota y las notas siguientes, sólo las alturas, por las de una secuencia dada

7. Agrega entre la nota dada y la siguiente una secuencia de notas, una secuencia de grados con respecto a la base o con respecto a la misma nota o un conjunto de saltos definidos por la cantidad de semitonos entre cada salto, siendo que la cantidad de notas que se agregan dependerá directamente de la distancia entre la nota seleccionada y la nota anterior o la nota siguiente.
8. Divide la duración de la nota seleccionada, dejando una cantidad dada de copias donde la suma de las duraciones es equivalente a la de la nota que se modifica.

El tercer parámetro determina una garantía, siendo que aplicando una determinada modificación dentro de un intervalo dado se cumpla dicha garantía. Las garantías son respectivamente:

1. Luego de modificar la melodía, la diferencia de altura entre todas las notas del intervalo será mayor, menor o igual a un número determinado.
2. La cantidad de ocurrencias de un tono, nota o grado será mayor, menor o igual a un número dado.
3. El porcentaje de ocurrencias de un tono, nota o grado será mayor, menor o igual un número dado
4. Los intervalos entre las notas serán mayor, menor o igual a un número dado
5. La octava de las notas será mayor, menor o igual a un número dado, o a la octava de la nota anterior o siguiente.
6. Garantiza que después de modificar quedará una secuencia ascendente o descendente, con o sin repetidos de un determinado largo.

El cuarto parámetro es la condición para la modificación, que en caso de las garantías actuará como precondición únicamente, siendo independiente de lo que suceda después de modificar. En el caso sin garantías, servirá para determinar qué notas modificar.

En particular estas condiciones son análogas a las anteriores, siendo que lo único que se agrega o diferencia de la anterior, es que en la última restricción, existe la posibilidad de negar la condición de la secuencia.

El último parámetro es un porcentaje de éxito que se espera cuando se aplique la regla, siendo que no siempre será aplicada y permitiendo mayor diversidad de resultados.

5.8.3.2 Concatenar

Existirá un operador encargado de concatenar tanto resultados como reglas, con el único requisito de que se conserven los tipos, es decir concatenar dos reglas o dos resultados, pero no cruzados.

La ventaja de contar con este operador es que permite la réplica de partes de la música, permitiendo superar la barrera con respecto al control de la totalidad de la obra.

5.8.4 Prueba conceptual del lenguaje mediante la implementación de las reglas de modificación

Previo al desarrollo del lenguaje de creación de reglas se probó que efectivamente el mismo tiene la capacidad de desarrollo de las reglas ya existentes en Duphly [48] y propuestas por la compositora.

Continuidad en el fraseo: En particular en este caso se aplica sobre toda la melodía, por lo que no es necesario que se realice alguna selección.

Aplicando el lenguaje:

R1->FiltrarNotas (Secuencia ascendente (igual 3) conRepetidos) (excluyentes (Si)) (complemento de la selección Si)

R2 -> FiltrarNotas (Secuencia descendente (igual 3) conRepetidos) (excluyentes (Sii)) (complemento si)

R3-> FiltrarNotas (cantidadDeNotas 3) (excluyentes (si))

R4 -> Modificar (sustituirPor (NotaValida (mayor) (anterior)) (garantizando (secuencia asc) (conRepetidos si) 3) en caso de (DistanciaEntreNotas mayor 0)

R5 -> Modificar (sustituirPor (NotaValida (menor) (anterior))) (garantizando (secuencia asc) (conRepetidos si) 3) en caso de (DistanciaEntreNotas menor 0)

RF -> concatenar (concatenar (concatenar (concatenar (R1, R2) R3) R4) R5)

Limitar difusión en la dirección: Al igual que la regla anterior, esta regla se aplica sobre toda la melodía, por lo que no es necesario que se realice alguna selección.

Aplicando el lenguaje:

R1-> FiltrarNotas (DiferenciaEntreNotas Mayor 3 (cantidadIntervalosSeguidos 1))

R2-> Modificar (Desde 2) (NotaValida menor anterior)

RF-> concatenar(R1, concatenar(R2, concatenar(R3, R4)))

Continuidad en el fraseo sin permitir repetidas: Regla de funcionamiento similar a continuidad en el fraseo, sólo que no genera notas repetidas.

Aplicando el lenguaje:

R1->FiltrarNotas (Secuencia ascendente (igual 3) sinRepetidos) (excluyentes (Si)) (complemento de la selección Si)

R2 -> FiltrarNotas (Secuencia descendente (igual 3) sinRepetidos) (excluyentes (Sii)) (complemento si)

R3-> FiltrarNotas (cantidadDeNotas 3) (excluyentes (si))

R4 -> Modificar (sustituirPor (NotaValida (mayor) (anterior)) (garantizando (secuencia asc) (conRepetidos no) 3) en caso de (DistanciaEntreNotas mayor 0)

R5 -> Modificar (sustituirPor (NotaValida (menor) (anterior))) (garantizando (secuencia asc) (conRepetidos no) 3) en caso de (DistanciaEntreNotas menor 0)
RF -> concatenar (concatenar (concatenar (concatenar (R1, R2) R3) R4) R5)

Continuidad en el fraseo permitiendo repetidas: La diferencia entre este caso y los dos anteriores radica en permitir secuencias ascendentes de más de dos elementos.

Aplicando el lenguaje:

R1->FiltrarNotas (Secuencia ascendente (mayor 2) conRepetidos) (excluyentes (Si)) (complemento de la selección Si)

R2 -> FiltrarNotas (Secuencia descendente (mayor 2) conRepetidos) (excluyentes (Sii)) (complemento si)

R4 -> Modificar (sustituirPor (NotaValida (mayor) (anterior)) (garantizando (secuencia asc) (conRepetidos si) 3) en caso de (DistanciaEntreNotas mayor 0)

R5 -> Modificar (sustituirPor (NotaValida (menor) (anterior))) (garantizando (secuencia asc) (conRepetidos si) 3) en caso de (DistanciaEntreNotas menor 0)

RF-> concatenar (concatenar (concatenar (concatenar (R1, R2) R3) R4) R5)

Disminución de la utilización de la nota de blues sustituyendo por la nota anterior: El fin de esta regla es disminuir la nota de blues a 40 por ciento o menos en cada compás. Para ello primero se seleccionan los compases con más de 40 por ciento de ocurrencia de la nota de blues. Luego se selecciona la nota dentro de los pasajes. Por último, se modifica dicha nota un 60 por ciento de los casos. El porcentaje de modificación se podría haber logrado durante la selección también.

Aplicando el lenguaje:

R1 -> FiltrarCompases 1 (PorcentajeNota (grado 6) mayor 40)

R2 -> FiltrarNotas 1 (Grado igual 6)

R3-> Modificar (sustituirPor igualar anterior) Probabilidad (60)

RF->Concatenar(Concatenar(R1,R2),R3)

Disminución de la utilización de la nota de blues sustituyendo por nota aleatoria: La variante entre esta regla y la anterior es la sustitución únicamente.

Aplicando el lenguaje:

R1 -> FiltrarCompases 1 (PorcentajeNota (grado 6) mayor 40)

R2 -> FiltrarNotas 1 (Grado igual 6)

R3-> Modificar (sustituirPor AleatoriaValida)

Probabilidad (60)

RF->Concatenar(Concatenar(R1,R2),R3)

Continuidad en el fraseo con variante:

Aplicando el lenguaje:

R1->FiltrarNotas (Secuencia ascendente (igual 3) conRepetidos) (excluyentes (Si)) (complemento de la selección Si)

R2 -> FiltrarNotas (Secuencia descendente (igual 3) conRepetidos) (excluyentes (Sii)) (complemento si)

R3-> FiltrarNotas (cantidadDeNotas 3) (excluyentes (si))

R4 -> Modificar (sustituirPor (AleatorioValido (mayor) (anterior)) (garantizando (secuencia asc) (conRepetidos si) 3) en caso de (DistanciaEntreNotas mayor 0) probabilidad 60

R5 -> Modificar (sustituirPor (AleatorioValido (menor) (anterior))) (garantizando (secuencia asc) (conRepetidos si) 3) en caso de (DistanciaEntreNotas menor 0) probabilidad 60

RF -> concatenar (concatenar (concatenar (concatenar (R1, R2) R3) R4) R5)

Aumentar la repetición de notas: Regla que se enfoca en el aumento de la cantidad de notas repetidas seleccionando qué notas cambiar garantizando la continuidad en el fraseo.

Aplicando el lenguaje:

R1->FiltrarNotas (Secuencia ascendente (igual 3) conRepetidos) (excluyentes (Si)) (complemento de la selección Si)

R2 -> FiltrarNotas (Secuencia descendente (igual 3) conRepetidos) (excluyentes (Sii)) (complemento si)

R3->Modificar (Desde 2) (sustituirPor igualar anterior) Probabilidad (70)

RF -> (concatenar (concatenar (R1, R2) R3))

Buscar saltos ascendentes o descendentes y agregar escalas cromáticas o escalas diatónicas desde una nota a la otra en un 10 por ciento de los casos cada 2 compases:

R1-> FiltrarCompases (CantidadDeCompases 2) (Probabilidad de que se seleccione 50) (PorcentajeDeCompases 50)

R2-> FiltrarNotas (CantidadDeNotas 2) (DiferenciaEntreNotas mayor 4 (cantidadIntervalosSeguidos 1)

R3-> Modificar (Desde 2) (AgregarSecuencia entre anterior con saltos de 1 semitono) enCasoDe (DistanciaEntreNotas Mayor 4) probabiliad 5

R4-> Modificar (Desde 2) (AgregarSecuencia entre anterior con saltos de 2 semitono)

enCasoDe (DistanciaEntreNotas Mayor 4) probabilidad 5

RF -> concatenar (concatenar (concatenar (R1, R2) R3) R4)

Dividir notas en posiciones aleatorias y subir un semitono o un tono la primer o la segunda nota un 10 por ciento de los casos:

R1-> FiltrarNotas (CantidadDeNotas 1) (aleatorio (cantidad de pasajes 5)) Probabilidad de que se seleccione 10

R2 -> Modificar (SustituirPor (Dividir en 2 partes una nota))

R3 -> Modificar (desde 1) (sustituirPor aumentar semitonos 1) enCasoDe (DistanciaEntreNotas igual 0) probabilidad 5

R3 -> Modificar (desde 1) (sustituirPor aumentar semitonos 2) enCasoDe (DistanciaEntreNotas igual 0) probabilidad 5

RF -> (concatenar (concatenar (R1, R2) R3))

En el caso de encontrar dos notas iguales corridas, dividir la primera en dos y aumentar la segunda mitad de la nota en un semitono o un tono:

R1-> FiltrarNotas (CantidadDeNotas 2) (DiferenciaEntreNotas igual 0 (cantidadIntervalosSeguidos 1))

R2 -> Modificar (Desde 1) (SustituirPor (Dividir en 2 partes una nota))

R3 -> Modificar (Desde 2) (sustituirPor aumentar semitonos 1) enCasoDe (DistanciaEntreNotas igual 0) probabilidad 5

R3 -> Modificar (desde 2) (sustituirPor aumentar semitonos 2) enCasoDe (DistanciaEntreNotas igual 0) probabilidad 5

RF -> (concatenar (concatenar (R1, R2) R3))

Si se encuentra un intervalo ascendente dividir la segunda nota del intervalo y aumentar un tono la primer nota de la división, sobre los compases más al medio de la melodía:

R1-> FiltrarNotas (CantidadDeNotas 2) (DiferenciaEntreNotas mayor 0 (cantidadIntervalosSeguidos 1)) probabilidad de que se seleccione 10

R2 -> Modificar (Desde 2) (SustituirPor (Dividir en 2 partes una nota))

R3 -> Modificar (Desde 2) (sustituirPor aumentar semitonos 1) enCasoDe (DistanciaEntreNotas igual 0)

RF -> (concatenar (concatenar (R1, R2) R3))

Si se encuentra un intervalo descendente dividir la segunda nota del intervalo y disminuir un tono la primer nota de la división, sobre los compases más al medio de la melodía:

R1-> FiltrarNotas (CantidadDeNotas 2) (DiferenciaEntreNotas menor 0 (cantidadIntervalosSeguidos 1)) probabilidad de que se seleccione 10

R2 -> Modificar (Desde 2) (SustituirPor (Dividir en 2 partes una nota))

R3 -> Modificar (Desde 2) (sustituirPor disminuir semitonos 1) enCasoDe (DistanciaEntreNotas igual 0)

RF -> (concatenar (concatenar (R1, R2) R3))

Aumentar una octava la melodía en algún compás aleatorio:

R1 -> FiltarCompases (CantidadDeCompases 1) ubicación en la musica (Aleatorio) cantidadDeSelecciones 1 (excluyentes si)

R2 -> Modificar (sustituirPor (Aumentar 12 semitonos)) (enCasoDe Octava Mayor 0)

RF -> (concatenar (R1, R2))

Disminuir una octava la melodía en algún compás aleatorio:

R1 -> FiltarCompases (CantidadDeCompases 1) ubicación en la musica (Aleatorio) cantidadDeSelecciones 1 (excluyentes si)

R2 -> Modificar (sustituirPor (Disminuir 12 semitonos)) (enCasoDe Octava Mayor 0)

RF -> (concatenar (R1, R2))

Dividir algunas notas en 2 y aumentar una octava la primera de la división:

R1 -> FiltrarNotas (CantidadDeNotas 1) (aleatorio cantidad de pasajes 5) (probabilidad que se seleccione 10) (excluyentes si)

R2 -> Modificar (sustituirPor Dividir 2 partes una nota) (enCasoDe Octava Mayor 0)

R3-> Modificar (desde 1) (SustituirPor (aumentar semitonos 12) (en caso de distanciaEntreNotas igual 0)

RF -> (concatenar (concatenar (R1, R2) R3))

Dividir notas y la segunda nota de la división hacer un salto de 7a, 9a o 13ava:

R1 -> FiltrarNotas (CantidadDeNotas 1) (aleatorio cantidad de pasajes 5) (probabilidad que se seleccione 10) (excluyentes si)

R2 -> Modificar (sustituirPor Dividir 2 partes una nota) (enCasoDe Octava Mayor 0)

R3-> Modificar (desde 2) (SustituirPor (aumentar semitonos 10) (en caso de distanciaEntreNotas igual 0)

RF -> (concatenar (concatenar (R1, R2) R3))

Sustituir un porcentaje de notas por silencios:

R1 -> FiltrarNotas (CantidadDeNotas 1) (aleatorio cantidad de pasajes 5) (probabilidad que se seleccione 20) (excluyentes si)

R2 -> Modificar (sustituirPor Silencio) (enCasoDe Octava Mayor 0)

RF -> concatenar (R1, R2)

Transponer melodía en un compás dado n semitonos hacia arriba o abajo:

R1 -> FiltrarCompases (cantidadDeCompases 1) (ubicación en la musica aleatorio)

R2 -> Modificar(SustituirPor (aumentar semitonos 7)(enCasoDe distanciaEntreNotas igual 0)

RF -> concatenar (R1,R2)

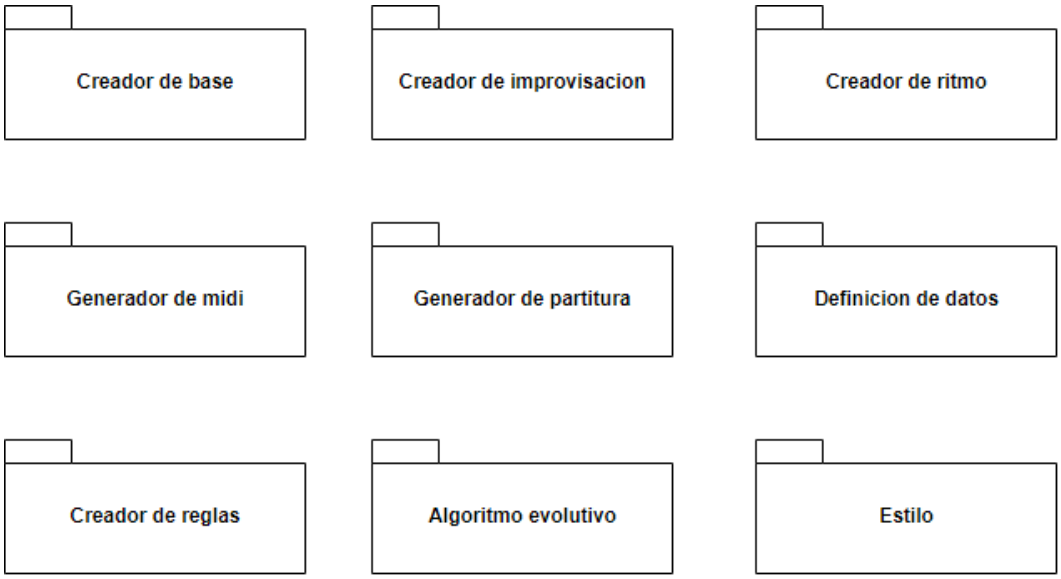
6. Implementación del sistema

El sistema propuesto cuenta con dos grandes partes, la primera es el set completo de reglas propuestas por Steedman [7] que tiene como finalidad generar la base utilizada durante la composición de la melodía. La otra gran parte del sistema consiste en el *framework* que facilita la creación de las reglas.

En este documento se presentarán los principales aspectos de la implementación dejando de lado detalles técnicos de baja importancia.

6.1 Vista general del sistema

El prototipo construido es una extensión del sistema planteado en [48]. El sistema original contaba con 6 pilares fundamentales: Creador de base, Creador de improvisación, Creador de ritmo, Generador de midi, Generador de partitura y Definición de datos. Más adelante en este documento se presentarán los detalles de modificaciones realizadas sobre dichos pilares, pero en particular se agregan: Creador de reglas, Algoritmo evolutivo y Estilo.



Los nuevos módulos que se agregan al igual que los anteriores son independientes y cuentan con una única responsabilidad asociada.

A fines de esta tesis únicamente se presentarán en mayor profundidad aquellos módulos que son nuevos o que se vieron mayormente afectados durante el desarrollo, siendo que el lector puede referirse a [48] para profundizar más sobre el resto de los módulos y el prototipo original.

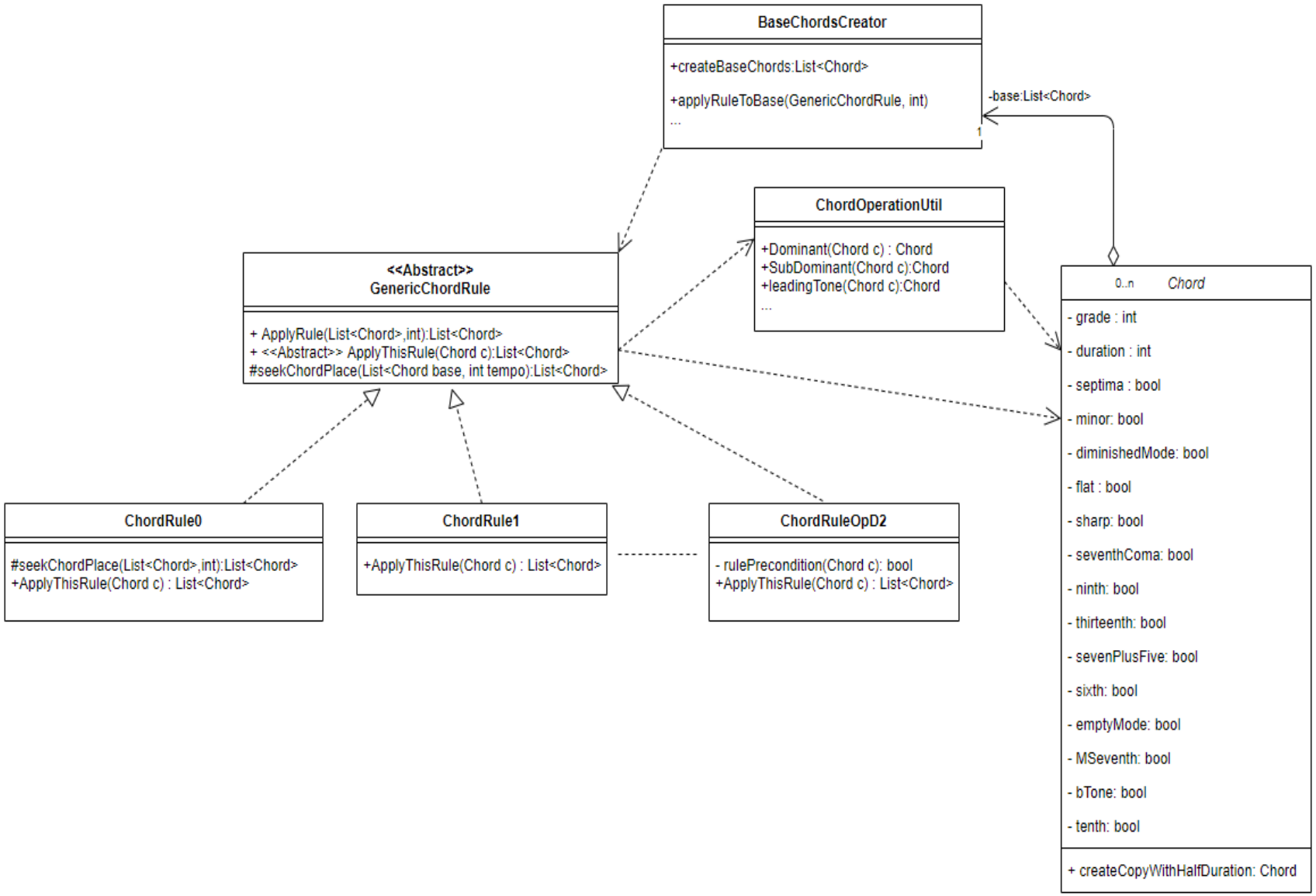
6.2 Implementación del creador de la base

A continuación se presentan los detalles de mayor importancia con respecto a la implementación del creador de base.

Con respecto al prototipo original mayormente se afectó la definición de los acordes, que cuentan con una gran variedad de formas que no se habían considerado. Por otra parte, aumentaron notoriamente la cantidad de reglas sobre acordes implementadas, por lo que fue necesario la extensión de la clase encargada de realizar cálculos sobre acordes además de funciones técnicas propias del lenguaje.

6.3 Vista general del creador de base

El diagrama que se presenta a continuación cuenta únicamente con algunas de las operaciones implementadas, ya que es de mayor importancia el mecanismo en sí que los detalles técnicos de funcionamiento.

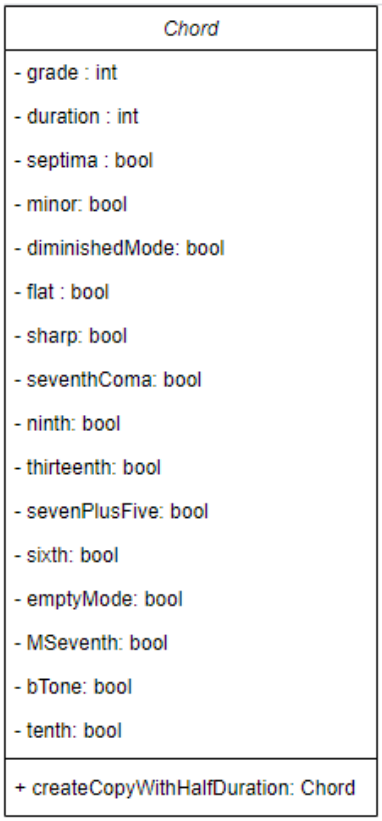

BaseChordsCreator
+createBaseChords:List<Chord>
+applyRuleToBase(GenericChordRule, int)
...
-base:List<Chord>
1
ChordOperationUtil
+Dominant(Chord c) : Chord
+SubDominant(Chord c):Chord
+leadingTone(Chord c):Chord
...
<<Abstract>>
GenericChordRule
+ ApplyRule(List<Chord>,int):List<Chord>
+ <<Abstract>> ApplyThisRule(Chord c):List<Chord>
#seekChordPlace(List<Chord base, int tempo):List<Chord>
0..n
Chord
- grade : int
- duration : int
- septima : bool
- minor: bool
- diminishedMode: bool
- flat : bool
- sharp: bool
- seventhComa: bool
- ninth: bool
- thirteenth: bool
- sevenPlusFive: bool
- sixth: bool
- emptyMode: bool
- MSeventh: bool
- bTone: bool
- tenth: bool
+ createCopyWithHalfDuration: Chord
ChordRule0
#seekChordPlace(List<Chord>,int):List<Chord>
+ApplyThisRule(Chord c) : List<Chord>
ChordRule1
+ApplyThisRule(Chord c) : List<Chord>
ChordRuleOpD2
- rulePrecondition(Chord c): bool
+ApplyThisRule(Chord c) : List<Chord>

La clase ***BaseChordCreator*** es la responsable de crear y manipular las distintas bases armónicas, siendo también responsable de que se apliquen las reglas. El juego de reglas utilizados se encuentra como resultado de la herencia de la clase abstracta ***GenericChordRule***, quien define la interfaz para interactuar con cada una de las reglas.

Por último, la clase ***ChordOperationUtil*** facilita métodos propios de la manipulación de acordes para facilitar la reutilización de código.

6.3.1 Representación de los acordes

La gramática propuesta en [7] trabaja sobre la base armónica y por lo tanto sobre acordes. Con el fin de representar los acordes de forma abstracta y similar al planteo del trabajo, se implementa una clase ***Chord***.



La implementación de los acordes cuentan principalmente con un grado y duración, siendo que el grado se utiliza como forma de abstraerse de una tonalidad en particular, al igual que sucede en [7]. Bajo la representación propuesta, un acorde por defecto sería un acorde mayor. Con el fin de permitir la representación de todas las formas de los acordes necesarias para la gramática se utiliza un conjunto de variables booleanas, que indican cada forma o combinación de formas.

Se presenta a continuación el significado de cada variable de la clase:

Septima: Representa que el acorde es un acorde con séptima.

Minor: En el caso que este valor sea verdadero, el acorde deja de considerarse mayor y pasa a ser un acorde menor.

DiminishedMode: Modo disminuido, se introduce con la regla 6 de la gramática.

Flat: Variable que se usa para representar los acordes que se les debe bajar un semitono a al grado correspondiente. Esta forma aparece por primera vez en la regla 4 de la gramática.

Sharp: Análoga a la función de la variable anterior, con la diferencia que agrega un semitono. Se introduce en la regla 6

SeventhComa: Variable que refiere a introducir la séptima por debajo de la primer nota de un acorde menor. Se corresponde con la regla c1.

Ninth, thirteenth, sixth, tenth: Estas variables refieren a los modos en los que se agregan una 9a, una 13ava, una 6a o una 10a por encima del acorde correspondiente.

SevenPlusFive: Refiere al agregado de una 5a aumentada al acorde. Se introduce en la regla b3.

EmptyMode: Indica que el acorde es semidisminuido. La gramática utiliza esta forma en la regla d2.

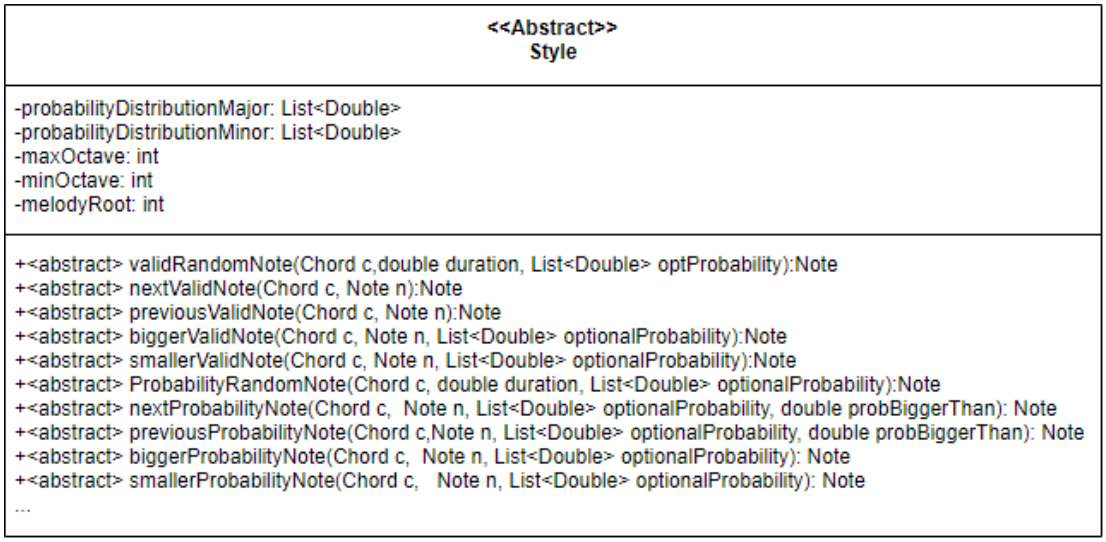
MSeventh: Representa un acorde mayor con una séptima por debajo del primer grado. Se utiliza en la regla a1.

BTone: Acorde con 3era menor. Se utiliza en la presenta por primera vez en la regla b1.

6.3.2 Estilo

Una de las metas de esta tesis radica en superar la limitación a un único género musical.

Se crea una clase abstracta que define las operaciones propias del estilo, siendo que cada subclase que represente un estilo particular deberá definir dichas operaciones.



Al igual que en el caso anterior sólo se presentan las funciones principales, en particular todas aquellas que deben ser definidas por un nuevo estilo. Las funciones que no fueron representadas son funciones de ayuda para facilitar el desarrollo.

Dentro de los atributos de la clase se presentan dos probabilidades, que acorde al estilo determinan un mapa de probabilidades de los distintos grados de las notas para bases mayores y bases menores. A su vez, el estilo contiene como atributo la tonalidad de la obra, y dos enteros para delimitar las octavas que se utilizan.

Las funciones abstractas que deben implementarse determinan las notas que son válidas para el estilo, y permiten pedir dada una base y una nota: la siguiente o anterior nota válida, una nota válida mayor o menor a la que se recibe, una nota aleatoria y notas aleatorias mayores o menores a la recibida por parámetro.

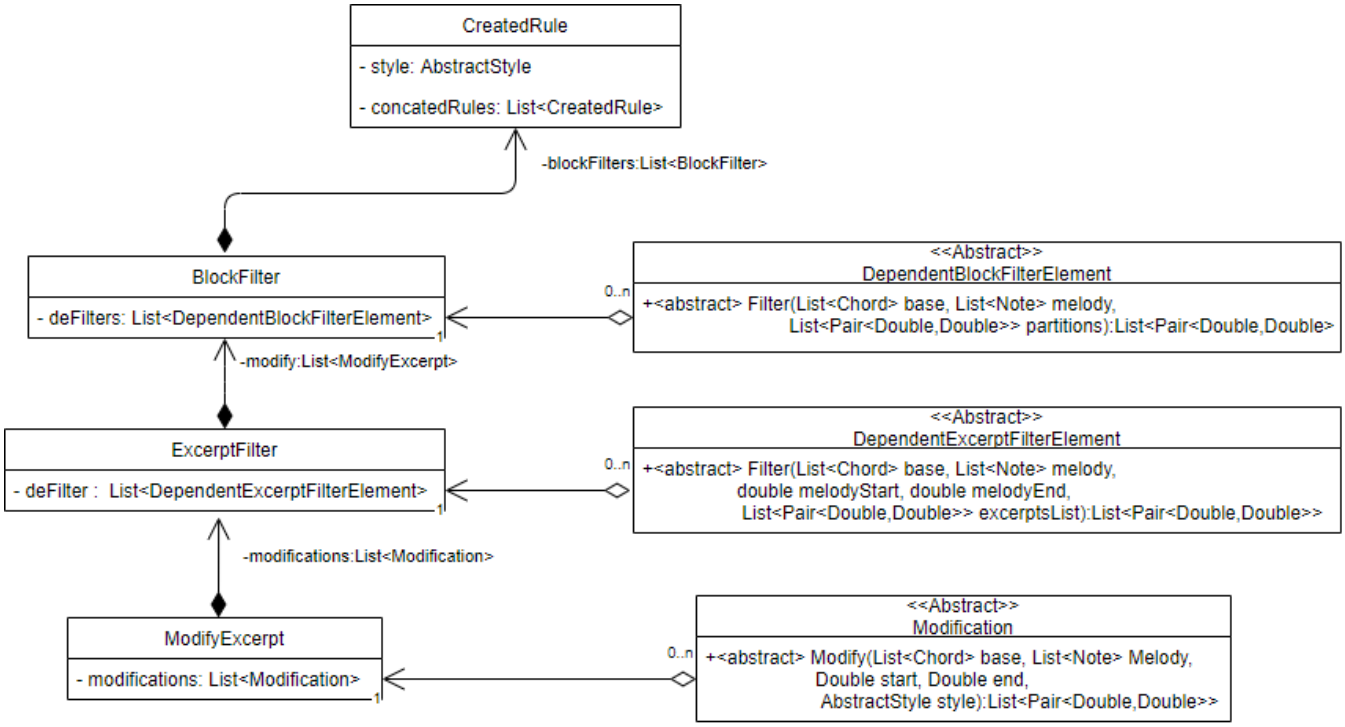
6.3.2.1 Estilo implementado

Si bien se facilita la posibilidad de utilizar diferentes estilos, durante esta tesis se conserva el estilo de blues, ya que el creador de la base genera progresiones armónicas para blues de doce compases.

6.3.3 Creador de reglas

Una de las principales funcionalidades del sistema es el creador de reglas. Recordando brevemente, el sistema permite la creación de distintos filtros y modificaciones por los cuales se va a ir pasando la melodía, contando con un orden determinado.

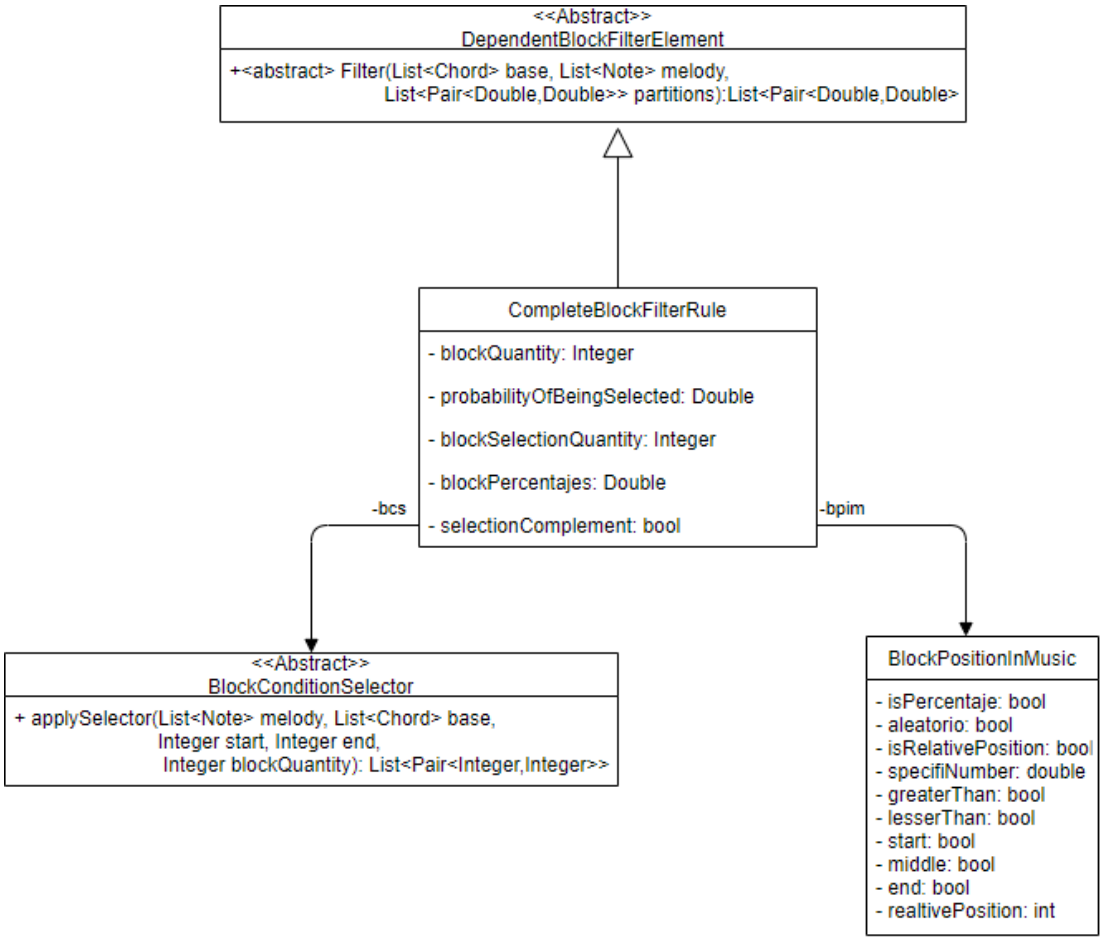
Cada regla contiene un conjunto de filtros de compases, donde cada uno de esos filtros cuenta con un conjunto de seleccionadores de pasajes, y a su vez cada selección cuenta con un conjunto de modificaciones.



Cada nivel dentro de la regla contiene una lista de filtros o modificaciones que se van a aplicar en el orden que se hayan ingresado. A su vez, los filtros y modificaciones se plantean en clases abstractas para permitir contar con varias implementaciones, aunque durante el desarrollo del prototipo se usó únicamente una implementación de cada una de las reglas abstractas.

6.3.3.1 Implementación filtro de compases

Durante la definición de reglas se planteó la existencia de diferentes parámetros, siendo algunos opcionales y otros obligatorios. Para la implementación se usan clases inmutables, cuyos atributos representan dichos parámetros y son ingresados mediante el constructor. Aquellos parámetros que son opcionales pueden valer nulo dentro el sistema y en caso de no querer usarlos se deben pasar de esa forma.



CompleteBlockFilterRule es la clase que se encarga de orquestar los distintos parámetros que se plantean en el filtro, encargándose de pedirle a ***BlockPositionInMusic*** la zona de la melodía sobre la cual se va a trabajar, luego llama a ***BlockConditionSelector***, quien tiene como responsabilidad aplicar el criterio de selección. Una vez obtenidos los intervalos correspondientes, se aplica el recorte

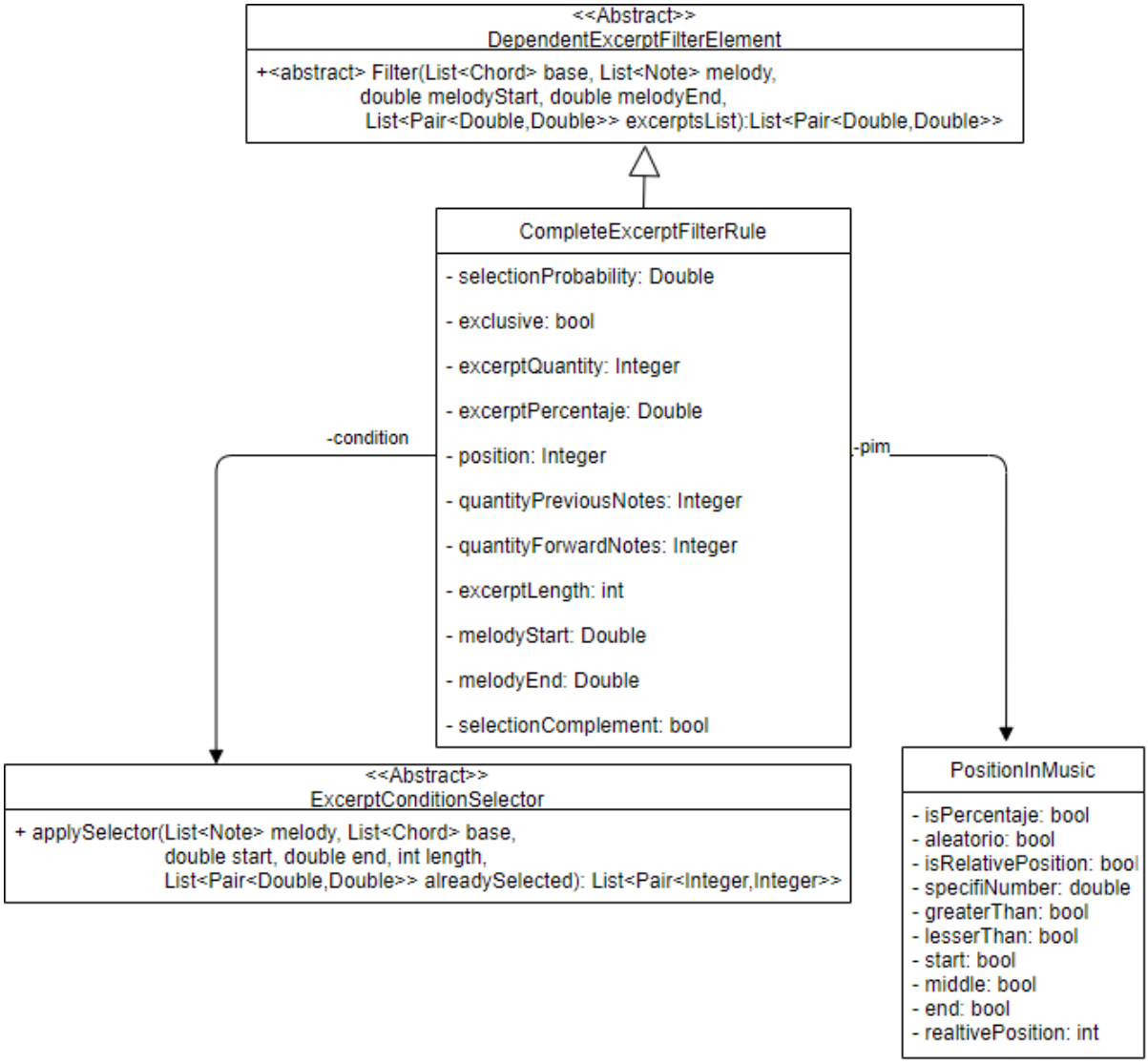
a menor cantidad de selecciones, se filtra el resultado acorde al porcentaje que se pase por parámetro, y se invierten los intervalos si se pide el complemento.

BlockPositionInMusic, cuenta con la única responsabilidad de calcular la posición sobre la cual se va a realizar la selección.

BlockConditionSelector, bajo el patrón ***Strategy***, esta clase tiene la responsabilidad de realizar la selección según el criterio implementado. En particular están implementados cada uno de los casos posibles que se plantean en el lenguaje, permitiendo extensibilidad en el caso que se deseen agregar nuevos criterios

6.3.3.2 Implementación de filtro de pasajes

El mecanismo de funcionamiento que tiene el filtro de pasajes es muy similar al que tiene el filtro de bloques, la diferencia que resguardan se corresponde principalmente por la forma de selección, ya que estas reglas seleccionan intervalos en cualquier lugar de la música y no únicamente compases. Por otra parte, los parámetros planteados en el lenguaje son diferentes y por esto varían con respecto a la selección de compases.



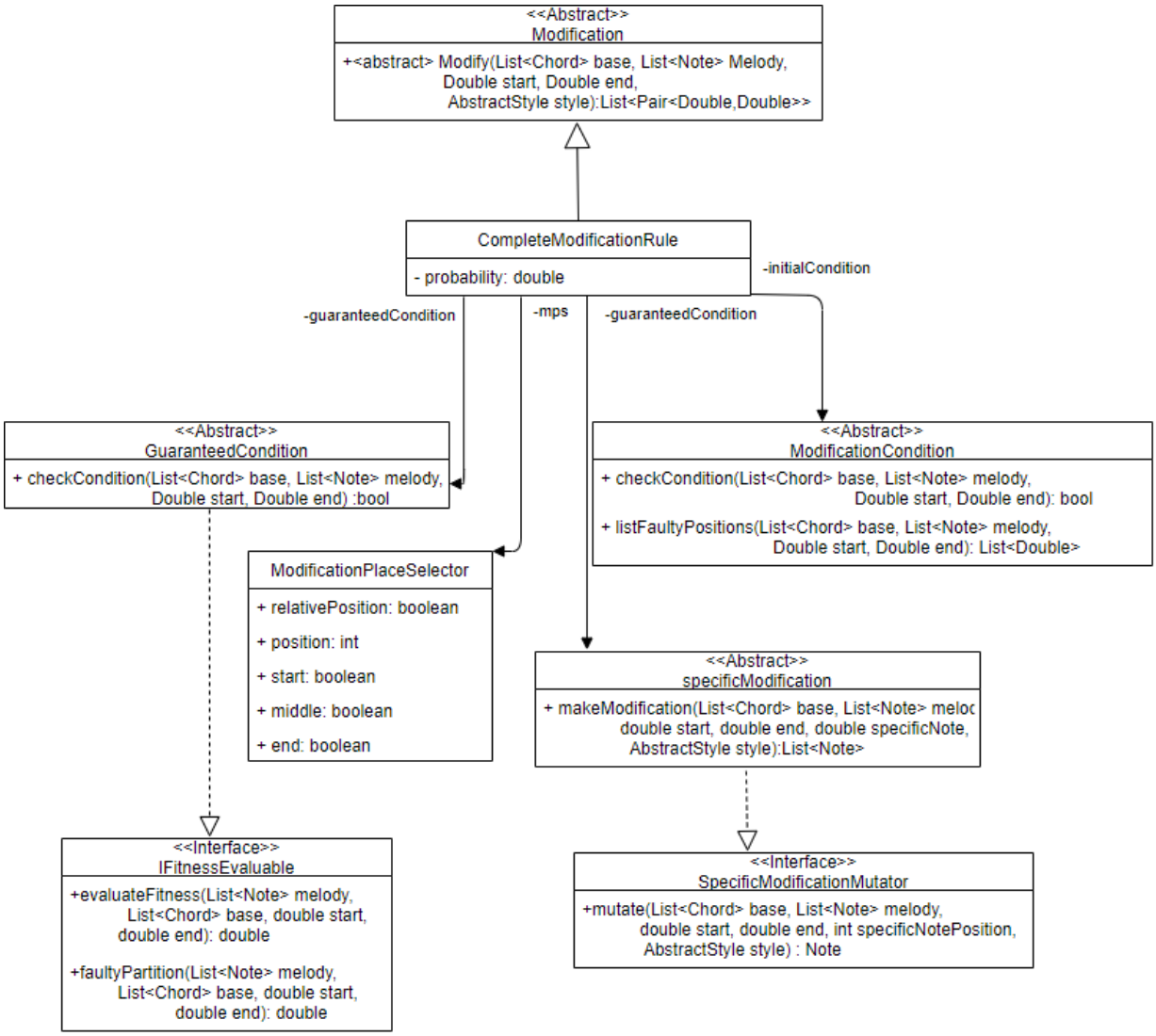
Si bien ***PositionInMusic*** cuenta con los mismos atributos que ***BlockPositionInMusic*****,** todos sus métodos tienen formas diferentes, ya que se busca posicionamiento a nivel de pasajes y no de compases.

CompleteExcerptFilterRule tiene un comportamiento similar al del filtro de compases y acorde al lenguaje de creación de reglas presentado, siendo que cada uno de sus atributos representa uno de los parámetros que se plantean.

6.3.3.3 Implementación de la modificación.

Las modificaciones presentan una mayor complejidad desde el punto de vista del prototipo.

Repasando brevemente, las modificaciones cuentan con una precondición, una acción y una característica a maximizar. Como cualquiera de estos parámetros pueden variar el criterio, se utiliza el mismo mecanismo que en las reglas anteriores con las condiciones.



Al igual que con los filtros anteriores con el fin de permitir la extensibilidad de las reglas del sistema se utilizó el patrón ***Strategy***, en cada una de las condiciones y la modificación específica.

Siguiendo el proceso que efectúa ***CompleteModificationRule***, dada una precondición para operar, aplica una modificación y chequea que se cumpla la condición que se quiere optimizar o garantizar. En el caso que no haya condición que garantizar, la precondición (***ModificationCondition***) determina la lista de lugares "defectuosos" y realiza la modificación específica (***SpecificModification***) en dichos lugares. Pero en el caso que haya una garantía, la precondición se chequea para luego modificar.

Los parámetros de entrada para lograr que se cumpla la garantía o postcondición, son un segmento de la música y una modificación, siendo que aplicando dicha modificación se espera lograr cumplir la garantía. En particular la modificación en cualquier lugar de la melodía puede hacer que se logre cumplir la garantía.

Se presenta un ejemplo: Vamos a suponer que el segmento de la línea melódica que queremos modificar cuenta con tres notas descendentes y se quiere lograr que la melodía quede de forma ascendente, utilizando una modificación que permita aumentar la altura de las notas. Para simplificar la exposición supongamos simplemente cada nota como un número que representa su altura y una secuencia 5,3,1 respectivamente. La cantidad de soluciones posibles es en teoría infinita, porque hay infinitas formas de lograr una secuencia ascendente aumentando los valores, siendo soluciones por ejemplo: 5,6,7 o 10,15,150.

En particular en el espectro de la música, las posibilidades se van a ver limitadas por el estilo y los límites superiores e inferiores de las octavas, por lo que la cantidad posibles de soluciones no serán infinitas pero pueden ser un gran número, así como puede ser un problema que no tiene solución.

Supongamos que contamos con un límite en una altura equivalente al número 40 y contamos con una secuencia descendente: 39,38,37; y se desea mediante la misma modificación que la anterior lograr una secuencia ascendente. Este caso no tiene

solución alguna, ya que el 41 escapa al límite. En particular en este tipo de casos se intenta maximizar la cantidad de notas que cumplen con la condición deseada.

Volviendo al concepto de que existen múltiples soluciones, al tratarse de un compositor de música, la variedad implica algo positivo y es de interés conservar la mayor cantidad posible de resultados.

Otra cuestión interesante a tener en cuenta, es que los filtros de compases y pasajes pueden llegar a modificar múltiples secciones de la música que contengan unas pocas notas, así como puede resultar en una única sección que contenga toda la melodía.

Desde el punto de vista de la programación nos encontramos ante un interesante problema, ya que modificar cualquier parte de la melodía puede llevar o no a una solución y no se puede saber de antemano qué camino puede funcionar. A su vez, calcular todos los casos posibles para encontrar todas las posibles soluciones puede funcionar en casos donde las secciones son pequeñas, pero las secciones pueden ser del largo total de la melodía. Ante este problema se aplica un algoritmo evolutivo.

Los algoritmos evolutivos cuentan con varias características que son útiles en este caso, tienen un grado de aleatoriedad en sus resultados, alcanzan soluciones en tiempos aceptables para un ser humano y en caso de no alcanzar una solución maximizan una propiedad.

En particular las interfaces que aparecen ***IFitnessEvaluable*** y ***SpecificModificationMutator***, son utilizadas por el algoritmo evolutivo.

6.4 Algoritmos evolutivos

Se van a introducir de forma teórica los algoritmos evolutivos y su funcionamiento previo a presentar la implementación utilizada y resultados. El algoritmo que se implementa durante esta tesis es el propuesto en [3].

6.4.1 Vista general

Los algoritmos evolutivos son un subconjunto de algoritmos pertenecientes a la rama de la computación evolutiva, en particular suelen ser utilizados en algoritmos de inteligencia artificial. Al igual que la teoría de la evolución se fundamentan en el concepto de prueba y error con el fin de lograr optimizar un determinado parámetro dentro de una población. Existen múltiples algoritmos propios de esta rama que difieren en la representación genética y detalles de implementación, siendo ejemplos de algunos de ellos: algoritmos genéticos, algoritmos meméticos, programación genética, entre otros.

6.4.2 Funcionamiento de los algoritmos evolutivos

En un principio los algoritmos evolutivos cuentan con una población de individuos primitivos de los cuales se espera lograr una optimización. Para lograr esto se van a aplicar de forma iterativa un conjunto de pasos hasta lograr una optimización aceptable.

Se itera sobre:

1. Evaluación de la **aptitud** (fitness) de cada individuo de la población
2. Selección de individuos en base a dicha **aptitud**.
3. Generación de nuevos individuos en base a **cruza** (crossover) y **mutaciones** (mutation)
4. Evaluación de **aptitud** de cada uno de los nuevos individuos
5. Reemplazo dentro de la población con los nuevos individuos en base a la **aptitud**.

El primer paso se realiza evaluando a cada individuo considerando el parámetro que se desea optimizar, por lo que se debe implementar una función que permita asignar un número que represente a cada individuo y permita compararlos, de forma de poder hacer posible el paso 2.

Para realizar el 3er paso existen muchas técnicas diferentes que son, en parte, lo que distingue a algunos tipos de algoritmos, existiendo a su vez variaciones sobre la cantidad de individuos que se consideran para **mutar** o **cruzar.**

Es importante tener claro que la generación de individuos no los hace parte de la población, sino que en el paso 5, se reemplazan individuos en la población. El criterio bajo el cual se ingresan también difiere entre los diferentes algoritmos.

6.4.3 Algoritmo implementado

Como se mencionó anteriormente el algoritmo implementado en esta tesis se corresponde al planteado en [3]. El algoritmo en particular es un algoritmo genético.

La estructura que se plantea, en términos generales, es la siguiente; en cada iteración se actualiza la población, y cada elemento de población se evalúa con la función que determina la aptitud de cada individuo. De la población probabilísticamente se seleccionan aquellos individuos más aptos; algunos de éstos pasan a la siguiente iteración intactos y otros son utilizados como base para generar nuevos individuos mediante la cruza y la mutación.

El algoritmo en sí mismo cuenta con un conjunto de variables para su funcionamiento, siendo que es variable la cantidad de individuos de la población, el número de individuos que serán reemplazados por la **cruza** y la velocidad de **mutación**. También se define el número de aptitud que se considera suficiente. Abreviamos acorde a la siguiente tabla:

Nombre	Significado
P	Población
AS	Aptitud suficiente
M	Cantidad de elementos que van a mutar en cada iteración.
C	Cantidad de elementos que se van a cruzar en cada iteración.

En un principio se generan de forma aleatoria los individuos de la población acorde a la variable de número de individuos. Se evalúa cada individuo y se verifica que ningún individuo haya superado **AS**, siendo que si algún individuo alcanzó un número superior ya se encontró una solución.

Se seleccionan de forma probabilística (**P**-**C**) elementos para agregarlos a la siguiente generación **PS**, siendo que la probabilidad de un elemento de ser seleccionado es su número de aptitud dividido la suma de todas las aptitudes. Llamémosle **h** a un individuo cualquiera de la población:

$$Prob(h) = \frac{aptitud(h)}{\sum_{i=1}^{P} aptitud(P(i))}$$

En una siguiente etapa se seleccionan **C**/2 pares de individuos de **P** utilizando la función de probabilidad descrita en la parte anterior. Se cruza cada par de individuos usando el **operador de cruza** y se agrega el par generado de la cruza a **PS**.

En esta etapa ya se cuenta con una población nueva **PS** cuya cantidad de elementos es igual a la de **P**, donde recapitulando, existen **P**-**C** individuos que se pasaron sin modificación alguna y **C** individuos que son producto de la **cruza**. Hay que considerar que los elementos seleccionados perfectamente pueden ser los mismos que se usaron para la cruza producto de la función de probabilidad, siendo que en ningún momento se especifica que sean excluyentes uno de los otros.

El paso siguiente es realizar las mutaciones correspondientes. Para esto se seleccionan de forma aleatoria con una probabilidad uniforme **M** elementos de **PS** y se les realiza la mutación correspondiente.

Se actualiza **P** asignando los elementos de **PS** y se calcula nuevamente la aptitud de cada elemento de la población.

Si algún elemento supera **AS** se retorna el elemento que tenga la mayor aptitud y termina la ejecución.

6.4.4 Operador de cruza y mutación

Generalmente suelen representarse los elementos de la población como tiras binarias. A fines prácticos se utilizarán tiras de bits para representar elementos a modo de ejemplo para esta sección.

A la hora de realizar una cruza se utiliza lo que se llaman máscaras de cruza, donde una tira binaria determina dicha máscara, siendo que esta dispone que parte de cada individuo conforma cada tira resultante.

Supongamos entonces que tenemos dos individuos seleccionado para cruzar:

1. 11010100
2. 00001010

Existen múltiples máscaras comúnmente utilizadas para la cruza:

- Cruza de punto único (*Single-point crossover*): 11110000
- Cruza de dos puntos (*Two-point crossover*): 00111100
- Cruza uniforme (*uniform crossover*): 10011001

Durante la operación se generará un individuo fruto de tomar los elementos del primer individuo cuyas posiciones coinciden con las posiciones de valor 1 de la máscara y del segundo las que corresponden con el valor 0. Luego el segundo individuo se logra seleccionando de forma inversa los elementos de cada individuo. El resultado de cruzar los dos individuos expuestos bajo la máscara de punto único tiene el resultado que se muestra a continuación. Se expresa en negrita las partes correspondientes al primer individuo de la cruza.

1. **1101**1010
2. 0000**0100**

En el caso de utilizar la máscara de dos puntos:

1. 00**0101**10
2. **11**0010**00**

Y por último el caso de utilizar la máscara de cruza uniforme:

1. **1**00**10**01**0**
2. 0**10**01**10**0

Con respecto a la mutación Mitchell [3] expone lo que se llama mutación puntual, donde se toma un bit de forma aleatoria y se muta dicho bit. Por ejemplo si mutamos el individuo 1 de la parte anterior podría ser algo de la siguiente forma (cambio resaltado en negrita):

11010100 -> 111**1**0100

6.4.5 Función de aptitud

La función de fitness tiene que como finalidad asignar un valor numérico a cada individuo que representa que tan apto se encuentra con respecto al resto, siendo que este valor numérico debe ser calculado para optimizar la característica que sea de interés.

6.4.6 Detalles de implementación en el prototipo

Como se explicó durante el desarrollo, es de interés lograr una determinada característica en una línea melódica permitiendo un determinado cambio. La población inicial que se utilizará para el algoritmo estará determinada por una cantidad determinada de copias de la línea melódica, donde, con el fin de variar la población a cada línea melódica (individuo de la población) se le aplicará una vez la modificación en diferentes notas de la línea previo a ingresarlo como población base del algoritmo.

Al momento de evaluar la aptitud de la población hay que considerar que la misma varía según cual sea el parámetro que se quiere optimizar, donde la aptitud de la melodía estará determinada por el porcentaje de elementos que cumplen con la condición que se busca optimizar. Teniendo en cuenta como se considera la aptitud, cuando se fija el valor de aptitud suficiente, se está determinando cuánto de la melodía cumplirá con el parámetro dado y será considerado correcto.

Teniendo en cuenta que la melodía es en particular una lista de notas, no hizo falta ninguna transformación para poder aplicar una máscara de cruza sobre los individuos. En la búsqueda de lograr maximizar la diversidad en los individuos, la máscara que se utiliza en cada iteración es generada aleatoriamente y cada cruza de dos elementos se hace bajo una máscara diferente. La cruza se efectiviza seleccionando qué notas se toman de cada individuo para generar nuevos individuos.

Al momento de realizar la mutación se realizó una pequeña variación sobre el algoritmo, permitiendo que las mutaciones afecten a más de una nota de cada individuo. Teniendo esto en cuenta se crea una máscara de mutación aleatoria para cada individuo y se les aplica la modificación correspondiente. Teniendo en cuenta que la población inicial son clones de una misma línea melódica con una pequeña

variación, la mutación se presenta como la forma de aumentar el nivel de cambios en la población y generar diversidad.

A la hora del uso del algoritmo dentro del prototipo se utilizaron además de las planteadas en [3] otras variables. Teniendo en cuenta que lo que se quiere maximizar puede no tener solución como se expuso en la parte anterior, se limitaron la cantidad de iteraciones posibles durante la evolución y se dejó de forma variable, también, la cantidad de notas que se desea mutar en cada iteración.

En cuanto al caso implementado, es importante notar que cuanto mayor sea la cantidad de individuos de la población, mayor sea la cantidad de iteraciones posibles y menor sea la cantidad de elementos que mutan en cada individuo durante cada iteración, mayor es la probabilidad de que se encuentre una solución, pero por otra parte aumenta considerablemente el tiempo de ejecución. Viendo el sistema desde una perspectiva mayor, una regla puede contener una gran cantidad de elementos con modificaciones que utilizan este algoritmo, y no sería aceptable una gran demora. Por lo que se optó por mejorar el tiempo de ejecución en detrimento de la calidad de la solución.

Sin embargo, en la búsqueda de obtener soluciones que cumplan mayormente con la propuesta que haga la regla se implementa un simple algoritmo de mejora en el cual se generan tres soluciones y se retorna la mejor.

El algoritmo consiste en tres etapas: la primera es aplicar el algoritmo genético sobre la melodía con la modificación correspondiente. En la segunda etapa se toma ese resultado, se busca todas aquellas posiciones que no cumplen con el criterio y se aplica el algoritmo a cada una de esas secciones de la música, con el fin de probar si solucionar cada caso puntual mejora el caso general. Por último, a la línea melódica resultado de los dos pasos anteriores se le aplica nuevamente el algoritmo evolutivo.

En la práctica aplicar el algoritmo de esta forma mejora aproximadamente un 10% la calidad de los resultados en la mayoría de los casos, y no implica una gran desventaja a nivel de tiempo de ejecución.

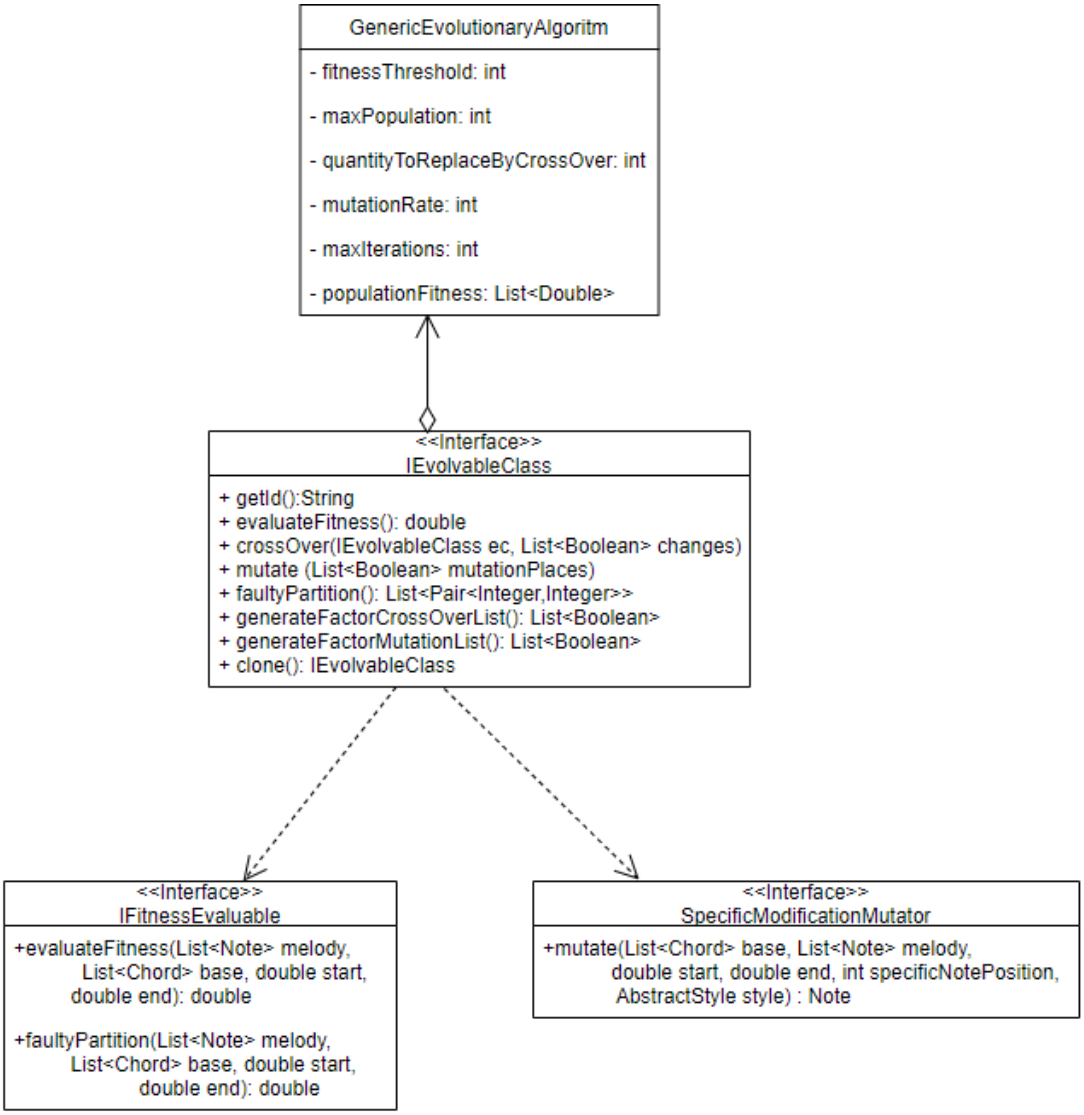
6.4.6.1 Parámetros utilizados en el algoritmo

Para determinar los parámetros que se utilizan en el algoritmo evolutivo se hicieron un conjunto de pruebas, buscando cuál es el mejor resultado que se puede obtener utilizando un tiempo de ejecución relativamente razonable. Con el algoritmo planteado en 3 partes, una línea melódica de aproximadamente 120 notas, un 10% de cruzas de individuos, una población de 500 individuos, donde en cada iteración se muta el 50 por ciento de los individuos, realizando 20 mutaciones por individuo por iteración y con un tope de 1000 iteraciones, se logra en general un resultado de entre 50 y 70 por ciento de melodía acorde a la optimización, ejecutando la modificación en aproximadamente 30 segundos en su peor caso.

6.5 Implementación del algoritmo evolutivo

El algoritmo evolutivo fue implementado de forma totalmente independiente del resto del sistema, por lo que teóricamente debería ser posible usarlo en cualquier otro dominio. Para lograr esto se crearon algunas interfaces que deben ser implementadas.

La primer interfaz existe bajo el nombre ***IEvolvableClass***, que representa los individuos que conforman la población. Por lo que el algoritmo (***GenericEvolutionaryAlgorithm***) cuenta con una lista de ***IEvolvableClass***.



La clase que implemente la interfaz ***IEvolvableClass*** cuenta con la responsabilidad de evaluar la aptitud de un individuo, mutar dicho individuo y cruzar los individuos.

Como las mutaciones y evaluaciones de fitness varían según los criterios aplicados en la clase que implementa dicha interfaz en el sistema ***ConditionEvolvableClass***, se utilizaron dos interfaces más para definir el comportamiento de las evaluaciones y modificaciones: ***SpecificModificationMutator*** y ***IFitnessEvaluable***. La relación que tienen estas interfaces con las reglas se pueden ver en el diagrama correspondiente a las reglas de modificación.

Queda plasmado entonces el mecanismo general de funcionamiento del algoritmo evolutivo en la clase ***GenericEvolutionaryAlgorithm*** y la interfaz ***IEvolvableClass***.

7. Resultados del prototipo

Con el fin de ejemplificar los resultados del trabajo dentro de la carpeta del código se podrá encontrar salidas de audio midi y pdf con las partituras correspondientes. A su vez se provee un repositorio GIT para acceder al código, partituras y algunos ejemplos de salidas: https://github.com/themanco/DuphlyCompositorAutomatico

En los anexos se encuentra el manual de usuario completo en caso de que se quiera utilizar el sistema.

Durante las pruebas, al implementar una regla que agrega notas desconociendo la duración que se está modificando, el resultado escapa a lo que está implementado para la conversión a partituras; sin embargo los archivos de audio funcionan a la perfección.

Se crearon cuatro reglas de las planteadas por la compositora, con algunas leves libertades, y se implementaron en el sistema.

Con el fin de poder comparar la melodía previo a modificarse y luego de modificarse se concatenó la música original al principio y la música luego de aplicar la regla al final, permitiendo de esta forma fácilmente ver los cambios en las partituras.

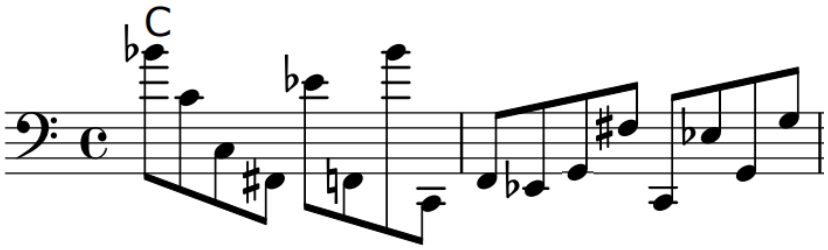
7.1 Regla agregar escalas cromáticas

Una de las reglas que se implementó fue la de agregar escalas cromáticas entre algunos intervalos.

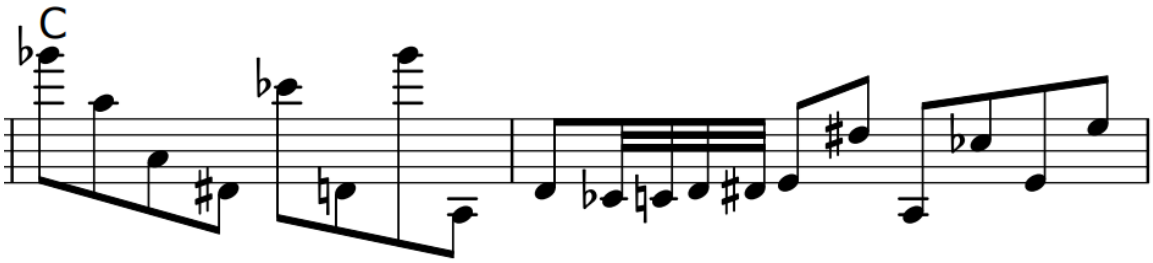
La regla de modificación que se utilizó recibió como parámetro de precondición que el intervalo entre las notas fuera exactamente 4. No se le pasó regla de garantía. La modificación puntual que se usó fue ***AddSequence*(false,1)** lo que significa que la secuencia se agrega después de la nota seleccionada y con intervalos de un semitono. No se le pasó parámetro de ubicación, y se aplica en todos los casos encontrados.

Antes de llegar a esto la regla pasa por un filtro de pasajes que forma pasajes de dos posiciones que tengan un intervalo de cuatro semitonos, y por último la regla de compás determina que solo se van a tomar de a un compás que contenga como máximo ocho notas, lo que significa que toda la música se dividió por compás y sólo se prestaron a la modificación aquellos que tenían ocho notas; la tendencia es que la duración más pequeña sea la corchea.

La melodía original en los dos primeros compases es:



Mientras que en la repetición luego de modificado:



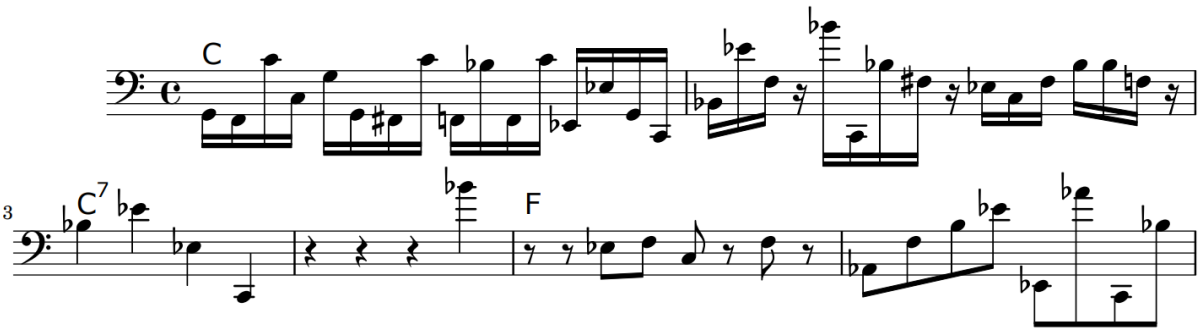
La segunda nota del segundo compás se subdividió en 4 y se agregaron todas las notas intermedias de a semitono. Considerando la imposibilidad de imprimir figuras muy pequeñas en la partitura, fue sumamente complejo lograr los parámetros para tener una modificación que se pudiese visualizar.

7.2 Regla agregar silencios

Se implementó una regla para agregar silencios a la melodía. El criterio para agregar los silencios fue bajo la precondición de que una nota estuviese en la octava 4, la modificación puntual es cambiarlo por un silencio, y se agregó una garantía de que la melodía quedará por encima de la octava 4. Esta regla se limitó al 80 por ciento de los casos.

Previo a realizar la modificación se filtraron los compases de a grupos de a dos, donde del resultado se toma solo el 40 por ciento y además contengan menos de cuatro ocurrencias de la nota Do.

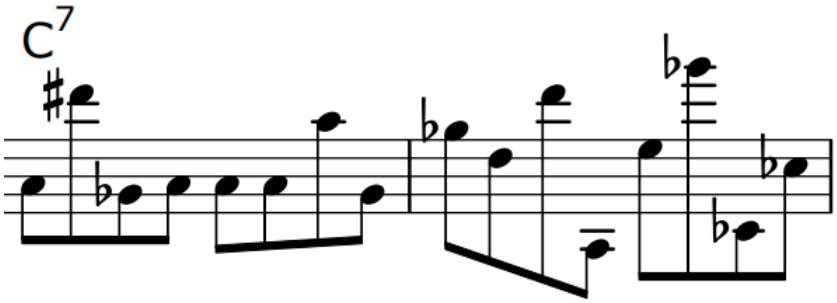
Esta regla se aplicó sobre la concatenación de la melodía.



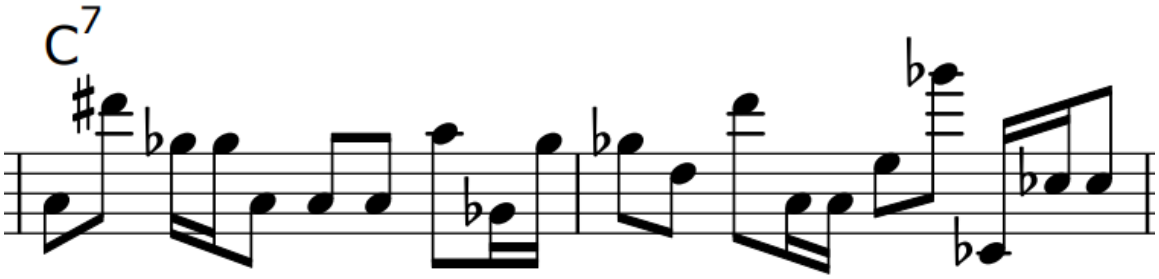
Resultando en aplicaciones de silencio como se ven en la partitura.

7.3 Regla dividir notas y agregar saltos de octava

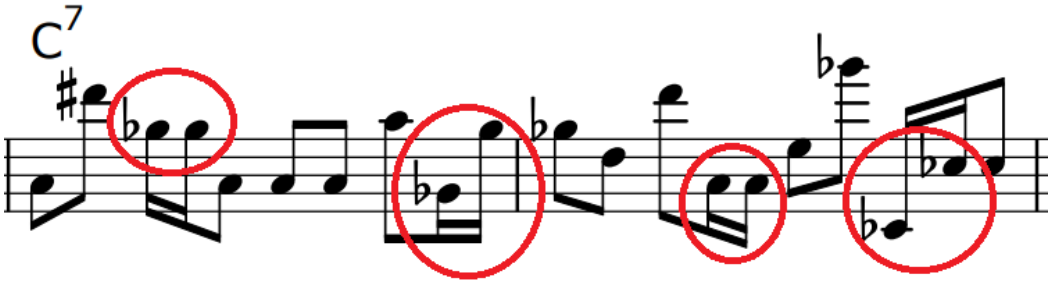
Para esta regla se implementaron dos reglas de condición completa y se agregaron a un mismo contenedor. La primera regla sustituye todas las notas Do 5 que se encuentren en el segmento. La segunda regla modifica el 50 por ciento de las notas Do 5 que encuentre en un segmento. Esto sucede luego de que se seleccionan de a dos compases, aquellos compases que cuentan con diez o menos notas, y de ese resultado se toma el 60 por ciento de los casos.



El pasaje de arriba es el tercer y cuarto compás de la música previo a la modificación.

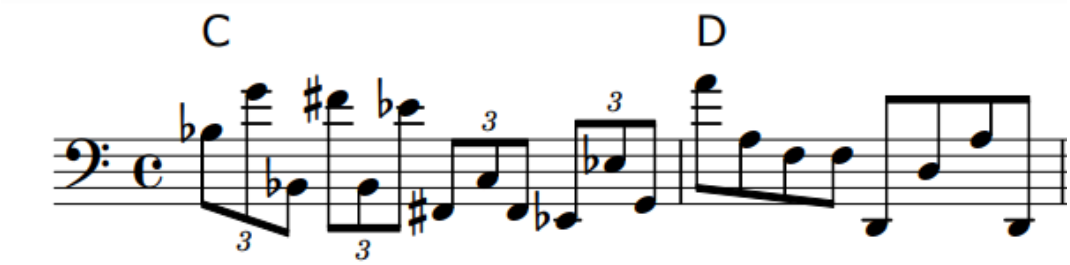


La imagen de abajo son los compases quince y dieciséis, luego de la modificación donde aparecen las notas dobles y los saltos de octava como se marca a continuación:

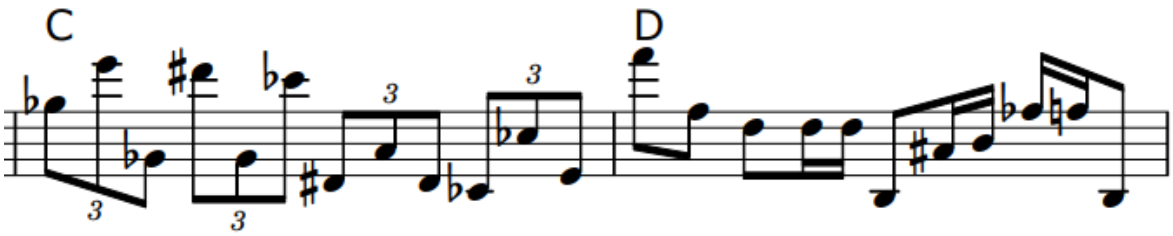


7.4 Regla dividir notas y disminuir un semitono la primera

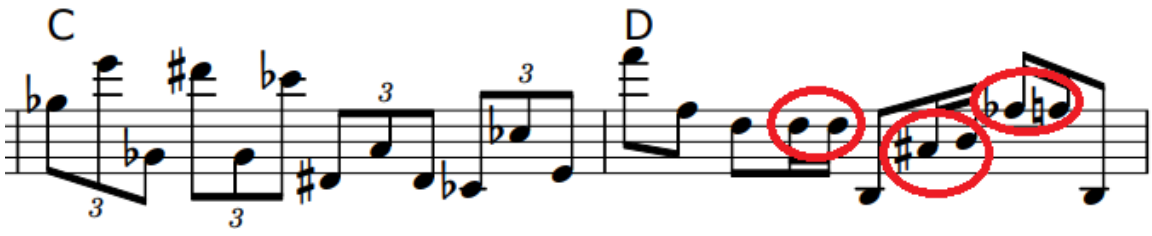
Esta regla se implementa creando un contenedor de modificaciones con dos modificaciones: la primera divide en dos las notas que se encuentren en la octava 5, y la segunda disminuye un semitono las notas cuyo intervalo sea 0. Para la primera se modifica el 80 por ciento de las veces, mientras que la segunda se modifica siempre. Esto se hace sobre el resultado de tomar el 40 por ciento de los compases de a 1.



Los primeros dos compases previo a la modificación.



Los mismos compases luego de la modificación. A continuación se remarca un caso donde se duplicó una nota y dos donde se aplicó la bajada de un semitono



8. Conclusiones

Esta tesis aporta un prototipo de composición automática o semiautomática de música categorizado dentro de los sistemas expertos y de composición algorítmica. El sistema permite la creación de blues de doce compases, facilitando la creación de la base armónica utilizando la gramática completa planteada por *Steedman* [7]. A su vez, el sistema permite la creación de música utilizando reglas ya ingresadas, así como un entorno de trabajo que permite la creación de reglas nuevas.

En particular durante este trabajo se plantea un primer acercamiento al desarrollo de un lenguaje para la creación de reglas. Dicho lenguaje permite implementar dos etapas de filtrado y una de modificación, bajo una condición y optimizando una postcondición. El sistema desarrollado consigue efectivamente componer música, siendo que los resultados dependen directamente de las reglas que se utilicen y su creación. El sistema de creación de reglas permite romper con limitantes que se contaban en el prototipo anterior, ya que permite tratar la melodía a varios niveles diferentes, así como superar la duración de doce compases con la posibilidad de mantener una línea coherente.

El prototipo implementado a su vez cuenta con la capacidad de generar la salida en archivo midi, así como en una partitura.

Durante el desarrollo y prueba del sistema de creación de reglas fue posible notar un conjunto grande de capacidades, así como algunas carencias, entre ellas la imposibilidad de manipular la métrica, siendo que ello cuenta con una complejidad bastante mayor a la hora de cuadrar con la base. Sólo algunas operaciones muy básicas fueron implementadas que modifican la métrica y no se cuenta con ningún selector basado en ella. Otra carencia del sistema es la falta de una interfaz gráfica que permita el fácil uso del mismo --en el estado actual es necesario que la persona que lo utilice cuente con conocimientos de programación para poder usarlo.

El funcionamiento del prototipo se encuentra dentro de la expectativa, siendo que la curva de aprendizaje de uso del lenguaje es bastante más compleja de lo que

parecería. Se suelen obtener resultados fuera de lo esperado producto de los niveles de aleatoriedad implementada en los algoritmos.

Más allá de eso, la primera experiencia ha sido sumamente satisfactoria, lográndose resultados muy aceptables. Es sumamente interesante poder probar una idea conceptual sobre diferentes melodías generadas de forma aleatoria, pudiendo ser la puerta a la mejora de la experiencia de estudios musicales.

Este trabajo ha despertado interés en el proceso de composición de la compositora colaboradora, ya que la forzó a analizar su propio proceso de composición. En particular, la compositora colaboradora terminó atraída hacia el concepto propuesto en esta tesis.

Para el trabajo a futuro, sería interesante implementar una interfaz gráfica que permita la creación de reglas, así como un script similar al lenguaje natural con el fin de poder alcanzar público no técnico. Por otra parte, el desarrollo del lenguaje podría dirigirse hacia la parte métrica, así como comenzar a permitir las modificaciones de la base armónica conjunto a la melodía, lo cual es algo que se comentó repetidas veces por la compositora.

9. Referencias

[1] D. Johnson and D. Ventura, "Musical Motif Discovery from Non-Musical Inspiration Sources", Computers in Entertainment, vol. 14, no. 2, pp. 1-22, 2017.

[2] B. JACOB, "Algorithmic composition as a model of creativity", Organised Sound, vol. 1, no. 3, pp. 157-165, 1996.

[3] T. Mitchell, Machine learning. New York: McGraw Hill, 2017.

[4] H. Davis and S. Mohammad, “Generating Music from Literature,” Proceedings of the 3rd Workshop on Computational Linguistics for Literature (CLFL), 2014.

[5] H. Y. Leng, N. B. M. Norowi, and A. H. Jantan, “A preliminary framework of measuring music attractiveness based on facial beauty theory,” 2016 4th International Conference on User Science and Engineering (i-USEr), 2016.

[6] H. Davis and S. Mohammad, “Generating Music from Literature,” Proceedings of the 3rd Workshop on Computational Linguistics for Literature (CLFL), 2014.

[7] M. Steedman, "A Generative Grammar for Jazz Chord Sequences", Music Perception: An Interdisciplinary Journal, vol. 2, no. 1, pp. 52-77, 1984. Available: 10.2307/40285282.

[8] C. Long, R. C.-W. Wong, and R. K. W. Sze, “Trend-MC: A Melody Composer by Constructing from Frequent Trend-Based Patterns,” 2015 IEEE International Conference on Data Mining Workshop (ICDMW), 2015.

[9] Y. Isowa, N. Iino, S. Okino, and Y. Iizuka, “A Musical Composition Assistant System Using a Stochastic Musical Model,” 2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI), pp. 1047–1048, 2017.

[10] D. Herremans, C. Chuan and E. Chew, "A Functional Taxonomy of Music Generation Systems", ACM Computing Surveys, vol. 50, no. 5, pp. 1-30, 2017.

[11] N. Collins, "Towards Machine Musicians Who Have Listened to More Music Than Us", Computers in Entertainment, vol. 14, no. 3, pp. 1-14, 2017.

[12] F. Mugica, I. Paz, A. Nebot and E. Romero, "A Fuzzy Inductive approach for rule-based modelling of high level structures in algorithmic composition systems", 2015.

[13] F. Everardo Pérez and F. Aguilera Ramírez, "Armin: Automatic Trance Music Composition using Answer Set Programming", Fundamenta Informaticae, vol. 113, no. 1, pp. 79-96, 2011.

[14] T. Anders and E. R. Miranda, “Constraint programming systems for modeling music theories and composition,” ACM Computing Surveys, vol. 43, no. 4, pp. 1–38, Jan. 2011.

[15] I. Paz, À. Nebot, F. Mugica and E. Romero, "Modeling perceptual categories of parametric musical systems", Pattern Recognition Letters, vol. 105, pp. 217-225, 2018.

[16] H. B. Lopes, F. V. C. Martins, R. T. N. Cardoso, and V. F. D. Santos, “Combining rules and proportions: A multiobjective approach to algorithmic composition,” 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 2282–2289, 2017.

[17] I. Paz, A. Nebot, E. Romero, F. Mugica, and A. Vellido, “A methodological approach for algorithmic composition systems parameter spaces aesthetic exploration,” 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 1317–1323, 2016.

[18] G. Nierhaus, Algorithmic composition. Wien: Springer, 2009.

[19] M. Edwards, "Algorithmic composition", Communications of the ACM, vol. 54, no. 7, p. 58, 2011.

[20] J. Middleton and D. Dowd, "Web-Based Algorithmic Composition from Extramusical Resources", Leonardo, vol. 41, no. 2, pp. 128-135, 2008.

[21] W. Wainiya and P. Sooraksa, 2017 21st International Computer Science and Engineering Conference (ICSEC). Bangkok: IEEE, 2017, pp. 93-96.

[22] Y. Isowa, N. Iino, S. Okino and Y. Iizuka, "A Musical Composition Assistant System Using a Stochastic Musical Model", 2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI), pp. 1047-1048, 2017.

[23] D. Herremans, C. Chuan and E. Chew, "A Functional Taxonomy of Music Generation Systems", ACM Computing Surveys, vol. 50, no. 5, pp. 1-30, 2017.

[24] A. Artinian and A. Wilson, "On Improvised Music, Computational Creativity and Human-Becoming", Leonardo Music Journal, vol. 27, no. 27, pp. 36-39, 2017. Available: 10.1162/lmj_a_01006.

[25] G. Nierhaus, Patterns of Intuition. Dordrecht: Springer Netherlands, 2015.

[26] M. Komosinski and P. Szachewicz, "Automatic species counterpoint composition by means of the dominance relation", Journal of Mathematics and Music, vol. 9, no. 1, pp. 75-94, 2014.

[27] R. Ramirez, E. Maestre and X. Serra, "A Rule-Based Evolutionary Approach to Music Performance Modeling", IEEE Transactions on Evolutionary Computation, vol. 16, no. 1, pp. 96-107, 2012.

[28] R. Whorley and D. Conklin, "Music Generation from Statistical Models of Harmony", Journal of New Music Research, vol. 45, no. 2, pp. 160-183, 2016.

[29] A. Kirke, Application of intermediate multi-agent systems to integrated algorithmic composition and expressive performance of music. University of Plymouth, 2011.

[30] P. Wiriyachaiporn, K. Chanasit, A. Suchato, P. Punyabukkana, and E. Chuangsuwanich, “Algorithmic Music Composition Comparison,” 2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2018.

[31] M. Edwards, "Algorithmic composition", Communications of the ACM, vol. 54, no. 7, p. 58, 2011.

[32] R. Oliveira and T. Tavares, "Impact of Algorithmic Composition on Player Immersion in Computer Games: A Case Study Using Markov Chains", Revista Música Hodie, vol. 18, no. 1, p. 61, 2018.

[33] Anders, Torsten, and Eduardo R. Miranda, “Interfacing Manual and Machine Composition.”, Contemporary Music Review, vol. 28, no.2, pp. 133–147. 2009

[34] P. Ball, "Computer science: Algorithmic rapture", Nature, vol. 488, no. 7412, pp. 458-458, 2012.

[35] M. Supper, "A Few Remarks on Algorithmic Composition", Computer Music Journal, vol. 25, no. 1, pp. 48-53, 2001.

[36] Anders, Torsten, “Composing Music by Composing Rules: Design and Usage of a Generic Music Constraint System”, 2007.

[37] Jacob, Bruce, “Algorithmic Composition as a Model of Creativity”, Organised Sound, 1997.

[38] Järveläinen, Hanna, “Algorithmic musical composition”, 2000.

[39] J. Dion, “Automated Music CompositionL An Expert Systems Approach”, River College Online Academic Journal, Vol. 2, no. 1, 2006.

[40] T. Anders and E. R. Miranda, “Constraint Application with Higher-Order Programming for Modeling Music Theories,” Computer Music Journal, vol. 34, no. 2, pp. 25–38, 2010.

[41] T. Anders, “Composing Music by Composing Rules: Design and Usage of a Generic Music Constraint System”, “School of Music & Arts”, 2007.

[42] T. Anders and E. Miranda, "Constraint Application with Higher-Order Programming for Modeling Music Theories", Computer Music Journal, vol. 34, no. 2, pp. 25-38, 2010. Available: 10.1162/comj.2010.34.2.25.

[43] T. Anders, C. Anagnostopoulou, and M. Alcorn, “Strasheela: Design and Usage of a Music Composition Environment Based on the Oz Programming

Model”, Multiparadigm Programming in Mozart/Oz Lecture Notes in Computer Science, pp. 277–291, 2005.

[44] Anders, Torsten, “A wizard's aid: efficient music constraint programming with Oz”, pp.152-155, 2002.

[45] Anders, Torsten, and Miranda, Eduardo, “Poster: A Computational Model that Generalises Schoenberg’s Guidelines for Favourable Chord Progressions”, Smc, 2009.

[46] Anders, Torsten, and Miranda, Eduardo, “Constraint-Based Composition in Realtime”, International Computer Music Conference, 2008

[47] G.Percival and T.Anders and G.Tzanetakis, “Generating Targeted Rhythmic Exercises for Music Students with Constraint Satisfaction Programming”, 2008.

[48] G. J. Nieves and J. I. Zunino, “Duphly: compositor de música automático,” Universidad Ort Uruguay, Facultad de Ingeniería, 2016.

Anexos

1. Anexo 1

1.1 Funcionamiento del prototipo

El prototipo implementa en su totalidad el lenguaje propuesto en la parte anterior, así como el creador de la base y algoritmo evolutivo. Para la utilización del prototipo en términos generales existe una interfaz gráfica, pero en particular para utilizar el lenguaje creador de reglas no existe una interfaz. A continuación se describirán los pasos necesarios para crear y utilizar una regla.

1.2 Requisitos previos

Para poder utilizar el *framework* de reglas es necesario contar con algún IDE que permita trabajar con Java. En particular durante el desarrollo se utilizó **Apache NetBeans 10.0**, que se puede descargar de: https://netbeans.apache.org/download/nb100/nb100.html

Por otra parte, con el fin de poder utilizar las partituras será necesario contar con LilyPond instalado en el sistema, es posible descargarlo de: http://lilypond.org/

1.3 Creación de una nueva regla

Para crear una nueva regla es necesario dirigirse al paquete ***ImprovisationRules.CustomRules*.** Dentro de dicho paquete existe una regla que se llama ***CustomRuleTemplate*.** Copiar dicha regla utilizando la opción de *refactor* dentro del mismo paquete. Cambiarle el nombre al que se desee. Una vez creada la regla, deberá aparecer en el mismo paquete. A esta altura se cuenta con una nueva regla con un nuevo nombre que cuenta con una implementación de ejemplo. Antes de modificar dicha implementación se va a agregar la regla a la lista de reglas que el sistema utiliza; para esto, el siguiente paso es abrir ***ImprovisationRulesInterface***

dentro del mismo paquete. Dentro del método ***ImprovisationRulesInterface*****,** copiar la última línea y pegarla debajo de dicha línea. La primera secuencia de caracteres es la que se va a presentar en la interfaz, asignar un nombre a dicha línea teniendo en cuenta que se va a corresponder con la regla que se está creando. Modificar el último parámetro luego de la palabra **new**, y escribir el nombre de la nueva clase que creamos mediante la copia. En este momento ya se cuenta con la regla que se creó y es utilizable dentro de la interfaz.

Con el fin de modificar la regla de *template*, hacer doble *click* en el archivo creado dentro del paquete ***ImprovisationRules.CustomRules*****.**

La clase que copiamos contiene un ejemplo de regla creado y un conjunto de métodos de ayuda para facilitar la creación de reglas. Dentro del paquete existe una clase llamada ***HelperCustomRule*** que se puede utilizar para formar algunos casos de secuencias y los comparadores correspondientes en cada caso.

Cada método se encuentra bajo un nombre descriptivo y para cada uno de ellos se encuentra comentado un ejemplo de instancia con todos los casos posibles, siendo que se eligió y descomentó uno de los parámetros para dejarlo funcional. A continuación se van a describir cada una de las instancias, parámetros y funcionamiento. Con el fin de poder aplicar a gusto las reglas hay que considerar que la octava mínima que se utiliza es la 4 y la máxima la 6 para el estilo del blues en particular.

1.3.1 Mecanismo general

Cada regla comienza por una ***CreatedRule***, que contiene una lista de filtros de compases, siendo que va a aplicar cada filtro sobre la misma melodía conservando los diferentes cambios. A su vez cada contenedor de filtro de compases (***BlockFilter***) va a aplicar sus propios filtros (***CompleteBlockFilterRule***) para obtener los distintos compases sobre los que va a trabajar. Este resultado se pasa a cada contenedor de filtro de pasajes (***ExcerptFilter*****)** que contenga. El contenedor de filtro de pasajes va a aplicar a cada intervalo sus filtros (***CompleteExcerptFilter***) y el resultado se lo va a pasar al contenedor de modificaciones (***ModifyExcerpt***). Una vez en esta etapa, a

cada intervalo recibido se le aplica cada una de las modificaciones (***CompleteModificationRule***) que se encuentren en el contenedor. En el caso que no se desee aplicar alguna etapa de filtrado, se puede agregar el contenedor de la etapa siguiente sin agregar ninguna regla, siendo que en dicho caso simplemente se saltea la etapa de filtro correspondiente.

1.3.2 Regla completa de filtro de compases

Comenzaremos por detallar los parámetros y funcionamiento de ***CompleteBlockFilterRule***.

El constructor de la regla recibe:

1. ***Integer blockQuantity***: Determina la cantidad de compases que van a componer los intervalos. Este parámetro no puede ser nulo.
2. ***BlockConditionSelector*** **bcs**: Condición de selección que se va a utilizar, se explica más adelante. Este parámetro no puede ser nulo.
3. ***BlockPositionInMusic*** **bpim**: Parámetro para determinar una zona de la música sobre la cual se desee realizar la selección. Todas las formas posibles de este parámetro están simplificadas en la clase de ayuda. Si no se desea utilizar se puede pasar en *null*.
4. ***Double probabilityOfBeingSelected***: Una vez realizada la selección según esta probabilidad se seleccionan elementos. Número del 0 al 100. Se puede pasar *null* si no se quiere utilizar.
5. ***Integer blockSelectionQuantity***: Determina un número de selecciones que se quiera considerar, si el filtro retorna más selecciones se recortan. Se puede pasar en *null*.
6. ***Double blockPercentajes***: Una vez hecha la selección de compases se van a recortar hasta contar con el porcentaje dispuesto por este parámetro. Se puede pasar en *null*.

7. ***boolean selectionComplement***: Parámetro obligatorio, en caso de ser falso no tiene efecto alguno y si es verdadero, retorna el complemento de los segmentos seleccionados.

1.3.3 Filtros de selección de compases

Se presentan los filtros, sus parámetros y su funcionamiento.

1. ***Form*(*boolean* asc, *Integer* pitch, *Integer* octave, *Integer* noteQuantity)**; que busca una secuencia que sea ascendente o descendente(**asc**), que comience en el tono(***pitch*** del 1 al 12), con octava (***octave***) en caso que se le pase siendo que puede ser *null*, de largo dado(***noteQuantity***). En caso de encontrarla retorna el compás o compases que la contengan.
2. ***NotePercentage*(*Comparador* comp, *Integer* pitch, *Integer* octave, *Integer* percentage)**: Busca los compases que cumplan contener mayor menor o igual (**comp**) porcentaje de notas al porcentaje pasado por parámetro(***percentage***). Las notas las determina la altura (***pitch***) y si se le pasa, la octava (**octave**, parámetro que puede ser *null*).
3. ***NotePercentage*(Comparador comp, *Integer* grade, *Integer* percentage)**: Mismo funcionamiento con la diferencia que considera grados en lugar de notas.
4. ***NoteQuantity*(Comparador comp, *Integer* pitch, *Integer* octave, *Integer specificNumber*)**: Caso análogo al anterior donde se verifica la cantidad de apariciones de la nota que se pase por parámetro, en lugar de porcentaje de apariciones. Si se pasa null **octave**, compara únicamente los tonos.
5. ***NoteQuantity*(Comparador comp, *Integer* specificNumber, *Integer grade*):** Cuenta la cantidad de notas de un determinado grado y verifica si se cumple la comparación entre la cantidad y el número ***SpecificNumber*.**
6. ***NotesTotal*(*Comparador* comp, *Integer specificNumber*):** Selecciona aquellos compases que contengan un total de notas según especifique **comp** (menor, mayor o igual) a ***specificNumber*.**
7. ***Sequence*(*List<Integer> pitches*, *List<Note> notes*):** Retorna aquellos compases que contengan la secuencia de tonos (***pitches***) en el mismo orden o la secuencia de notas (***notes***). El parámetro que no se quiera utilizar se debe

pasar en nulo, de lo contrario el sistema optara por utilizar uno de los dos que se le pase.

.

1.3.4 Regla de selección de pasajes

Detalle de funcionamiento y parámetros de ***CompleteExcerptFilterRule*.**

El constructor recibe:

1. ***PositionInMusic* pin**: Determina la zona en la que se aplicará el filtro dentro del intervalo. Este parámetro puede ser nulo.
2. ***Double selectionProbability***: Luego de realizada la selección determina el porcentaje de pasajes que van a obtenerse. Este parámetro puede ser nulo
3. ***boolean exclusive***: Garantiza que los intervalos resultados de la operación sean exclusivos. Es necesario determinar esta variable.
4. ***Integer excerptQuantity***: Determina el máximo de intervalos posibles para el resultado, si se supera este número en etapas previas de filtrado se recortan la cantidad de intervalos. Puede ser *null*.
5. ***Double excerptPercentaje***: Determina el porcentaje de intervalos que se obtendrán como resultado de la operación, siendo que es un porcentaje sobre el resultado de las etapas de filtro previas. Puede ser *null*
6. ***Integer position***: Determina una posición dentro de los intervalos del resultado, por ejemplo si se obtienen 5 intervalos se puede pedir el 3ero. Puede ser *null*.
7. ***Integer quantityPreviousNotes***: Determina la cantidad de notas previas a los intervalos seleccionados se quieren agregar, si excede el tamaño del intervalo sobre el que se realiza la búsqueda no se aplica al resultado. Puede ser *null*.
8. ***Integer quantityForwardNotes***: Análogo al anterior, sólo que agrega lugares al final de los resultados. Puede ser *null*.
9. ***ExcerptConditionSelector condition***: Condición de filtrado, se detallan más adelante los distintos funcionamientos. Este parámetro puede ser null, siendo

que si está en null se utiliza el resto para seleccionar todos los casos posibles que son excluyentes.

10. **Int excerptLength**: Determina el largo de cada pasaje que se seleccione. Parámetro obligatorio.
11. **boolean complement**: Parámetro obligatorio que en el caso de ser verdadero invierte los intervalos del resultado del filtrado.

1.3.5 Condiciones de filtrado de pasajes

Se describen las diferentes condiciones y su funcionamiento:

1. ***BiggerThan*(*Integer note*, *Integer octave*,*Integer quantity*)**: obtiene aquellos pasajes que contengan una cantidad ***quantity*** de notas continuas de tono ***note*** y en el caso de utilizarse octava ***octave***(puede ser nulo).
2. ***public BiggerThan*(*Integer grade* ,*Integer quantity*)**: Funcionamiento análogo al anterior donde se considera el grado ***grade*** (número del 1 al 7) en lugar de la nota.
3. ***SmallerThan*(*Integer note*, *Integer octave, Integer quantity*)**: Mismo funcionamiento que la anterior, sólo que busca los intervalos que contengan notas menores a ***quantity***.
4. ***SmallerThan*(*Integer grade*,*Integer quantity*)**: Retorna aquellos pasajes que contengan ***quantity*** cantidad de elementos con un grado menor a ***grade***.
5. ***ContinuosNotesTendency*(*Integer quantity*)**: Selecciona aquellos pasajes que conserven una tendencia ya sea ascendente o descendente por una cantidad ***quantity*** de notas.
6. ***DifferenceBeetweenNotes*(Comparador comp, *Integer number*, *Integer defectLength***): Retorna aquellos intervalos de largo ***defectLength***, que contengan saltos mayores, menores o iguales (comp) a number semitonos.
7. ***Position*(*Integer relativePosition*)**: Retorna el intervalo que contiene a la nota en la posición ***relativePosition*** con respecto al inicio del límite de búsqueda.
8. ***Random*(*int quantity*)**: Retorna intervalos en posiciones aleatorias de largo ***quantity***.
9. ***Sequency*(*List<Integer> grades*, Comparador comp, *List<Note> notes*, *List<Integer> pitches*, *Boolean* asc, *int quantity*, *boolean allowRepeat*):**

Recibe a lo sumo una de las tres listas, siendo que las otras dos deben estar en *null*. Según la lista que reciba busca secuencias de notas, de tonos o de grados y retorna aquellos intervalos donde se encuentren presentes. En el caso que todas las listas sean nulas, busca secuencias ascendentes o descendentes (asc) de largo que se comparar usando comp con ***quantity***, donde considera válido las repetidas o no según ***allowRepeat***

10. ***SpecifiNote*(*Integer note*, *Integer octave*)**: retorna aquellos intervalos que contienen el tono **note** y octava ***octave*** en caso que se pase, siendo que puede ser *null*.
11. ***SpecificGrade*(Comparador *gradeComp*, *Integer grade*, *Integer quantity*, Comparador *quantityComp*)**: Permite la búsqueda de una cantidad que se va a comparar usando ***quantityComp*** con ***quantity***, de notas que cumplan ser mayor, menor o igual, según determine ***gradeComp*** a ***grade***.

1.3.6 Regla completa de modificación

La regla completa de modificación se encuentra bajo el nombre de ***CompleteModificationRule*** y su constructor recibe los siguientes parámetros:

1. ***ModificationCondition initialCondition*:** Condición que se considera para realizar una modificación. Si no se cumple la condición no se efectúa ningún cambio. Esta condición puede ser nula; en dicho caso, se realiza la modificación sobre todos los elementos del intervalo. En el caso de haber una garantía, esta se aplicará buscando la postcondición únicamente.
2. ***GuaranteedCondition guaranteedCondition***: Condición que se intenta garantizar u optimizar. Estas condiciones se resuelven mediante el algoritmo biológico, por lo demoran más en resolver. En la salida de consola del sistema se puede ver el progreso. Otro dato importante es que estas condiciones no se pueden aplicar con cualquier modificación. En particular no funcionan con aquellas que modifiquen la duración de una nota.

3. ***SpecificModification modification***: Modificación que se efectuará sobre los elementos seleccionados, ya sea mediante la selección que efectúa la condición o durante el desarrollo de la garantía.
4. ***ModificationPlaceSelector mps***: Determina una zona sobre la cual se delimitará las modificaciones.
5. ***double probability***: Probabilidad de que se aplique la modificación. Si no hay garantía, se aplica la probabilidad antes de cada modificación atómica. Si existe una garantía, se aplica la probabilidad antes de comenzar el proceso del algoritmo evolutivo.

1.3.7 Condiciones de modificación

Se listan a continuación todas las posibles condiciones (***ModificationCondition***):

1. ***Check Octave*(Comparador comp, *int diffNumber*)**: Verifica que todos los elementos cumplan con el comparador ***comp*** con respecto a la octava determinada por ***diffNumber***.
2. ***CheckPercentage*(*Note note*, *boolean absolut*, *int percentage*,c omparador comp)**: Recibe una nota ***note***, de la cual dependiendo si ***absolut*** tiene valor verdadero, verifica la cantidad de ocurrencias del tono o el tono y la octava. Luego compara con **comp** que se cumpla el porcentaje ***percentage*** de ocurrencias.
3. ***CheckPercentage*(*Integer grade*, *int percentage*, Comparador comp):** Análoga a la anterior, solo que considera ocurrencias del grado ***grade***.
4. ***CheckSequency*(*boolean isASequency*, *boolean* asc, *Integer number*, *boolean allowRepeat*)**: La variable ***isASequency*** determina si se busca una secuencia, o que no sea una secuencia en particular, siendo ascendente o descendente según **asc** y de largo ***number***, permitiendo que hayan repetidos si ***allowRepeat*** tiene valor verdadero.
5. ***CheckTimesANoteAppear*(*Note note*, *boolean absolut*, *int numberOfApparitions*, Comparador comp)**: Compara utilizando **comp**, la cantidad de ocurrencias de una nota contra ***numberOfApparitions***, siendo que se considera el tono y octava o solo el tono dependiendo de ***absolut*.**
6. ***CheckTimesANoteAppear*(*Integer grade*, *int numberOfApparitions*, *Comparador comp*)**: Análoga a la anterior pero utilizando el grado en lugar de la nota.
7. ***DifferenceBetweenNotes*(Comparador compare, *int diffNumber*):** Suma todos los valores absolutos de las notas del intervalo y compara con compare ***diffNumber***.
8. ***IntervalBetweenNotes*(Comparador *compare*, *int diffNumber*)** : verifica que se cumpla el comparador ***compare*** en los saltos de todas las notas comparando con ***diffNumber***.

1.3.8 Garantías

El sistema permite manejar una postcondición (al aplicar una regla repetidas veces sobre un intervalo del código. Se presentan a continuación:

1. ***GuaranteeDifferenceBetweenNotes(Comparador comp, int diffNumber)****:* Garantiza el concepto presentado en la condición de modificación ***DifferenceBetweenNotes***.
2. ***GuaranteeIntervalBetweenNotes(Comparador comp, int diffNumber):*** Garantiza el concepto presentado en **IntervalBetweenNotes**.
3. ***GuaranteeOctave(Comparador comp, int diffNumber):*** Garantiza el concepto presentado en CheckOctave.
4. ***GuaranteePercentage(Note note, boolean absolut, int percentage, Comparador comp)***: Garantiza el concepto presentado en ***CheckPercentage***
5. ***GuaranteePercentage(Integer grade, int percentage, Comparador comp)***: Análoga a la anterior.
6. ***GuaranteeSequence(boolean asc, boolean allowRepeat)*****:** Garantiza que todo el segmento sea ascendente o descendente según **asc**, y considera repetidas o no según ***allowRepeat***.
7. ***GuaranteeSequence(boolean asc, boolean allowRepeat,Integer length)*****:** Análoga a la anterior, con la diferencia de que garantiza la existencia de una línea melódica ascendente o descendente de largo **length**.
8. **GuaranteeTimesANoteAppear(Note note, boolean absolut, int numberOfApparitions, Comparador comp):** Garantiza el concepto propuesto en ***CheckTimesANoteAppear***.
9. ***GuaranteeTimesANoteAppear(Integer grade, int numberOfApparitions, Comparador comp):*** Análoga a la anterior pero utilizando el grado con respecto a la base.

1.3.9 Modificaciones

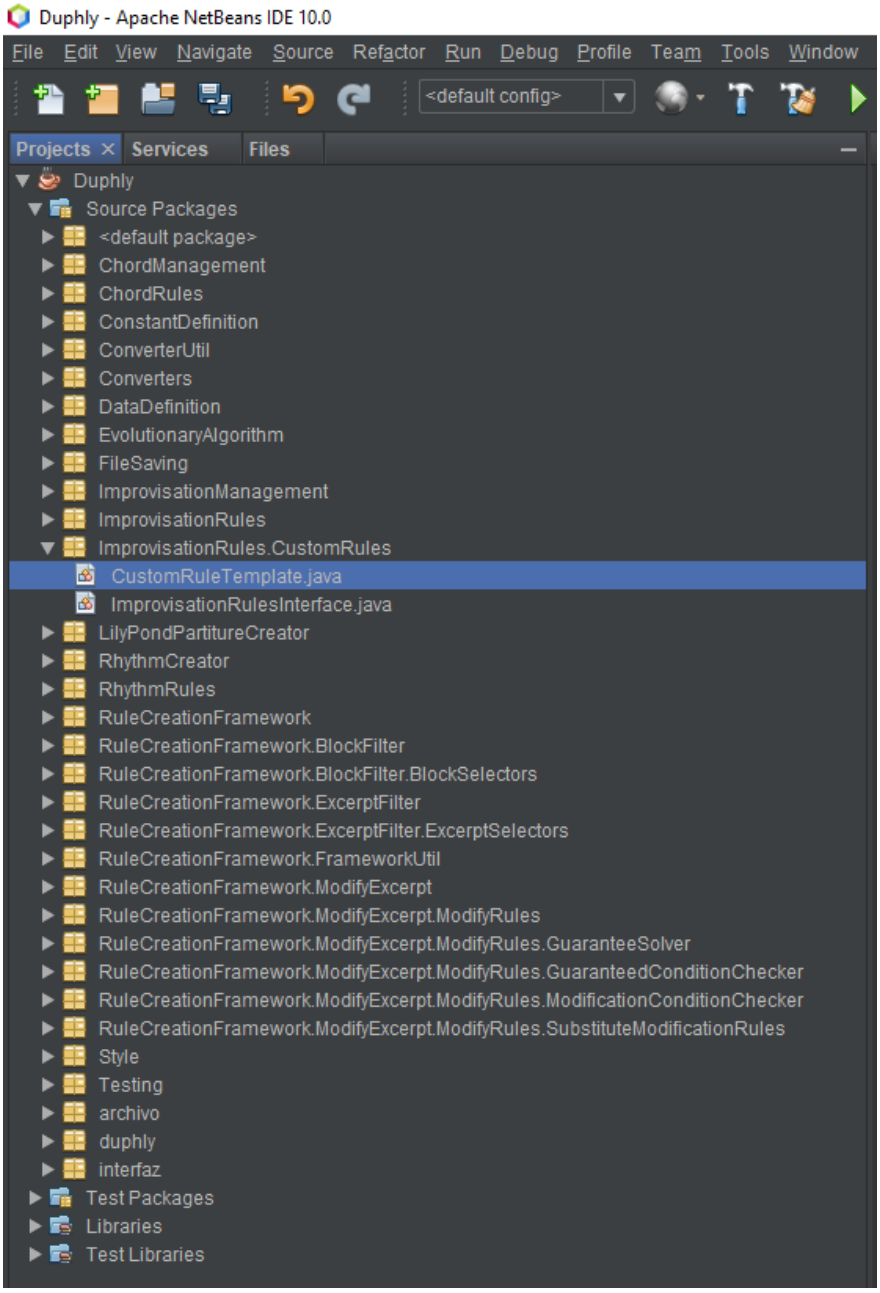
Se lista el conjunto de constructores de las modificaciones (***SpecificModification***) que se pueden utilizar:

1. ***AddSequency(boolean after, List<Note> sequence*)**: Agrega una secuencia antes o después de la nota que se haya seleccionado para modificar según ***after***, siendo que la nota divide su duración para ser ocupadas por las notas de ***sequence*.**
2. ***AddSequency(boolean after, boolean respectBase, List<Integer> grades):***
   Varía sobre la anterior que se toma una lista relativa de grados ***grades***, que según ***respectBase*** determine, serán grados con respecto a la nota que se modifica o respecto a la base.
3. ***AddSequency(boolean betweenPrevious, int semitoneQuantityJumps):*** Permite agregar una secuencia de notas entre 2 notas, siendo la que se haya seleccionado para modificar y la siguiente o la anterior según ***betweenPrevious***, se van a agregar tantas notas como permita el salto entre las notas y el tamaño de ***semitoneQuantityJumps*** que determina la distancia entre las notas que se agregan. Esta modificación no se puede usar en conjunto con las garantías.
4. ***ChangeMetric(Integer divition)***: Divide la nota que se haya seleccionado en ***divition*** cantidad de partes, siendo que conserva la altura y tono. Esta modificación no se puede usar en conjunto con las garantías.
5. ***CopyMelodyNote(boolean siguiente*)**: Copia la nota y altura de la nota siguiente si **siguiente** es verdadero, sino lo hace con respecto a la anterior.
6. ***MoveDemiTones(boolean up, int quantity)****:* Mueve la nota una cantidad de semitonos **quantity**, hacia arriba o abajo según **up**.
7. ***SubstituteForASequency(List<Note> sequence, boolean completeNote)***: Sustituye la nota que se haya seleccionado y las notas siguientes por la secuencia ***sequence***, donde se conservan las duraciones. En el caso que ***completeNote*** sea verdadero, sustituye octava y tono, sino solo tono.

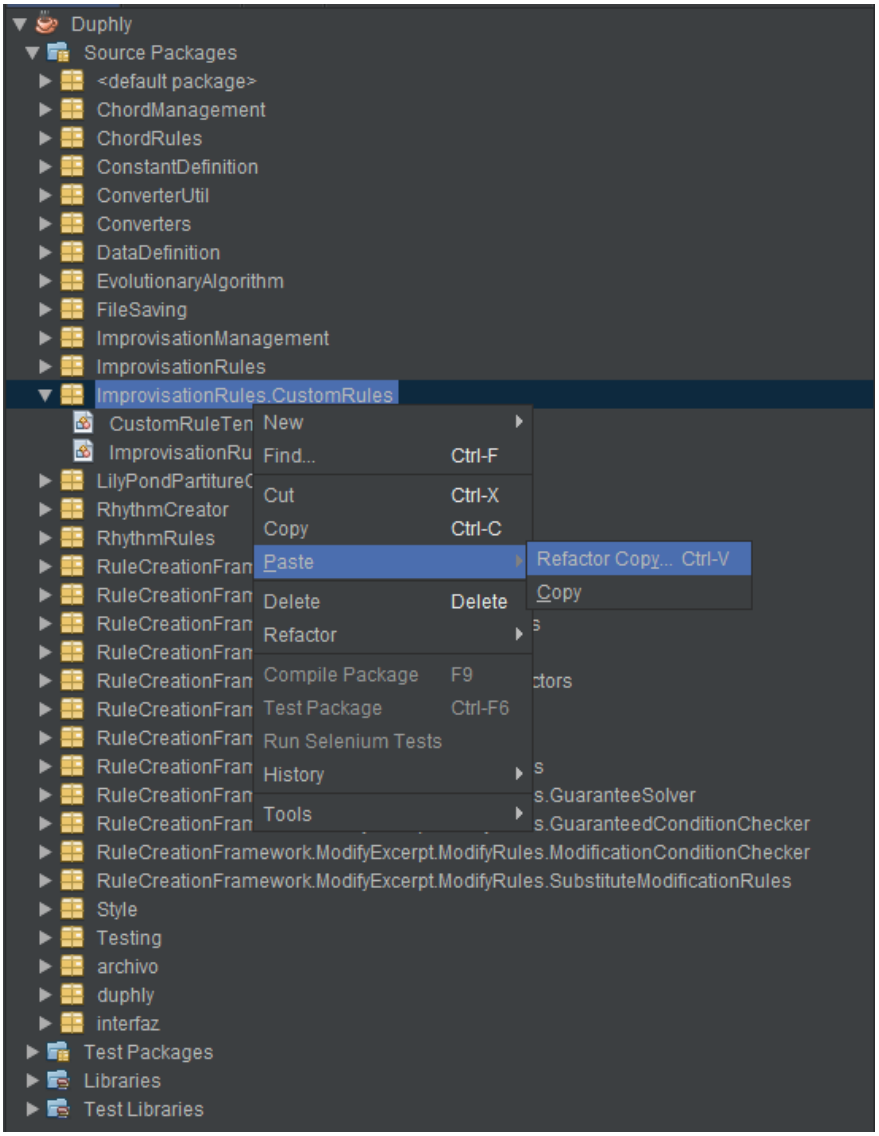
8. ***SustituteNote(int pitch, int octave, boolean noteOnly)****:* Sustituye el tono y la octava de la nota seleccionada por ***pitch*** y ***octave*** en el caso que ***noteOnly*** sea falso, si es verdadero copia solo el tono.
9. ***SustituteNote(boolean silence)***: Sustituye la nota por un silencio conservando la duración.
10. ***SustituteRandomValidNote(Comparador comp, int distance, List<Double> optionalProbability)****:* Sustituye la nota seleccionada por una nota aleatoria mayor o menor, según defina **comp,** a la siguiente (***distance***=1) o a la anterior (***distance***= -1), si ***distance*** toma otro valor se cuenta hasta llegar a la nota en la posición relativa con respecto a la seleccionada. Si **comp** se define con valor de igualdad no tiene efecto la operación.
11. ***SustituteRandomValidNote(boolean constraint)****:* Permite definir ***constraint*** como falso y retorna una nota aleatoria válida o real.
12. ***SustituteValidNote(Comparador comp, int distance, boolean nearNote)****:* Sustituye la nota seleccionada con la siguiente mayor o siguiente menor (**segunComp)** a la nota que se encuentra a ***distance*** cantidad de notas de la nota seleccionada. Sí ***nearNote*** es falso, retorna una nota mayor o menor pero no necesariamente la siguiente de la escala.

1.3.10 Visualización de pasos para crear reglas.

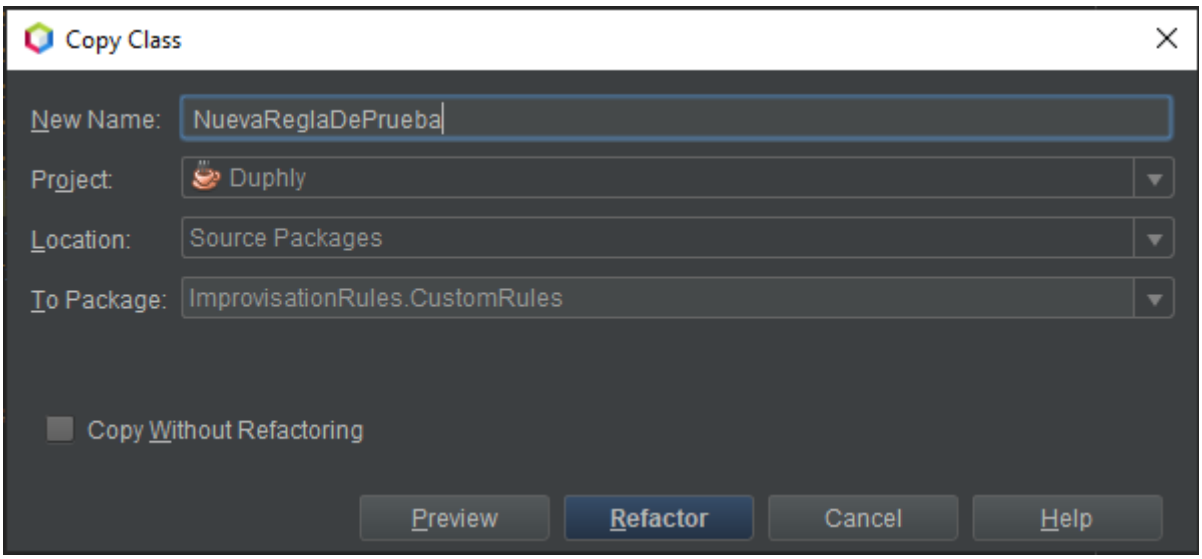
1- Abrir el paquete ImprovisationRules.CustomRules



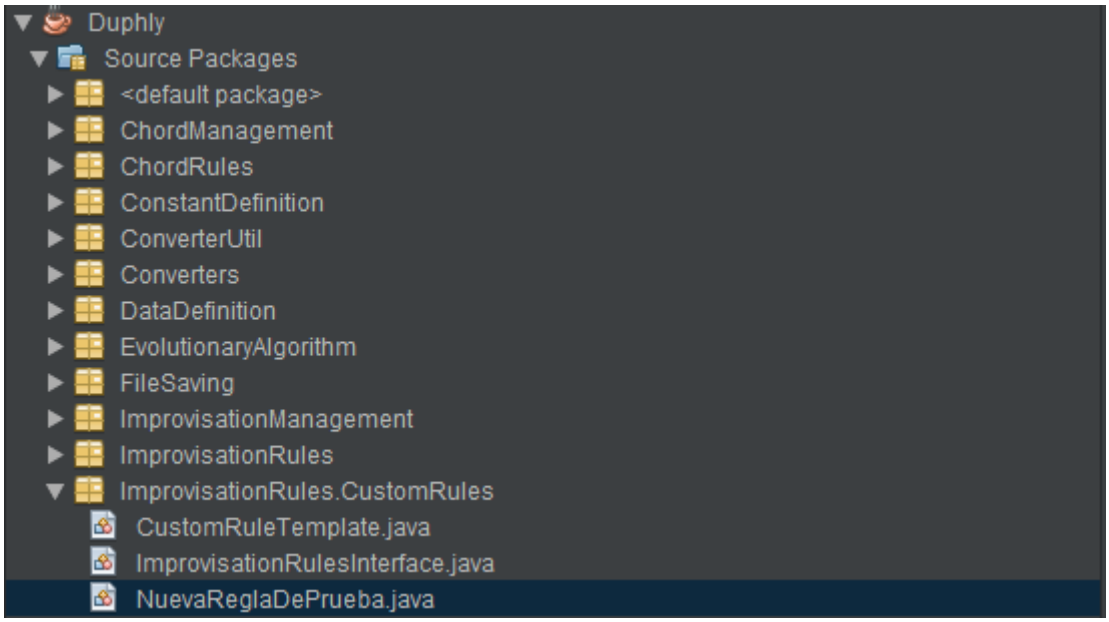
Copiar **CustomRuleTemplate** y pegar en el mismo paquete utilizando la herramienta de Refactor.



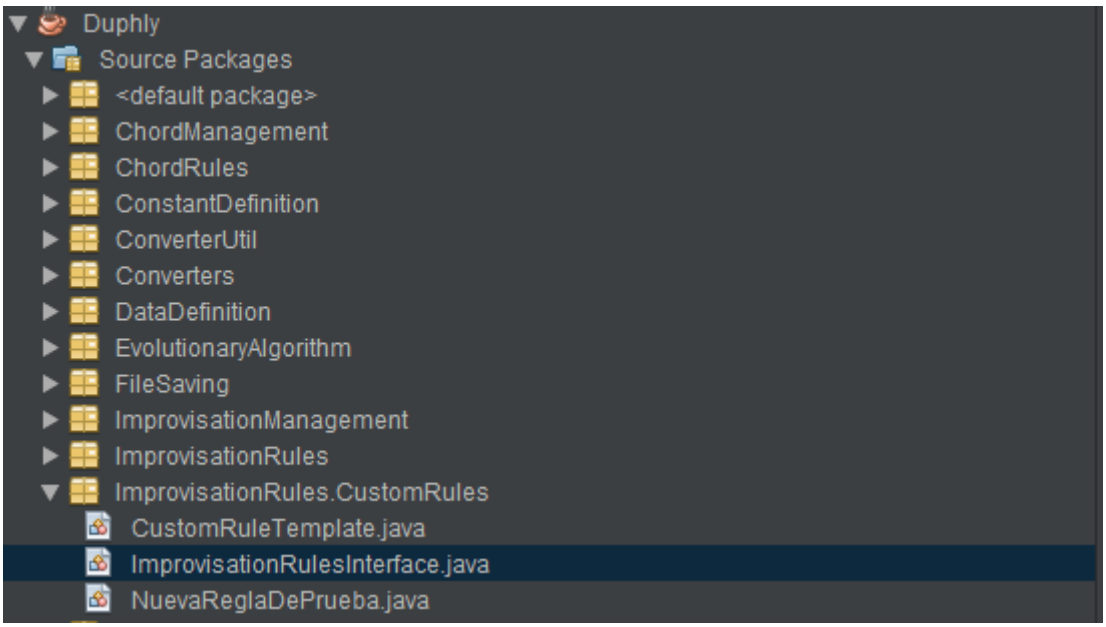
Cambiar el nombre del archivo y presionar en el botón refactor.



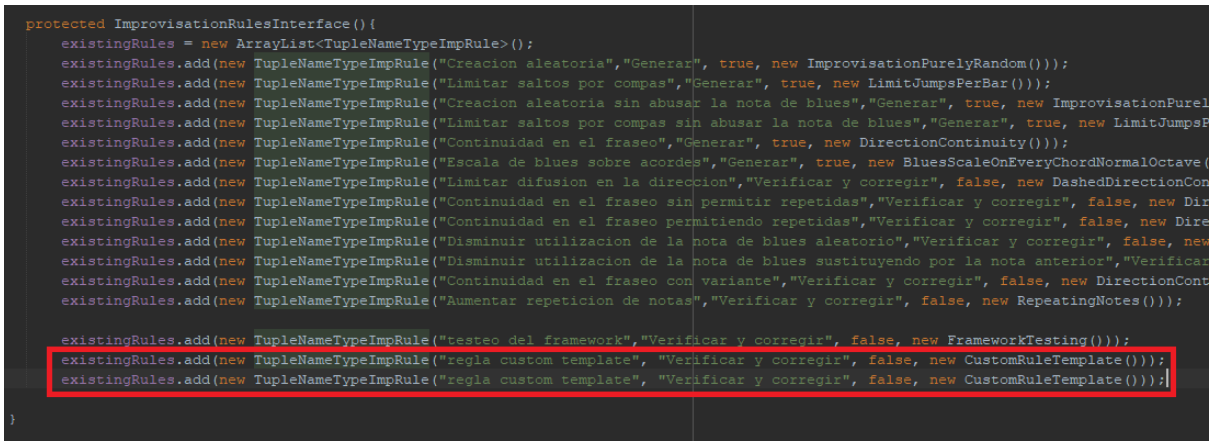
La nueva regla que acabamos de crear debería aparecer dentro del mismo paquete



Abrir **ImprovisationRulesInterface**



Copiar y pegar la última línea dentro del método **ImprovisationRulesInterface**

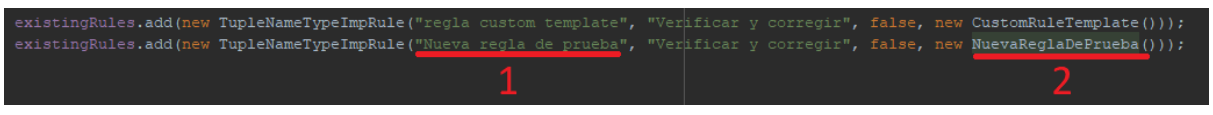


```
protected ImprovisationRulesInterface(){
    existingRules = new ArrayList<TupleNameTypeImpRule>();
    existingRules.add(new TupleNameTypeImpRule("Creacion aleatoria","Generar", true, new ImprovisationPurelyRandom()));
    existingRules.add(new TupleNameTypeImpRule("Limitar saltos por compas","Generar", true, new LimitJumpsPerBar()));
    existingRules.add(new TupleNameTypeImpRule("Creacion aleatoria sin abusar la nota de blues","Generar", true, new ImprovisationPurel
    existingRules.add(new TupleNameTypeImpRule("Limitar saltos por compas sin abusar la nota de blues","Generar", true, new LimitJumpsP
    existingRules.add(new TupleNameTypeImpRule("Continuidad en el fraseo","Generar", true, new DirectionContinuity()));
    existingRules.add(new TupleNameTypeImpRule("Escala de blues sobre acordes","Generar", true, new BluesScaleOnEveryChordNormalOctave(
    existingRules.add(new TupleNameTypeImpRule("Limitar difusion en la direccion","Verificar y corregir", false, new DashedDirectionCon
    existingRules.add(new TupleNameTypeImpRule("Continuidad en el fraseo sin permitir repetidas","Verificar y corregir", false, new Dir
    existingRules.add(new TupleNameTypeImpRule("Continuidad en el fraseo permitiendo repetidas","Verificar y corregir", false, new Dire
    existingRules.add(new TupleNameTypeImpRule("Disminuir utilizacion de la nota de blues aleatorio","Verificar y corregir", false, new
    existingRules.add(new TupleNameTypeImpRule("Disminuir utilizacion de la nota de blues sustituyendo por la nota anterior","Verificar
    existingRules.add(new TupleNameTypeImpRule("Continuidad en el fraseo con variante","Verificar y corregir", false, new DirectionCont
    existingRules.add(new TupleNameTypeImpRule("Aumentar repeticion de notas","Verificar y corregir", false, new RepeatingNotes()));

    existingRules.add(new TupleNameTypeImpRule("testeo del framework","Verificar y corregir", false, new FrameworkTesting()));
    existingRules.add(new TupleNameTypeImpRule("regla custom template", "Verificar y corregir", false, new CustomRuleTemplate()));
    existingRules.add(new TupleNameTypeImpRule("regla custom template", "Verificar y corregir", false, new CustomRuleTemplate()));

}
```

Modificar el primer parámetro asignando el nombre que se va a ver en la interfaz y el último luego del new con el nombre que se le puso a la clase



```
existingRules.add(new TupleNameTypeImpRule("regla custom template", "Verificar y corregir", false, new CustomRuleTemplate()));
existingRules.add(new TupleNameTypeImpRule("Nueva regla de prueba", "Verificar y corregir", false, new NuevaReglaDePrueba()));
                                           1                                                                  2
```

Habiendo realizado estos pasos ya es posible probar la regla.

Dentro del archivo correspondiente a la regla vamos a encontrar todos los métodos necesarios para el funcionamiento; en particular se puede variar aquellos que se deseen o manipular el código para cambiar la forma de las reglas.

```
private SpecificModification crearCambioParaModificar(){
    //return new AddSequency(false,1);

    //return new ChangeMetric(2);

    //return new CopyMelodyNote(true);

    //return new SubstituteForASequency(HelperCustomRule.createNoteSequency(),true);

    //return new SustituteNote(true);

    //return new SustituteRandomValidNote(false);

    //return new SustituteValidNote(HelperCustomRule.ComparadorMayor(),3,false);

     return new MoveDemiTones(true,3);
}
```

Dentro del archivo la función **crearCambioParaModificar** presenta todas las posibles modificaciones que se pueden hacer comentadas; des comentando otra línea se puede lograr otro resultado, o se puede modificar los valores de los parámetros.

```
private ModificationCondition crearCondicionModificacion(){
   return new CheckOctave(HelperCustomRule.ComparadorMayor(),4);

   // return new CheckPercentage(5,30,HelperCustomRule.ComparadorMenor());

   // return new CheckSequency(true,true,3,true);

   // Note dummyNote = HelperCustomRule.createNote(1, 5); // el 1 refiere a la nota
   // return new CheckTimesANoteAppear(dummyNote, true, 3,  HelperCustomRule.Compar

   // return new DifferenceBetweenNotes(HelperCustomRule.ComparadorMenor(),5);
   // return new IntervalBetweenNotes(HelperCustomRule.ComparadorMayor(),2);



}
```

Dentro de la función **crearCondicionModificacion**, se puede seleccionar o cambiar la condición que se verifica previo a una modificación.

```
private GuaranteedCondition crearGarantiaModifiacion(){
    //return new GuaranteeDifferenceBetweenNotes(HelperCustomRule.ComparadorMayor(),3);

   //return new GuaranteeIntervalBetweenNotes(HelperCustomRule.ComparadorMayor(),3);

    /* Note dummyNote = HelperCustomRule.createNote(1, null); // el 1 refiere a la nota Do
    return new GuaranteePercentage(dummyNote,(dummyNote.getOctave()!= -1),50,
                                        HelperCustomRule.ComparadorMayor());*/

    /*Note dummyNote = HelperCustomRule.createNote(1, null); // el 1 refiere a la nota Do
   return new GuaranteeTimesANoteAppear(dummyNote,(dummyNote.getOctave()!=-1),
                                    10,HelperCustomRule.ComparadorMenor());*/

    //return new GuaranteeSequence(true, true, 3);

   return new GuaranteeOctave(HelperCustomRule.ComparadorMayor(),4);

}
```

Dentro de la función **crearGarantiaModificación** se encuentran todos los ejemplos de garantías posibles.

```
private ModifyExcerpt crearContenedorDeModificacionesAAplicarse(){
    ModifyExcerpt me = new ModifyExcerpt();
    me.addModification(this.crearReglaCompletaDeModificacion());
    return me;
}

private CompleteModificationRule crearReglaCompletaDeModificacion(){
    CompleteModificationRule cr;
    cr = new CompleteModificationRule(this.crearCondicionModificacion(),
                                      this.crearGarantiaModifiacion(),
                                      this.crearCambioParaModificar(),
                                      HelperCustomRule.LugarModificacionPrincipio(),80);
    return cr;
}
```

crearReglaCompletaDeModificacion, utilizando lo ya definido crea la regla correspondiente. Y luego **crearContenedorDeModificacionesAAplicarse**, crea el contenedor y agrega a la regla creada.

```
private ExcerptConditionSelector crearCondicionDePasajes(){
    //return new ContinuosNotesTendency(5);

    //return new DifferenceBeetweenNotes(HelperCustomRule.ComparadorMayor(),3,3);

    //return new Position(3);

    //return new Random(10);

    /*return new Sequency(HelperCustomRule.createGradesSequency(),
                        HelperCustomRule.ComparadorMayor(),null,null,true,3,true);*/

    //return new SmallerThan(5,3);

    //return new SpecifiNote(1,null);

    /*return new SpecificGrade(HelperCustomRule.ComparadorMayor(), 3, 5,
                               HelperCustomRule.ComparadorMenor());*/

    return new BiggerThan(5,4);

}
```

Bajo la función **crearCondicionesDePasajes** se encuentran todas las posibles condiciones que se pueden crear.

```
private ExcerptFilter crearContenedorDeFiltrosDePasajesAAplicarse(){
    ExcerptFilter ef = new ExcerptFilter();
    ef.addModification(this.crearContenedorDeModificacionesAAplicarse());
    ef.addDependentFilter(this.crearReglaDeFiltroDePasajesCompleta());
    return ef;

}

private CompleteExcerptFilterRule crearReglaDeFiltroDePasajesCompleta(){
    CompleteExcerptFilterRule cefr;
    cefr = new CompleteExcerptFilterRule(null, 50.0, true, 10, null,
                                         null, 2, 3, this.crearCondicionDePasajes(), 0, true);
    return cefr;
}
```

La función **crearReglaDeFiltrosDePasajes** crea la regla utilizando el filtro creado anteriormente y luego la función **crearContenedorDeFiltrosDePasajesAAplicarse**, crea el contenedor de reglas de filtro y a su vez recibe el contenedor de modificaciones, siendo que va a pasar los intervalos de aplicar los filtros a las modificaciones.

```
private BlockConditionSelector crearCondicionDeFiltroDeCompases(){
    //return new Form(true,1,null,3);

    //return new NotePercentaje(HelperCustomRule.ComparadorMayor(),1,null,20);

    //return new NoteQuantity(HelperCustomRule.ComparadorMenor(), 1,null, 4);

    //return new Sequence(null,HelperCustomRule.createNoteSequency());

    return new NotesTotal(HelperCustomRule.ComparadorIgual(), 4);
}
```

crearCondicionDeFiltroDeCompases contiene todos los constructores posibles de filtros de compases.

```
private BlockFilter crearContenedorDeFiltrosDeCompases(){
    BlockFilter bf = new BlockFilter();
    bf.addExcerptFilter(this.crearContenedorDeFiltrosDePasajesAAplicarse());
    bf.addDependentFilter(this.crearReglaCompletaDeFiltroDeCompases());
    return bf;
}

private CompleteBlockFilterRule crearReglaCompletaDeFiltroDeCompases(){
    CompleteBlockFilterRule cbfr;
    cbfr = new CompleteBlockFilterRule(1,this.crearCondicionDeFiltroDeCompases()
                                       ,null,40.0,null,null,false);
    return cbfr;
}
```

La función **crearReglaCompletaDeFiltroDecompases** crea la regla utilizando la operación de filtrado de la función anterior.

La función **crearContenedorDeFiltrosDeCompases** crea el contenedor correspondiente, agrega la regla recién creada, y también el contenedor de filtros de pasajes que se creó anteriormente.

```
private CreatedRule crearReglaCompleta(){
    CreatedRule cr = new CreatedRule(this.style);
    cr.addNewFilter(this.crearContenedorDeFiltrosDeCompases());
    return cr;
}
```

La función **crearReglaCompleta** se encarga de generar el manejador de todos los contenedores y tiene como finalidad iniciar el mecanismo. Para esto se le agrega el contenedor de filtro de compases.

```
private Pair<List<Note>,List<Chord>> ConcatenarMelodias(List<Note> ml,
                                    List<Note> m2, List<Chord> basel
                                                 , List<Chord> base2){
    return ConcatRules.concatMelody(ml, basel, m2, base2);
}
```

La función **ConcatenarMelodias** permite, dadas dos melodías y sus respectivas bases, concatenarlas.

```
 private Pair<List<Note>,List<Chord>> GenerarMelodia(CreatedRule cr,
                                                     List<Note> melody,
                                                     List<Chord> base){

     return new Pair<> (cr.createMelody(base, melody),base);
 }
```

GenerarMelodia, haciendo uso de la regla completa que se creó anteriormente, comienza el proceso de filtrado y modificación para retornar el resultado.

```
@Override
public Pair<List<Note>,List<Chord>> VerifyAndCorrectImprovisation(List<Note> improvisation,
                                                                  List<Chord> base) {

    Pair<List<Note>,List<Chord>> ml = this.GenerarMelodia(this.crearReglaCompleta(),
                                                          improvisation, base);
    return this.ConcatenarMelodias(improvisation, ml.getKey(),base , ml.getValue());

}
```

Por último, se implementa la función correspondiente al mecanismo general de funcionamiento del sistema, siendo que esta será llamada durante la ejecución. En particular se devuelve como resultado la concatenación de la melodía original y el resultado de aplicar la regla.

2. Anexo 2

2.1 Análisis bottom-up del lenguaje de creación de reglas para la melodía.

2.1.1 Tipado del lenguaje.

Con el fin de expresar de forma más clara el posible funcionamiento del lenguaje se expresan como funciones y se mostraran los tipos de las definiciones con la misma sintaxis que se usa en Haskell.

2.1.2 Nivel de acción

El nivel de acción refiere al nivel más básico sobre el cual actuará el lenguaje. En este nivel se realizará un cambio efectivo, siendo que dada una nota se realizará una modificación a la altura de la nota o duración. Es importante tener en cuenta que para realizar la acción se deberá saber sobre qué nota y por cuál nota sustituir. Por lo tanto podríamos definir en un principio como sintaxis:

ModificarAlturaNota:: Altura -> Altura

ModificarDuracionNota:: Duracion->Duracion

Teniendo en cuenta que éste es el nivel más básico y que la música cuenta con un conjunto de reglas estrictas, es importante considerar que dichas modificaciones deben estar claramente orquestadas ya que no se puede superar la duración de un compás modificando duraciones. T, también es importante considerar que las modificaciones de duraciones pueden implicar el cambio de un acorde de la base en la duración de la nota, siendo que esto puede acarrear resultados indeseados.

Por otra parte, es importante ver que los casos descritos arriba son los casos más específicos, que se podrían abstraer en un único caso a nivel de nota:

ModificarNota:: Nota-> Nota

Siendo que ModificarNota interactuaría haciendo uso de las reglas de arriba.

Sería ideal que el usuario pudiera acceder a ambos niveles, teniendo la capacidad de plantear algo más exacto o menos exacto (como es el caso de disminuir el uso de la nota de blues).

Las notas a modificar se elegirán mediante el proceso de selección, mientras que la nota resultante dependerá de la propiedad que se quiera lograr, por lo que deberíamos contar con una forma de expresar dicha propiedad y distintas formas de resolver la posible nota que se utilice para sustituir en la melodía.

2.1.3 Nivel de sustitución.

Como se explicó en la parte anterior necesitaremos un proceso de selección y uno de sustitución, para saber qué nota modificar y cuál utilizaremos en la sustitución. En un principio nos centraremos en la nota que se utilizara para la sustitución, que pareciera ser lo más directo.

Para poder plantear el tipo de la función, teniendo en cuenta que se debe seleccionar una parte de la melodía que se desea modificar, donde la extensión del pasaje dependerá de la decisión del usuario, y a su vez la modificación deberá mejorar un determinado parámetro dado en la forma de la selección. Por ende esta función estará ligada a dos cosas, la melodía y la función de selección, y por último debería devolver la nota por la cual estamos sustituyendo.

Es importante entender que la función de selección debería devolver todas aquellas notas que se encuentran fuera de las características deseadas (resultado de la búsqueda), y sobre ésta se deberá iterar y aplicar modificaciones, por lo que hay que considerar que una modificación puede cambiar el parámetro de selección y

siempre se debe chequear si ya se cumple la regla antes de realizar futuras modificaciones. Por lo que tendremos una función de la siguiente forma:

Sustitucion:: Melodia -> Criterio-> Nota -> Nota

Esta sustitución si bien cuenta con los parámetros anteriores, deberá tener la capacidad de omitir alguno de ellos con el fin de aplicar de forma más específica, dada una nota particular por ejemplo, o general, como puede ser dado un grado.

2.1.4 Nivel de selección:

Con el fin de aplicar reglas es necesario la selección del pasaje que se desee modificar, siendo que deberían existir dos etapas de este proceso, una que determine el pasaje sobre el cual se estará trabajando, de forma que se cuente con distintos niveles de aplicación, y una etapa de selección de las notas en particular que se desean modificar.

En un principio la selección deberá poder trabajar sobre una melodía siendo de la siguiente forma:

SeleccionMelodia :: Melodia-> Criterio ->Melodia

De forma que dada una melodía se seleccione bajo algún criterio una sub melodía, la cual luego pueda ser nuevamente dividida por la misma función bajo otro criterio.

Luego de obtener la melodía resultado de aplicar los criterios de búsqueda, será necesario obtener todas las notas que presenten alguna característica bajo algún criterio, que en particular serán aquellas que van a ser modificadas. A esta operación le vamos a llamar búsqueda y será del siguiente tipo:

BusquedaNotas :: Melodia->Criterio-> [Notas]

2.1.5 Nivel de aplicación:

En el nivel de aplicación, el nivel más alto tratado hasta el momento, debe ser posible aplicar sobre toda la melodía un determinado criterio y no sólo sobre un pasaje o delimitación de la melodía, clara limitante del punto anterior. Por lo que será necesario que esta función permita conseguir una lista de melodías con una determinada característica y de un determinado largo. Lo que se logra con este planteo es que se puedan aplicar reglas a nivel global y considerando la totalidad de la melodía, rompiendo con la limitante existente en el sistema anterior.

BusquedaAplicacion :: Melodia -> Criterio -> Largo -> [Melodia]

Es vital comprender que la lista resultante de melodías deben ser conjuntos disjuntos, de forma que luego cuando se recomponga la línea musical no se generen inconsistencias, ya sea por la modificación de algunas notas en una melodía o la superposición de criterios de resolución distintos.

Una vez recibida la lista de melodías se deberá iterar y operar sobre esa lista las funciones descritas anteriormente en el documento.

2.1.6 Acerca de lo definido hasta el momento

Hasta este momento ha sido definido un conjunto de funciones de forma bottom-up que permiten otorgan una estructura al lenguaje, quedando pendiente la resolución de Criterio y la escritura en lenguaje natural para utilizar dichas funciones. A continuación, se presentará un breve análisis del sistema propuesto con el fin de determinar si en un principio cuenta con las capacidades deseadas y además facilite la definición de la función bajo el nombre de **Criterio**.

2.1.6.2 Análisis del sistema propuesto bottom-up.

La forma de utilización del lenguaje consistirá de aplicar los pasos de forma estructurada, ya que es una presentación mucho más natural para el ser humano en forma de “receta”.

Suponiendo que contamos con una melodía M1 creada con alguna de las reglas de generación actuales. En un principio se puede proponer algo del estilo:

BusquedaAplicacion M1 (PorcentajeNotaBlues > 60) (DosCompases)

Asumiendo el sistema implementado, y haciendo una breve interpretación de las reglas, esta función debería devolver todos los subconjuntos de dos compases de la melodía M1 que contengan una aparición de más de 60 por ciento de la nota de blues.

Las nuevas incógnitas existentes es cómo definir los pasajes de dos compases a utilizar, donde empiezan y dónde terminan, por lo que será necesario la implementación de un nuevo criterio. A su vez la nota de blues es un determinado grado sobre la base armónica, por lo que no es suficiente contar con la melodía, se debe contar con la base armónica también. Considerando esto se replantea **BusquedaAplicacion**.

BusquedaAplicacion :: Melodía-> Base -> Criterio -> Largo -> CriterioLargo

Donde Melodía y base se podrían haber simplificado en un tipo más general Música, que contenga ambas cosas. E, pero en un comienzo se utilizará el planteo separado de forma que sea más explícito.

Por otro lado, el nuevo parámetro **CriterioLargo** determinará cómo elegir los 2 compases. Ejemplos de sistemas posibles son: que divida toda la melodía en 2 compases y los analice, otro es que considere como inicio de los dos compases analizados aquello que coincida con el criterio. Otro caso posible es que itere sobre toda la melodía y devuelva la mayor cantidad de agrupaciones posibles de dos compases que cuenten con el criterio buscado.

El replanteo del llamado a la función quedaría:

BusquedaAplicacion M1 B1 (PorcentajeNotablues > 60) (DosCompases) (DivisionSimple)

Este llamado debería considerar todos los conjuntos de dos compases de la melodía excluyentes entre ellos, donde los devueltos serán aquellos que cuenten con una aparición de más del 60 por ciento de la nota del blues sobre el total de las notas.

Es posible notar que el criterio, implica una condición específica y un condicionante. Siendo entonces qué el criterio podría aclararse de la siguiente forma:

Criterio::Condición->FCondicionante->Bool

Donde se cuenta con una condición que va a tener múltiples formas específicas que intentaremos resolver más adelante.

FCondicionante podemos interpretarlo como una función que recibe el retorno de **Condición** y devuelve un booleano --se puede intuir que siempre va a recibir un entero, pero no podemos afirmar eso aún.

Una vez utilizada **BusquedaAplicacion**, y habiendo obtenido la lista de melodías que cumplen con **Criterio**, podemos utilizar la función **SeleccionMelodia**, que nos permite para cada una de las melodías realizar una nueva selección bajo el mismo criterio o nuevos criterios.

Con el fin de simplificar la escritura llamaremos **MelodiasBusquedaAplicacion :: [Melodias]** a el conjunto de melodías devueltas por la búsqueda. Por lo que se podría plantear de la siguiente forma:

SeleccionMelodia (MelodiasBusquedaAplicacion!!1) (CantidadNotas < 10)

Esta función debería devolver un pasaje que cuente con menos de 10 notas. Nuevamente se puede apreciar la falta de precisión, ya que se cuenta con un pasaje con una cantidad de N notas que se intenta reducir y no se cuenta con un criterio para determinar el nuevo largo del pasaje. Otra cosa que se puede notar, es el caso de no necesitar dicha función, ya que es posible aplicar nuevamente **busquedaAplicación**

permitiendo un resultado similar. Por lo que se descarta esta función, siendo que BusquedaAplicacion cuenta con un mayor alcance que ésta y las mismas posibilidades. Para lograr lo que se estaba planteando arriba podríamos llamar de la siguiente forma:

MelodiaFiltrada = BusquedaAplicacion (MBA !! 1) (Cantidad Notas < 10) (UnCompas) (DivisionSimple)

***MBA = Melodías Búsqueda Aplicación.**

***MF = MelodiaFiltrada**

Asumiendo entonces que no se utiliza la función SeleccionMelodia, podemos pasar a la búsqueda de las notas.

Notas = BusquedaNotas (MF!!1) (GradoNota == 5)

El resultado de aplicar la función de arriba debería ser todas las notas de grado 5 con respecto a la base, por lo que es necesario pasar la base correspondiente a **BusquedaNotas** con el fin de poder determinar los grados de las notas en la melodía.

Por lo que la nueva definición quedaría con la siguiente forma:

BusquedaNotas:: Melodía-> Base -> Criterio

Y que el llamado quedaría de la forma:

Notas = BusquedaNotas (MF!!1) B1 (GradoNota == 5)

Por último, faltaría la modificación correspondiente a las notas, para ello se deberá iterar las notas resultado y operar sobre ellas. Para eso utilizaremos la función **ModificarNota** definida anteriormente.

ModificarNota (Notas!!1)

Siendo que esta definición se encuentra incompleta, ya que le falta conocer la melodía y el criterio de sustitución. Por lo que la nueva definición quedará con la siguiente forma:

ModificarNota :: Melodía -> Nota -> Criterio

Nuevamente hace falta conocer la base armónica para poder operar y una función para determinar la modificación, por lo que utilizaremos una función del tipo sustitución, quedando la definición de la siguiente forma:

ModificarNota:: Melodía-> Base->Nota->Criterio -> f::Sustitucion

Aplicando la nueva definición:

ModificarNota (MF!!1) B1 (Notas!!1) (CantidadNotas<10) (DividirEnCorcheas)

Obteniendo como resultado la división en corcheas de la nota en grado 5 que se encuentre en el pasaje de 1 compás.

Resumiendo las nuevas reglas con sus modificaciones correspondientes quedarían de la forma:

BusquedaAplicacion:: Melodía->Base->Criterio->Largo->CriterioLargo

Criterio::Condición->FCondicionante->Bool

BusquedaNotas :: Melodía-> Base -> Criterio

ModificarNota:: Melodía-> Base->Nota->Criterio -> Sustitucion

Sustitucion:: Melodia -> Criterio-> Nota -> Nota

ModificarAlturaNota:: Altura -> Altura

ModificarDuracionNota:: Duracion->Duracion

2.1.7 Definición de script en Lenguaje Natural.

Mediante lo definido en etapas anteriores del documento se puede hacer una primera aproximación a una sintaxis que permita la elaboración de reglas. Con el fin de facilitar la identificación de variables y operaciones se utilizarán la primera letra mayúscula como identificador de las funciones del script.

Primero que nada, es necesario contar con la igualdad para la generación de referencias, ya que durante el desarrollo se pudo notar el nivel de importancia de las mismas. El trabajo se interpretará como variables toda aquella palabra que se encuentre escrita con minúscula.

m1 = Musica.

Utilizaremos el término Música para referirnos a la línea melódica y base armónica que recibirá la regla, ya que es una regla de verificar y corregir; y el punto para indicar el fin de línea.

Utilizaremos la operación Melodía para obtener la línea melódica de un conjunto base armónica y melodía. Dicha operación se aplicará sobre el parámetro entre paréntesis luego de la palabra clave De. Un ejemplo de uso

melodia1 = Melodía De (m1)

Siendo que en este caso en particular se estaría obteniendo la línea melódica de m1 y asignándose a una nueva variable melodia1.

Otra operación importante es la de obtener la base armónica de un conjunto base armónica y melodía. Dicha operación se identificará mediante el nombre Base y consiguiente el operador De, siendo que va a contar con un uso análogo al de la melodía. Por ejemplo:

base1 = Base De (m1)

Planteado ya lo básico se va a recapitular las definiciones realizadas hasta el momento. Como regla general se cuenta con el operador igual para asignar variables, las cuales tienen un rol fundamental en el script. A su vez se ha definido que el script

va a funcionar como Case Sensitive, siendo que las variables contaran con su primer letra en minúscula y las palabras clave del script utilizaran su primer letra en mayúscula.

En una siguiente etapa se va a contar con el planteo de las reglas ya definidas como necesarias anteriormente en este documento. Primero que nada, vamos a plantear cómo se utilizará el proceso de selección. Es importante recordar que el proceso de selección se encargaba de obtener los tramos de música correspondiente a un determinado criterio, siendo que el resultado de aplicar esta operación puede devolver 0 o más elementos.

Para referir este paso se utilizará múltiples palabras claves, quedando de la forma:

pasajes = Selección De Pasajes (musica) Con Criterio (criterio) De Largo (largo) Con Criterio De Largo (criterioLargo).

Para hacer uso de esta operación es necesario la asignación a una variable, de forma que dicha variable representa una colección de una cantidad indefinida de elementos. Teniendo en cuenta que el número de elementos de la colección es desconocido para el usuario se permitirá trabajar sobre los distintos elementos de forma explícita o implícita, usando var[n] como forma de referirse a un elemento particular, siendo que si dicho elemento no existe no se aplicará ningún cambio, y si existe se aplicaran las consecuentes reglas de forma efectiva. La otra posibilidad es el trabajo sobre todos los elementos de la colección por igual al referirse a la variable sin la notación presentada anterior, es decir var.

La siguiente operación que queda por definir es **BusquedaDeNotas**, a la cual se le reserva la siguiente notación.

Busqueda Notas A Modificar (musica) Con Criterio (criterio)

Por último la operación de modificación:

Modificar (Musica) Cambiando (nota) Cumpliendo Con Criterio (criterio) Sustituyendo Por (FSustitucion)

En un principio este planteo debería poder permitirnos armar un esqueleto de funcionamiento del script dejando algunas interrogantes.

Teniendo en cuenta lo ya definido podemos realizar una prueba de funcionamiento:

1. **musica = Musica.**
2. **pasajes = Selección De Pasajes (musica) Con Criterio (Nota De Blues > 60%) De Largo (2 Compases) Con Criterio De Largo (Cambio De Compás).**

El planteo anterior se presenta como un lenguaje intermedio, siendo en definitiva una primera aproximación al lenguaje que se desea plantear. En la práctica el lenguaje debería separarse en 3 ramas, trabajar sobre la base, trabajar sobre la melodía y, trabajar sobre la métrica, siendo que cada regla creada se aplicará sobre una de las ramas. Aunque a fines de esta tesis nuestro interés se centra únicamente sobre la modificación de la melodía.

Las reglas creadas se definirán como un conjunto de reglas sobre sí mismas, siendo el caso que la melodía se modifica directamente, no hace falta utilizar una variable para referirse a ninguna parte de la melodía. También es importante reconocer, aunque no se ha tenido en cuenta aun, que son mucho más relevantes las alturas relativas de las notas, así como secuencias a la hora de la creación de las reglas; por ejemplo, modificar cada 5a con respecto a la base, de esta forma permitiendo una mayor variabilidad a las reglas posibles de creación.

3. Anexo 3

3.1 Extracción de datos durante la revisión

Nombre del artículo	Splicing music composition
Autor	Clelia De Felice, Roberto De Prisco, Delfina Malandrino, Gianluca Zaccagnino, Rocco Zaccagnino∗ , Rosalba Zizza
Jornal o conferencia	Information Science
Año de publicación	2017
Resumen	Sistema implementado con splicing, inspirado en la recombinación del adn a nivel molecular, que permite la composición de corales de 4 voces de forma automática. El documento plantea la existencia de 5 formas diferentes de composición: Algoritmos evolutivos, algoritmos inspirados biológicamente, gramáticas formales, autómatas celulares y machine learning; Destacando que en este caso en particular se trata de un algoritmo inspirado biológicamente.

	Los sistemas de splicing representan el proceso de recombinación de moléculas de ADN. El sistema implementado extrae información de un conjunto inicial (corales de bach), y un conjunto de reglas bien establecidas obtenidas por la extracción de información sobre notas acordes y tonalidades. Para este sistema particular se utiliza una representación de notas considerando aspectos de la música como tonalidad y grado de cada nota, a esta representación le llaman representación de grado-tonalidad, donde afirman que el resultado es mucho más parecido a un resultado humano y de mayor calidad. El algoritmo genético consiste en la generación de una población y la evaluación de los miembros de la población para determinar cuál es mejor y cuáles deben ser reemplazados. La operación encargada de evaluar fue realizada por una red neuronal entrenada por los gustos del autor.
Método utilizado para la composición	Splicing (Método de inspiración biológica)

Nivel de satisfacción reportado	Muy alto. La prueba se realizó con expertos.
Evaluación del resultado	N/A

Nombre del artículo	GenJam: A Genetic Algorithm for Generating Jazz Solos
Autor	J.A Biles
Jornal o conferencia	Proceeding of International Computer Music Conference
Año de publicación	1994
Resumen	GenJam se implementa en con el fin de generar mediante algoritmos genéticos una improvisación de jazz de nivel de un aprendiz sobre una secuencia fija utilizada de base. Para su funcionamiento GenJam lee un archivo de progresiones que provee el tempo y el estilo rítmico, el número de solos en la obra y la progresión armónica. Al momento de la improvisación existe un mentor que se encarga de introducir si el resultado de la improvisación hasta el momento es bueno o no. El sistema

	trabaja sobre una representación binaria.
Método utilizado para la composición	Splicing (Metodo de inspiración biológica)
Nivel de satisfacción reportado	NA
Evaluación del resultado	N/A

Nombre del artículo	Automatic melody composition based on a probabilistic model of music style and harmonic rules
Autor	Carles Roig, Lorenzo J.Radon, Isabel Barbancho, Ana MBarbancho
Jornal o conferencia	Knowledge-Based Systems
Año de publicación	2014
Resumen	El sistema basa su funcionamiento en un modelo probabilístico para la caracterización de música aprendida de ejemplos. El modelo utiliza parámetros abstraídos de forma automática. En particular aprende de patrones rítmicos y tonales para la caracterización de los diferentes estilos musicales, para luego reproducirlos.
Método utilizado para la composición	Modelo probabilístico
Nivel de satisfacción reportado	Alto, los resultados no son distinguibles de composiciones humanas en un 70 por ciento.
Evaluación del resultado	N/A

Nombre del artículo	Automatic Music Composition using Answer Set Programming
Autor	Geog Boenn, Martin Brain, Marina de Vos, John FFitch
Jornal o conferencia	Theory and Practice of Logic Programming
Año de publicación	2009
Resumen	El sistema desarrollado se hace bajo el paradigma answer set programming, donde se determina los requerimientos que deben ser contemplados en una solución a un cierto problema. El sistema es capaz de componer la melodía, el ritmo y la base armónica, así como corregir errores en composiciones dadas.
Método utilizado para la composición	Answer set programming (paradigma logico)
Nivel de satisfacción reportado	Medio, el sistema genera música válida pero poco atractiva por los patrones rítmicos utilizados.
Evaluación del resultado	N/A

Nombre del artículo	Chopin or not? A memetic approach to music composition
Autor	Jacek Mandziuk, Marcin Goss , Aleksandra Wozniczko
Jornal o conferencia	Conference
Año de publicación	2013
Resumen	El objetivo de este estudio, referido en [3], es crear composiciones que tengan sentido razonable. Mediante un algoritmo memético, que es una combinación de genético y optimización local en la generación, se pretende generar composiciones originales basadas en obras de Chopin. La optimización local, en términos de algoritmos genéticos, es una mutación que no es aleatoria sino que se corresponde con un conjunto predefinido de reglas producto del conocimiento del dominio. El algoritmo, luego de un número predefinido de generaciones, da como resultado una lista con los “especímenes” más aptos, que surgieron mediante la selección elitista o mutación y fueron probados con una función de selección específica

	que se nutre de conceptos musicales tales como la tonalidad, el tempo, etc. El estudio genera piezas musicales mediante el algoritmo mencionado que luego son evaluadas por humanos en busca de discriminar cuáles presentan características notorias que puedan sugerir que fueron creadas por un programa y cuáles aparentan haber sido creadas por un humano. El estudio concluye que hubo composiciones que lograron ser catalogadas como creaciones humanas pese a haber sido creadas artificialmente.
Método utilizado para la composición	Algoritmo memético.
Nivel de satisfacción reportado	Na
Evaluación del resultado	N/A

Nombre del artículo	Memetic music composition
Autor	Jos M Cadenas, Yew Soon Ong, Giovanny Acampora

Jornal o conferencia	Transaction on Evolutionary Computation
Año de publicación	2014
Resumen	El sistema consiste en la utilización de un algoritmo memético para la composición musical. El sistema se compone de varios agentes con dos pasos principales. El primer paso consiste en brindarle a una población de los agentes una base armónica de forma que trabajen paralelamente de forma cooperativa y adaptativa para encontrar un conjunto posible de melodías para esa línea de bajo. En la segunda etapa, otro grupo de agentes se encarga de analizar el resultado de la primera parte e identificar la línea melódica que mejor complemente el bajo.
Método utilizado para la composición	Algoritmo memético.
Nivel de satisfacción reportado	Na
Evaluación del resultado	N/A

Nombre del artículo	Automatic composition of happy melodies based on relations
Autor	Xizheng Cao, Lin Sun, Jingwen Niu, Ruiqi Wu, Yanmei Liu, Huijuan Cai
Jornal o conferencia	Business Media New York
Año de publicación	2014
Resumen	El sistema utiliza estructuras jerárquicas y sus relaciones como medio para la composición. A una estructura base seleccionada por el usuario se le aplican una serie de transformaciones siempre teniendo en cuenta las reglas de composición correspondientes de forma de lograr una nueva melodía.
Método utilizado para la composición	Algoritmo experto basado en reglas.
Nivel de satisfacción reportado	Medio bajo. El público utilizado para probar la efectividad del algoritmo evaluó de forma negativa la composición por la computadora.
Evaluación del resultado	N/A

Nombre del artículo	Intelligent Music Composition
Autor	Kaliakatsos-Papakostas, Maximo A, Floros, Andres, vrahatis, Michael N
Jornal o conferencia	Swarm Intelligence and bio-Inspired Computation.
Año de publicación	2103
Resumen	El documento presenta 3 formas de composición inteligente. La primera, composición inteligente sin supervisión, consiste en que la inteligencia se exprese a través de simples reglas que permitan producir un resultado complejo e impredecible. La composición inteligente supervisada consiste en algoritmos inteligentes que se utilicen para modificar los parámetros de composición del sistema de forma que sea capaz de componer música que conforme un criterio predefinido. Por último, la composición interactiva inteligente consiste en que el sistema conozca la preferencia del usuario y se adapte a las mismas. El artículo procede a profundizar en las distintas metodologías y las formas de implementación posibles, ya sea

	mediante algoritmos evolutivos, genéticos o de inteligencia artificial.
Método utilizado para la composición	Algoritmo experto basado en reglas.
Nivel de satisfacción reportado	Medio bajo. El público utilizado para probar la efectividad del algoritmo evaluó de forma negativa la composición por la computadora.
Evaluación del resultado	N/A

Nombre del artículo	Automatic Music Composition Based on CounterPoint and Imitation using Stochastic Models
Autor	Tsubasa Tanaka, Takuya nishimoto, Nobutaka Ono, shigeki Sagayama
Jornal o conferencia	Swarm Intelligence and bio-Inspired Computation.
Año de publicación	2103
Resumen	El método propuesto por este documento consiste en la composición basada en métodos estocásticos como son el contrapunto y la imitación. Con el fin de generar las piezas de contrapunto, se define qué es apropiado y qué no para el estilo de música en cuestión. Para esto se crea un juego de probabilidades asociados a las notas que podría elegir un compositor humano. Para la composición por imitación se plantean tres pasos. El primer paso es obtener un tema, que puede ser de una melodía existente o compuesto específicamente para el sistema. El segundo paso es planear la estructura de imitaciones y cadencias.

	Por último, se selecciona un conjunto de notas concreto para la pieza.
Método utilizado para la composición	Algoritmo basado en cadenas de markov.
Nivel de satisfacción reportado	Se plantea el modelo pero no se implementa.
Evaluación del resultado	N/A

Nombre del artículo	Gp-Music: an interactive Genetic Programming System for Music Generation with Automated fitness Raters
Autor	Brad Johanson, Riccardo Poli
Jornal o conferencia	Technical Report, School of computer Science.
Año de publicación	-
Resumen	El sistema presentado consiste de un sistema de composición basado en algoritmos evolutivos con asistencia del usuario, el cual tiene como finalidad evolucionar pequeñas melodías.
Método utilizado para la composición	Algoritmo evolutivo de 4 capas
Nivel de satisfacción reportado	Se plantea el modelo pero no se implementa.
Evaluación del resultado	N/A

Nombre del artículo	Open music
Autor	-
Jornal o conferencia	-
Año de publicación	-
Resumen	El sistema presentado consiste en un programador visual de música, permitiendo manejar las estructuras musicales, acordes y notas de forma visual. Su función principal es la de facilitar la creación de música durante la composición, así como un conjunto de herramientas visuales para el análisis matemático asistido por computadora de la música, proceso y síntesis de sonido.
Método utilizado para la composición	Asistencia mediante análisis matemático.
Nivel de satisfacción reportado	Se presenta la implementación pero no el modelo
Evaluación del resultado	N/A

Nombre del artículo	Common music
Autor	-
Jornal o conferencia	-
Año de publicación	-
Resumen	Common Music es un sistema de composición que transforma representaciones algorítmicas de procesos musicales y estructuras en distintas variedades de protocolos de control para la síntesis del sonido y visualización. Su principal programa a nivel de usuario es Grace, una plataforma de “drag and drop”. En Grace los algoritmos musicales se pueden ejecutar en tiempo real, o más rápido que tiempo real cuando se están haciendo composiciones basadas en archivos. El sistema desarrollado permite el uso de funciones de alto orden de forma visual.
Método utilizado para la composición	Asistente de composición algorítmica basada en un lenguaje específico.
Nivel de satisfacción reportado	Se presenta la implementación.

Evaluación del resultado	N/A

Nombre del artículo	Alda: A text based music composition language
Autor	-
Jornal o conferencia	-
Año de publicación	-
Resumen	Alda es un sistema presentado con el fin de escribir música en forma de texto, con la adición de ciertas estructuras de control que facilitan la codificación de la música.
Método utilizado para la composición	Composición asistida mediante el lenguaje de escritura de música.
Nivel de satisfacción reportado	Se presenta la implementación.
Evaluación del resultado	N/A