

Reproductor de canciones DTXMANIA para entornos libres

Guillermo Eduardo Cabezas Castillo

27 de abril de 2017

Índice general

1. Introducción	13
1.1. El proyecto DTXMANIA en el contexto actual	13
1.1.1. Historia de los principales juegos musicales	14
1.1.2. Historia. Composición musical mediante ordenador	24
1.1.3. Baterías-MIDI y proyectos en entornos libre	39
1.1.4. Actualidad de Reproductores DTXMania en entornos libres	45
1.1.5. Raspberry	47
2. Proyecto de canciones DTXMANIA entorno libre	58
2.1. Planteamiento general	58
2.1.1. Comparaciones funcionales Stepmania/DTXMania	61
2.1.2. pydance	63
2.1.3. DTXMania	65
2.2. Formatos a nivel básico de las canciones en juegos musicales	68
2.2.1. FoFiX / PhaseShift	68
2.2.2. MIDI	70

<i>ÍNDICE GENERAL</i>	2
2.2.3. pyDance	72
2.2.4. DTXMania	74
3. Posibles configuraciones en el DTXMania	78
3.1. Configuraciones de Interfaz a implementar	79
3.1.1. DARK	79
3.1.2. Autoplay	79
3.1.3. ScrollSpeed	79
3.1.4. Tight	79
3.2. Configuraciones de Interfaz interesantes para un futuro	80
3.2.1. InputAdjust	80
3.2.2. PreviewSoundWait	80
3.2.3. Fullscreen	80
3.2.4. Sudden	80
3.2.5. Hiden	80
3.3. Configuraciones de Interfaz que se escapan al proyecto	81
3.3.1. ComboPosition	81
3.3.2. Reverse	81
3.3.3. Position	81
3.3.4. Guitar/Bass	81
3.4. Configuraciones de uso a implementar	81
3.4.1. Drums Key Assign	81

<i>ÍNDICE GENERAL</i>	3
3.4.2. HH Group	82
3.4.3. HH Priority	82
3.4.4. FT Group	82
3.4.5. FT Priority	83
3.4.6. CY Group	83
3.4.7. CY Priority	84
3.5. Configuraciones de uso para un futuro	84
3.5.1. BGA	84
3.5.2. FillIn	84
3.5.3. DebugInfo	84
3.5.4. HitSound	85
3.5.5. SaveScore	85
3.5.6. D-MinCombo	85
3.6. Configuraciones de uso que no se implementaran	85
3.6.1. AVI	85
3.6.2. Guitar/Bass	85
3.6.3. StageFailed	85
3.6.4. BGMSound	86
3.7. Otras configuraciones	86
4. Formato xa	88
4.1. Problemas	89

4.2. Solución aportada al proyecto	90
5. Formato DTXMania al completo	92
5.1. Estructura básica	92
5.2. Cabecera	93
5.3. Comandos en cabecera a implementar	93
5.4. Comandos en cabecera para un futuro	95
5.5. Comandos en cabecera que no se implementaran	99
5.6. Objetos	100
5.7. Canales definidos por el DTXMania	103
5.8. Canales a implementar	103
5.8.1. 01 - BGM Back chorus	103
5.8.2. 03 - BPM	103
5.8.3. 08 - Extended tempo(BPM)	104
5.8.4. 11-1A - Notas de la batería	104
5.8.5. 61-62- Sonidos en autoplay	105
5.9. Canales a implementar en un futuro	105
5.9.1. 02 - BAR length	105
5.9.2. 04 - BGA	106
5.9.3. 07 - BGA/ 2 capa	106
5.9.4. 31-3A - Drums invible object	106
5.9.5. 50 - BAR line	107

5.9.6.	51 - Beat Line	107
5.9.7.	53 - Fill in	107
5.9.8.	55-59,60 - BGA / capas	107
5.9.9.	63-69, 70-79,80-89,90-92 - Sonidos en autoplay	107
5.9.10.	B1-B9, BC No-chip default sound	108
5.9.11.	C1 [BEAT-line shifting]	108
5.9.12.	C2 [hide BEAT/BAR lines]	108
5.9.13.	C4, C7, D5-D9, E0 [Swapping BGA bitmap]	108
5.10.	Canales que no se implementaran	109
5.10.1.	00	109
5.10.2.	05 - Extend object	109
5.10.3.	06 - Cambiar la animación de MISS	109
5.10.4.	09-10 - Reservados por BMS	109
5.10.5.	20-29 y 2F- Notas de la guitarra	109
5.10.6.	30 - flow-speed drums	109
5.10.7.	40 - Reservado por BMS	109
5.10.8.	41-46 Invisible object for the 2nd player	110
5.10.9.	47-49 - Reservado por BMS	110
5.10.10.	52 - MIDI drum chorus	110
5.10.11.	54 - Playback movie	110
5.10.12.	93-99 - Reservados	110

5.10.13.A0-A7 - Bass notes	110
5.10.14.A8 - Wailing (Bass)	110
5.10.15.AF - Wailing sound (Bass)	110
5.10.16.BA - No chip default sound (guitar)	111
5.10.17.BB - No chip default sound (bass)	111
5.11. Comentarios sobre el proyecto	111
6. Especificaciones generales de diseño	112
6.1. Consideraciones generales a python3	112
6.1.1. Posibles librerías gráficas y de sonido: pygame / tkinter / Qt5	113
6.1.2. Configuraciones en disco duro: configparser	114
6.1.3. MIDI: pygame / Cython y rtmidi2	114
6.2. Gráficos en qt5	115
6.3. Animaciones en qt5	115
6.3.1. Ningún timer, una gran animación	116
6.3.2. Todas las notas con su timer	116
6.3.3. Un timer para el siguiente compás	117
6.3.4. Un timer único para el control del tiempo	117
6.4. Sonidos en qt5	118
7. pyDTX	119
7.1. Ideas generales	119
7.2. Pantalla principal	119

7.3. Pantalla Inicial	120
7.4. Pantalla configuración batería-MIDI	120
7.4.1. MIDI	122
7.4.2. Configuración MIDI en pyDTX	122
7.4.3. Baterías-MIDI y Lanes	123
7.5. Pantalla configuración pyDTX	126
7.6. Pantalla Navegación	127
7.7. Reproducción de la canción	131
7.7.1. Inicialización básica y detalles de Qt5	131
7.7.2. Inicialización con una canción	131
7.7.3. Teclado	131
7.7.4. MIDI	132
7.7.5. Lanes	133
7.7.6. Ideas básicas de los lanes	133
8. Detalles de la implementación	140
8.1. Transformar .dtx a .dtx libre	140
8.2. DRUM-HAT MIDI	140
8.3. pyDTX	141
8.4. Instalación	141
8.4.1. Fedora 25 - x86 64bits	141
8.4.2. Raspberry3 - armv7	142

8.4.3. Conseguir el código fuente	142
8.4.4. Problemas encontrados con .dtx de internet	143
8.4.5. Comentarios finales en la Raspberry	144
8.4.6. Comentarios finales al proyecto	144
9. Posibles mejoras en el futuro	146
A. Código fuente	148
A.1. Transformar .dtx a .dtx libre	148
A.2. DRUMHAT en instrumento midi	148
A.3. pyDTX	150
A.3.1. ChipAuto.py	150
A.3.2. Chip.py	150
A.3.3. LaneAuto.py	152
A.3.4. LaneVisual.py	153
A.3.5. main.py	155
A.3.6. UtilidadesDTX.py	156
A.3.7. MyMainWindowQt5.py	160
A.3.8. MyWidgetConfigurarDrum.py	162
A.3.9. MyWidgetConfigurarDTX.py	167
A.3.10. MyWidgetNavegacion.py	169
A.3.11. MyWidgetPuntuacion.py	175
A.3.12. ReproductorDTX.py	178

B. Guía de estilo python 190

B.1. Formateo del código	191
B.1.1. Indentación	191
B.1.2. Tamaño máximo de línea	191
B.1.3. Líneas en blanco	191
B.1.4. Codificación de caracteres	191
B.1.5. Imports	191
B.1.6. Espacios en blanco en expresiones y sentencias	192
B.1.7. Otras Recomendaciones	193
B.2. Comentarios	195
B.2.1. Comentarios de bloque	195
B.2.2. Comentarios en línea	195
B.3. Cadenas de Documentación	196
B.4. Convenciones de nombres	196
B.4.1. Descriptivo: Estilos de Nombrado	196
B.4.2. Prescriptivo: Convenciones de Nombrado	198
B.5. Recomendaciones para la programación	201

C. GNU GENERAL PUBLIC LICENSE 205

Glosario 206

D. Bibliografía 207

Índice de figuras

1.1. Controlador del Guitar Hero	14
1.2. Interfaz del Guitar Hero	15
1.3. Batería de Rock Band	16
1.4. Batería de Rock Band Pro	16
1.5. Interfaz del DDR	19
1.6. Interfaz del Drum Mania	21
1.7. Mueble arcade del Drum Mania	21
1.8. Mueble arcade del Gitadora	22
1.9. Conector clásico MIDI	34
1.10. Ejemplo de batería electrónica de 4 pads basada en Arduino	41
1.11. Yamaha DD-65	42
1.12. Roland V-Drums Lite HD-3	43
1.13. Roland TD-30	44
1.14. Raspberry Pi 3 B	51
1.15. Piano HAT	54

<i>ÍNDICE DE FIGURAS</i>	11
1.16. Drum HAT	55
3.1. Grupo HH	82
3.2. Grupo FT	83
3.3. Grupo CY	84
5.1. Explicación lista objetos	101
5.2. Explicación varias lista objetos	102
5.3. Explicación barra reducida	106
7.1. Pantalla Inicial	120
7.2. Widget Configurar Drum	121
7.3. Widget Configurar DTX	127
7.4. Widget Navegacion	128
7.5. Widget Puntuación	137

Agradecimientos

*Dedicado a
mi familia*

Capítulo 1

Introducción

El objetivo de este proyecto es un reproductor de canciones DTXMANIA, en entornos de código abierto. Para ello se tendrá que implementar la idea de como es este formato, y que es necesario hacer para poder reproducirlo con algunos de sus añadidos.

Como primera toma de contacto se puede decir que el formato DTXMANIA, es un formato musical, especializado en baterías.

1.1. El proyecto DTXMANIA en el contexto actual

El origen del formato DTXMANIA proviene de una serie de juegos musicales desarrollados en Japón. Cuando los usuarios de dichos juegos que tenían conocimientos prácticos del uso de instrumentos de percusión (en concreto baterías) los probaron, se dieron cuenta de que con una serie de modificaciones adicionales dichos juegos podían ser un apoyo eficiente para mejorar su técnica habitual. De aquí surge la idea de desarrollo de este proyecto en el que se propone y se lleva a cabo un entorno basado en software libre para posibilitar el uso de dicha práctica.

Debido al auge que han tenido este tipo de juegos a partir del famoso Guitar Hero, con la finalidad de mostrar el amplio abanico de juegos e ideas existentes en el mercado, se hace necesario exponer un conjunto representativo de los principales juegos de tipo musical que disponen de un componente físico para simular instrumentos.

En consecuencia a la gran diversidad de entornos y sistemas existentes también se hace necesario efectuar un pequeño repaso a los formatos musicales digitales que dan soporte a dichos desarrollos ya que van a estar altamente integrados en este proyecto.

1.1.1. Historia de los principales juegos musicales

Cuando se piensa en un juego/simulador principal lo que primero puede venir a la mente, por ser lo más conocido es el caso del Guitar Hero, aunque es el más conocido también se tiene que hacer un repaso a otros juegos de corte musical para poder entender la evolución de este tipo de juegos y más concretamente los que tienen como instrumento principal la batería ya que sera uno de los principales objetivos del proyecto.

Guitar Hero

Guitar Hero es una popular franquicia de videojuegos musicales.

La serie Guitar Hero es conocida por el uso de un dispositivo con forma de guitarra que se utiliza como control de juego para simular y hacer música con ella, representando notas de colores en la pantalla que corresponde a cada uno de los botones del controlador. Los juegos de la serie permiten tanto partidas individuales como multijugador, pudiendo estas últimas ser cooperativas o competitivas. La serie ha utilizado una amplia gama de canciones de rock de diversas épocas, tanto licenciadas como independientes, hechas desde los años 60 hasta el presente, muchas de ellas son de bandas muy exitosas.

La serie ha vendido alrededor de 15 millones de juegos, ganando aproximadamente 1000 millones de dólares.

Guitar Hero es un juego inusual debido a que viene con un control que recrea a una “Gibson Les Paul” en lugar de los controles comunes. Jugar con el controlador en forma de guitarra simula a una guitarra verdadera, a excepción de los “Botones Traste” y la “Barra de Sonido” en lugar de los Trastes y las seis cuerdas. El desarrollo de Guitar Hero fue inspirado en el juego Guitar Freaks de Konami, que en esos tiempos no era muy popular en Estados Unidos.



Figura 1.1: Controlador del Guitar Hero

Su control básicamente son unos Chips que van bajando por la pantalla y se tiene que sincronizar el pulsar el botón asociado de la guitarra simulada cuando llegan a la parte más baja. Estos chips están sincronizados con la canción.



Figura 1.2: Interfaz del Guitar Hero

Comentar que hay una barra de vida y esta baja o sube según vamos dando a las notas más correctamente o menos. Si la vida bajara mucho se para la canción y se vuelve al menú principal.

Rock Band

Los productores del juego Guitar Hero después de lanzar varias versiones de éste, constataron que el empuje inicial de la dinámica de juego se estaba quedando obsoleto y que si deseaban continuar debían de aportar nuevas ideas al producto. Es entonces cuando deciden lanzar al mercado el Rock Band.

Rock Band es una serie de videojuegos de música en los cuales permite que hasta cuatro jugadores puedan jugar a tocar algunas de las canciones más populares de música Rock, jugando con los instrumentos de la empresa(o también con los instrumentos de Guitar Hero). Los jugadores pueden jugar con una guitarra, un bajo, una batería y cantando con un micrófono, además de el teclado (desde el Rock Band 3)

Exceptuando el control de micrófono, la idea básica continua siendo la misma, simular en mejor

o peor medida los instrumentos antes citados, con el sistema ya creado en el Guitar Hero.

La primera batería que salió a la venta para el Rock Band en principio contaba solamente con cuatro Pads y un pedal.



Figura 1.3: Batería de Rock Band

Más adelante con el juego Rock Band Pro, ya salió la posibilidad de una batería con siete Pads y un pedal, evidentemente con la batería avanzada se podía jugar con las canciones de la batería básica.



Figura 1.4: Batería de Rock Band Pro

Uno de los problemas que tuvo la gente al encontrarse con estas canciones con un nuevo formato,

es que la compañía solamente dio la posibilidad de actualizar las canciones antiguas al nuevo formato pagando.

Frets on fire

Debido al auge de estos juegos musicales no se tardó en sacar una versión libre de estos, el más famoso de ellos fue el Frets on fire, que copiaba el juego Guitar Hero. De modo que el jugador simula el acto de tocar una guitarra. Las notas aparecen en la pantalla sincronizadas con la canción, y son tocadas manteniendo presionadas las teclas correctas y marcándolas pulsando Enter en el momento preciso. El punteo en las notas correctas incrementa el coeficiente por el que se multiplican los puntos ganados al tocar (x2, x3 y hasta x4), pero una sola nota incorrecta hace que este coeficiente vuelva a x1. Si bien no hay ningún objetivo establecido para las canciones, la puntuación obtenida puede ser comparada con la del resto de jugadores en la web oficial.

El carácter más distintivo del juego es la forma de controlarlo: el teclado se coge con ambas manos como si de una guitarra se tratase, con la mano izquierda en los botones F1-F5 y la derecha en el botón Enter. Debido que algunos teclados no reconocen la pulsación simultánea de algunas de estas teclas, el juego permite cambiarlas para evitar este problema.

Su fecha de salida fue el 3 de agosto de 2006. Un detalle muy importante es que de inicio ya incorporaba el poder agregar “mods” para maximizar su funcionalidad y modificar su apariencia física según los gustos del jugador.

FoFiX

Un mod muy famoso del Frets on Fire fue el FoFix fue una versión libre del Rock Band.

Estos juegos al ser libres todo su código se puede mirar y modificar. Aunque mayoritariamente fueron detrás de sus originales en cuanto a las modificaciones que se programaban. Si el Guitar Hero insertaba un modo donde las canciones eran más difíciles al cabo de un tiempo en el Frets on fire se programaba un mod para ello. Un punto a favor de estos juegos es que si se tenía el juego original se podía copiar las canciones de estos juegos y además disponer de las propias que la comunidad creaba para ellos. Evitando de esta manera tener solo una única tienda donde estaba todo el control sobre las canciones que se creaban y sus precios.

Dance Dance Revolution

Dance Dance Revolution es la serie pionera del género de simuladores de baile en los video juegos. Los jugadores se colocan sobre una “plataforma de baile”, también llamada “pista” y presionan con los pies las flechas dispuestas en forma de cruz sobre esta pista, siguiendo el ritmo de la música y el patrón visual que aparece en la pantalla. De acuerdo a la sincronización que los jugadores muestren con el ritmo y lo bien que sigan el patrón de flechas, se les otorga una puntuación (que varía de acuerdo a la versión del juego). Su primer lanzamiento, en forma de arcade, fue realizado en Japón en 1998. Desde su creación han sido producidas múltiples variaciones, incluyendo algunas para uso casero. Es clasificado como un juego Bemani (Bemani es una abreviación japonesa de la palabra Beat Mania, el nombre del primer juego musical de Konami), cuyo creador es Yoshihiko Ota, que también participó en otras series como “pop’n music”.

En la composición de canciones para el juego participan varios artistas, incluyendo los que aparecen en la serie de discos Dance Mania, otros de Konami, como Yasuhiro Taguchi, principal compositor musical para esta serie de juegos desde DDR X y pocos, como Naoki Maeda, fundador de la serie, que se retiraron de Konami. Algunos artistas han dado letras de sus canciones como es el caso de Aerosmith que donó la canción “I Don’t Want to Miss a Thing” para el Dance Dance Revolution Extra Mix que también está disponible en versión para PlayStation.

Durante un juego normal, 4 flechas direccionales (u 8 en Double) permanecen estáticas en la parte superior de la pantalla. Otras flechas se desplazan desde la parte inferior de la pantalla hasta las flechas estáticas en la parte superior (se cambia de posición en caso de que el jugador opte por Reversa). Un jugador debe bailar con la canción, pisando la superficie correspondiente a las flechas marcadas en la pantalla en una plataforma de baile de metal y acrílico. En algunas ocasiones dos o más flechas deben ser pulsadas al mismo tiempo, lo que obliga al jugador a saltar para pulsar 2 flechas a la vez o agacharse para pulsarlas con las manos o rodillas, en caso de 3 o 4 flechas consecutivas, estas últimas aparecen en los edits. De esta manera, el juego invita al jugador a bailar según la coreografía pre-establecida para cada canción. Por otro lado se encuentran las Freeze Arrow (implementadas desde DDRMAX) que aparecen flechas en las que hay que mantener presionado el botón correspondiente durante el tiempo establecido. A veces, ciertas canciones tienen pausas de tiempo, sobre todo, antes de la flecha o al empezar el freeze arrow, el jugador debe estar más atento a la secuencia. En algunas ocasiones dos o más Freeze Arrows deben ser pulsadas y mantenidas al mismo tiempo, ya que levantando el pie durante el Freeze Arrow cuenta como una flecha perdida. Y finalmente, las Shock Arrows (implementadas desde DDR X) que solo aparecen en unas determinadas canciones, generalmente en Challenge, y consisten en que aparecen las 4 u 8 flechas al mismo tiempo, en un color metalizado y con unos relámpagos por dentro. La misión del jugador es esquivarlas, ya que de pulsarlas, cuenta como flecha perdida, frena el combo y desaparece las flechas por una fracción de segundo. Este tipo de paso recuerda y mucho a las populares Minas de la serie de juegos musicales “In the Groove”, de la cual Konami tiene los derechos tras su juicio con Roxor Games. Si se falla repetidamente las flechas (o pisando una Shock Arrow), la barra de vida se ve drásticamente reducida y se puede llegar a fallar la canción.



Figura 1.5: Interfaz del DDR

Dependiendo del tiempo en cada paso, el paso se marca como “Marvelous”, “Perfect”, “Great”, “Good”, “Almost” o “Miss”. En la parte superior de la pantalla hay una barra de vida, y empieza a la mitad al comienzo de la canción. Los pasos Great o más incrementan la barra de vida hasta que se llena. Mientras que los pasos “Almost” y “Miss” la disminuyen. Los pasos “Good” no tienen efecto positivo o negativo. Si a un jugador reduce su barra de vida totalmente por fallar muchas veces, falla la canción y, solo en Versus, cuando los dos jugadores fallan, o si uno de los dos logra llegar al final de una canción. El jugador recibe una puntuación final que puede ir de una “AAA+” a una “E” (donde “AAA+” la mejor calificación obtenida, “D” la peor y “E” cuando falla la canción), basado en la cantidad de pasos correctos que haya realizado, en la sincronía con el juego y el conteo de calorías quemadas (vista en la evaluación desde Supernova2 y en versiones caseras), y cuando ambos ven una “E” o si ha terminado todas las canciones, se termina el juego. Normalmente, si falla la canción en curso, aparece “FAILED” en rojo antes de la evaluación, lo mismo ocurre si termina la canción con la palabra “CLEARED” en verde.

En una canción, existen factores que determinan su dificultad de baile, los cuales están marcados en pies, desde 4thMIX, los beats por minuto (BPM) y desde el DDRMAX, el Groove radar (desactivado al jugar en modo Happy en X2 y X3). Originalmente, las canciones usaban fondos con gráficos o videos aleatorios, pero desde DDR Supernova, fueron remplazados por videos exclusivos o escenarios, ya que antes no sabían que canción se estaba jugando y desde DDR Supernova 2 apareció el visor de canciones en vez de fondos. Ciertas canciones con video desde DDR X2 aparecen en pantallas del escenario.

Stepmania

Stepmania es un video juego de baile y ritmo, entra en la categoría de simulación inspirado en Dance Dance Revolution. Es un software libre y totalmente gratuito, disponible para sistemas como

Linux, Windows o Mac OS X, como también su código fuente es utilizado para “In The Groove” y “Pump It Up Pro”.

Es idéntico a Dance Dance Revolution con la diferencia de que el usuario puede interactuar libremente con el programa, pudiendo soportar las canciones de DDR PC e incorporar canciones a su gusto, caratulas, estilos visuales, etc. Las tres primeras versiones usaban la interfaz de DDRMAX. Desarrolladores crearon SM4 en sus dos versiones. La versión CVS estaba basado en la construcción del 2006, llamándose Stepmania 3.95. Chris Danford, creador de Stepmania, bifurcó la otra versión CVS y lo llamó SM 4 Beta. Después del anuncio del 4 beta, varias versiones CVS y SVN no oficiales fueron creadas por la comunidad de Stepmania como una versión 5.

Es totalmente accesible y se puede modificar al gusto de la persona pero siempre manteniendo el estilo clásico de su núcleo.

Drum Mania

Drum Mania es un juego musical producido por Bemani, la división musical de Konami.

La fecha de salida de Drum Mania fue en 1999 como un juego arcade, después se porto a la PlayStation 2 en Japon en el año 2000 como un título de salida. Siguietes versiones del juego han salido aproximadamente una al año.

El juego puede ser enlazado con su versión de guitarra Guitar Freaks, permitiendo el juego cooperativo, versiones anteriores también se podían enlazar con Keyboard Mania. Desde el 7th mix hacia adelante, se pudo conectar al sistema de Konami e-Amusement para el juego competitivo on-line.

Drum Mania simula el hecho de tocar una batería real. Se juega usando un controlador que imita a una batería. Seis Lanes colocadas de derecha a izquierda como el hi-hat, caja , tom alto , tom de piso , platillos y además un pedal para simular el bombo. Durante el juego, el jugador tiene que tocar los Pads y el pedal en sincronía con los Chips que van cayendo verticalmente desde arriba, y en sincronía con la música.



Figura 1.6: Interfaz del Drum Mania

La exactitud del jugador es juzgada en cada nota, y aunque el sistema de medir esta exactitud se ha cambiado a través de las versiones, actualmente el sistema emplea Perfect, Great, Good, Poor y Miss para evaluar el sincronismo de cada nota. Las categorías de Poor y Miss vaciaran el “Excite Gauge” del jugador mientras que mantener el sincronismo lo aumentara. Si el “Excite Gauge” se vaciá completamente el juego acaba. Los jugadores podrán jugar de tres a cinco canciones dependiendo de la configuración y según sus puntuaciones podrán jugar dos canciones adicionales. Cuando se completa una canción a los jugadores se les otorga una puntuación que va desde E hasta A, también incluyendo S y SS dependiendo de lo bien que hayan tocado la canción y la configuración de la puntuación en esa versión.

Drum Mania usa una batería electrónica modificada de Yamaha de la gama DTXPRESS. Esta batería es usada para navegar por los menús y pantallas de selección, también se puede navegar mediante los botones Select y Start en los laterales de la máquina.



Figura 1.7: Mueble arcade del Drum Mania

Drum Mania XG / Gitadora

En el año 2010 la serie se cambio de nombre a Drum Mania XG, aquí se introdujo el cambio en los controles de añadir un tom de suelo, un cymbal izquierdo y un pedal izquierdo a la maquina original, y para que hubiera más desarrollo en este nuevo juego se paro el desarrollo en la saga Drum Mania.

Más adelante en el año 2012 se cambio el nombre de Drum ManiaXG a Gitadora, dejando los controles iguales, limpiando mucho la interfaz del Drum ManiaXG y modificando los modos de conexión a Internet, fueron los principales cambios.



Figura 1.8: Mueble arcade del Gitadora

DTXMania

El DTXMania es un juego musical basado en el Drum Mania, su programador lo llevo a liberar en el año 2000 bajo la licencia MIT. Su idea original era coger el Drum Mania y poder ejecutarlo en un ordenador, al principio pensado para las baterías electrónicas Yamaha y poco a poco lo fue modificando para poder conectar cualquier batería electrónica MIDI, que al final se ha convertido en su principal objetivo.

Ya desde el inicio se fue separando del proyecto Drum Mania para acercarse más a un simulador real, antes que a un juego.

Fue el primero en implementar el hihat abierto y cerrado, de esta manera y si la batería MIDI tenia esta posibilidad tener que usar el pedal izquierdo, para tocar el hihat tal y como se tendría que hacer en una batería real. Siempre se dejaba el escoger como funcionaban estas opciones que

programaba por si tu batería no era tan potente o por si querías acercarte más a un juego antes que a un simulador.

Se expone la manera de puntuar clásica:

$(\text{Porcentaje de notas en Perfect} \times .85) + (\text{Porcentaje de notas en Great} \times .35) + (\text{Porcentaje de Max Combo} \times .15)$

Basado en este ratio, se asignan las siguientes puntuaciones:

- SS = 95.00+
- S = 80.00 - 94.99
- A = 73.00 - 79.99
- B = 63.00 - 72.99
- C = 53.00 - 62.99
- D = 45.00 - 52.99
- E = 0 - 44.99

Un detalle que muestra uno de los motivos que gusto a la gente que toca la batería es el siguiente, se puede tocar todas las notas de una canción y sí se ha cometido algún fallo de sincronismo no tener la canción al 100 %.

Muchos baterías vieron en este programa una gran ayuda para sus sesiones de practica gracias principalmente a dos puntos.

- 1. Poder detallar a que nivel querían el sincronismo.
- 2. Con la inclusión de poder tocar con la batería electrónica podían entrenar ritmos tal y como se han de tocar en la batería acústica.

Poco a poco se fue haciendo una gran comunidad que creaba canciones para el DTXMania y aportaban ideas para cada convertirlo en una ayuda para sus sesiones de entrenamiento.

DTXManiaHD

Cuando salio el Drum Mania XG en el año 2010, el programador de DTXMania añadió los cambios que se habían realizado para poder importar las canciones del Drum Mania XG al DTXManiaHD, se hizo un gran cambio en la interfaz y se actualizo a gráficos en alta calidad, pero por el cambio que se hizo en el pedal izquierdo que se añadió al Drum Mania XG, y el ya lo tenia incluido par el hihat, ya que en Drum Mania XG se puede usar el pedal izquierdo para hihat o para doble bombo, mientras que el DTXManiaHD tenia más libertad ya que podía funcionar de esta manera y además si la batería eléctrica tenia doble pedal izquierdo se podía separar estas funciones y tenerlos independientes.

Esto genera un problema en las canciones del DTXMania (si se quieren usar en el DTXManiaHD) ya que al solo tener un lane de bombo, lo más habitual para tocar una canción con doble bombo era poner todas las notas en ese lane y el usuario poner dos señales de su batería-MIDI para esta lane, esto entra en conflicto con como manejar el doble bombo ya que con el nuevo formato se especificaba con dos lanes todas estas notas.

El programador del DTXMania tomo la decisión de mantener los dos programas al mismo tiempo, y que fuera el usuario que escogiera según el tipo de canción que reproductor usar, o usando el DTXManiaHD saber que si usaba una canción del DTXMania podría tener conflictos con como manejar el doble bombo. Esto se explicara con más detalle más adelante del proyecto.

1.1.2. Historia. Composición musical mediante ordenador

Ahora que tenemos una visión de los simuladores y juegos musicales principales que hay actualmente, se tiene que hacer un pequeño resumen sobre la historia de la composición musical mediante ordenador para poder entender mejor las relaciones que tendremos entre baterías musicales MIDI y el reproductor de DTXMania.

En 1878, Thomas A. Edison patentó el fonógrafo, que utilizaba cilindros para grabar sonidos y poder reproducirlos de nuevos. Aunque se siguieron utilizando los cilindros durante algún tiempo, este invento revoluciono el concepto de la música.

Se suele considerar como el primer instrumento electrónico el Theremin, inventado por el profesor Léon Theremin alrededor de 1919–1920.

La grabación de sonidos dio un salto en 1927, cuando el inventor estadounidense J. A. O'Neill desarrolló un dispositivo para la grabación que utilizaba un tipo de cinta recubierta magnéticamente.

En 1942, AEG ya estaba realizando pruebas de grabación en estéreo.No obstante, estos dispo-

sitivos y técnicas fueron un secreto fuera de Alemania hasta el final de la Guerra, cuando varios de estos aparatos fueron capturados y llevados a Estados Unidos por Jack Mullin y otros. Estos grabadores capturados sirvieron de base para los primeros grabadores de cinta profesionales que fueron comercializados en Estados Unidos, el Model 200 producido por la empresa Ampex.

La cinta de audio magnética abrió un vasto campo de posibilidades sonoras para músicos, compositores, productores e ingenieros. La cinta de audio era relativamente barata y muy confiable, y su fidelidad en la reproducción mejor que cualquier otro medio de audio conocido hasta la fecha. Más importantemente, y a diferencia de los discos, ofrecía la misma plasticidad que la película: puede ser ralentizada, acelerada o incluso reproducirse al revés. Puede editarse también físicamente, incluso sólo segmentos de la cinta. O unirse diferentes trozos de cinta en loops infinitos que reproducen continuamente determinados patrones de material pregrabado. La amplificación de audio y el equipo de mezcla expandieron todavía más allá las posibilidades de la cinta como medio de producción, permitiendo que múltiples grabaciones fueran grabadas a la vez en otra cinta diferente. Otra posibilidad de la cinta era su capacidad de ser modificada fácilmente para convertirse en máquinas de eco para producir de modo complejo, controlable y con gran calidad efectos de eco y reverberación (lo que es prácticamente imposible de conseguir por medios mecánicos).

La banda sonora de *Forbidden Planet*, de Louis y Bebe Barron, fue compuesta completamente mediante circuitos caseros y magnetófonos en 1956.

El primer ordenador del mundo en reproducir música fue el CSIRAC, que fue diseñado y construido por Trevor Pearcey y Maston Beard. El matemático Geoff Hill programó el CSIRAC para tocar melodías de música popular. No obstante, el CSIRAC reproducía un repertorio estándar y no fue utilizado para ampliar el pensamiento musical o para tocar composiciones más elaboradas.

El impacto de los ordenadores continuó en 1956. Lejaren Hiller y Leonard Isaacson compusieron *Iliac Suite* para un cuarteto de cuerda, la primera obra completa en ser compuesta con la asistencia de un computador utilizando un algoritmo en la composición. Posteriores desarrollos incluyeron el trabajo de Max Mathews en Bell Laboratories, quien desarrolló el influyente programa MUSIC I. La tecnología de vocoder fue otro importante desarrollo de esta época.

A inicios de la década de los ochentas, compañías estaban vendiendo versiones compactas y de bajo precio de los sintetizadores para el público. Esto, junto con el desarrollo de el protocolo Musical Instrument Digital Interface (MIDI), hizo más fácil integrar y sincronizar sintetizadores y otros instrumentos electrónicos para su uso en la composición musical. En los noventas, los emuladores de sintetizadores comenzaron a aparecer para computadoras, conocidos como sintetizadores de software. Posteriormente, VST's y otros plugins eran capaces de emular el hardware de sintetizadores clásicos hasta cierto punto.

En 1980, un grupo de músicos y fabricantes se pusieron de acuerdo para estandarizar una interfaz a través del que diferentes instrumentos pudieran comunicarse entre ellos y el ordenador principal. El estándar se denominó MIDI (Musical Instrument Digital Interface). En agosto de 1983,

la especificación 1.0 de MIDI fue finalizada.

La llegada de la tecnología MIDI permitió que con el simple acto de presionar una tecla, controlar una rueda, mover un pedal o dar una orden en un micro ordenador se pudieran activar todos y cada uno de los dispositivos del estudio remotamente y de forma sincronizada, respondiendo cada dispositivo de acuerdo a las condiciones prefijadas por el compositor.

Como el tema de composición y edición musical se escapa de los objetivos de este proyecto nos centraremos en los formatos de reproducción.

Formatos sin comprimir WAV

La manera general de almacenar audio digital es muestreando el voltaje de audio, que al reproducirlo, corresponde a un nivel de señal en un canal individual con una cierta resolución, el número de bits por muestreo en intervalos regulares (creando la frecuencia de muestreo). Estos datos después pueden ser almacenados sin comprimir o comprimidos para reducir el tamaño del formato.

Evidentemente los formatos sin comprimir son los primeros que se crearon y los más usados a la hora de una necesidad de inmediatez a costa de tener que tenerlos en memoria.

WAV (o WAVE), apócope de WAVE form audio file format, es un formato de audio digital normalmente sin compresión de datos desarrollado y propiedad de Microsoft y de IBM que se utiliza para almacenar sonidos en el PC, admite archivos mono y estéreo a diversas resoluciones y velocidades de muestreo, su extensión es .wav.

Es una variante del formato RIFF (Resource Interchange File Format, formato de fichero para intercambio de recursos), método para almacenamiento en “paquetes”, y relativamente parecido al IFF y al formato AIFF usado por Macintosh. El formato toma en cuenta algunas peculiaridades de la CPU Intel, y es el formato principal usado por Windows.

A pesar de que el formato WAV es compatible con casi cualquier códec de audio, se utiliza principalmente con el formato PCM (no comprimido) y, al no tener pérdida de calidad, es adecuado para uso profesional. Para tener calidad CD de audio se necesita que el sonido se grabe a 44100 Hz y a 16 bits. Por cada minuto de grabación de sonido se consumen unos 10 megabytes de espacio en disco. Una de sus grandes limitaciones es que solo se pueden grabar archivos de 4 gigabytes como máximo, lo cual equivale aproximadamente a 6,6 horas en calidad de CD de audio. Es una limitación propia del formato, independientemente de que el sistema operativo donde se utilice sea MS Windows u otro distinto, y se debe a que en la cabecera del fichero se indica la longitud del mismo con un número entero de 32 bits, lo que limita el tamaño del fichero a un máximo de 4294967295 bytes (o 4 gigabytes)

En Internet no es popular, fundamentalmente porque los archivos sin compresión son muy grandes. Son más frecuentes los formatos comprimidos con pérdida, como el MP3 o el Ogg Vorbis. Como éstos son más pequeños, la transferencia a través de Internet es mucho más rápida. Además, existen códecs de compresión sin pérdida más eficaces, como FLAC.

Formato con compresión Mp3

Ahora pasamos a describir los formatos de sonido donde se emplean algoritmos de compresión.

MPEG-1 Audio Layer III o MPEG-2 Audio Layer III, más comúnmente conocido como MP3 es un formato de compresión de audio digital patentado que usa un algoritmo con pérdida para conseguir un menor tamaño de archivo. Es un formato de audio común usado para música tanto en ordenadores como en reproductores de audio portátil.

Los archivos MPEG-1 corresponden a las velocidades de muestreo de 32, 44.1 y 48 kHz.

Los archivos MPEG-2 corresponden a las velocidades de muestreo de 16, 22.05 y 24 kHz.

MP3 fue desarrollado por el Moving Picture Experts Group (MPEG) para formar parte del estándar MPEG-1 y del posterior y más extendido MPEG-2. Un MP3 creado usando una compresión de 128kbit/s tendrá un tamaño de aproximadamente unas 11 veces menor que su homónimo en CD. Un MP3 también puede comprimirse usando una mayor o menor tasa de bits por segundo, resultando directamente en su mayor o menor calidad de audio final, así como en el tamaño del archivo resultante.

Este formato fue desarrollado principalmente por Karlheinz Brandenburg, director de tecnologías de medios electrónicos del Instituto Fraunhofer IIS, perteneciente al Fraunhofer-Gesellschaft-Schultagen - red de centros de investigación alemanes - que junto con Thomson Multimedia controla el grueso de las patentes relacionadas con el MP3. La primera de ellas fue registrada en 1987, en ese año en el laboratorio de tecnologías de medios electrónicos, los alemanes intentaban resolver un dilema, como difundir el sonido digital. Los archivos en CD, eran pesados y engorrosos, las lectoras de CD, eran novedad, también instalarlas en una PC, no así empezaban, a subir los primeros archivos en CD al disco duro de la computadora, todo esto ocurría en 1987, también registraron varias patentes más en 1991. No fue hasta julio de 1995 cuando Brandenburg usó por primera vez la extensión .mp3 para los archivos relacionados con el MP3 que guardaba en su ordenador, en el proceso de desarrollo del formato participó también el ingeniero Leonardo Chiariglione quien tuvo la idea de los estándares que podrían ser útiles para este fin. Un año después su instituto ingresaba en concepto de patentes 1,2 millones de euros. Diez años más tarde esta cantidad ha alcanzado los 26,1 millones.

Tras el desarrollo de reproductores portátiles, y su integración en reproductores para automó-

viles y minisistemas de sonido hogareños, el formato MP3 en 2002 llega más allá del mundo de la informática.

El formato MP3 se convirtió en el estándar utilizado para streaming de audio y compresión de audio con pérdida de mediana fidelidad gracias a la posibilidad de ajustar la calidad de la compresión, proporcional al tamaño por segundo (bitrate), y por tanto el tamaño final del archivo, que podía llegar a ocupar 12 e incluso 15 veces menos que el archivo original sin comprimir.

Fue el primer formato de compresión de audio popularizado gracias a Internet, ya que hizo posible el intercambio de ficheros musicales. Los procesos judiciales contra empresas como Napster y AudioGalaxy son resultado de la facilidad con que se comparten este tipo de ficheros.

A principios de la década de los 2000 otros formatos de audio comprimido como Windows Media Audio, ATRAC, AAC y Ogg Vorbis empiezan a ser masivamente incluidos en programas de audio para computación, dispositivos, sistemas operativos, teléfonos y reproductores portátiles, lo que hizo prever que el MP3 fuera paulatinamente cayendo en desuso, en favor de otros formatos, como los mencionados, de mucha mejor calidad. Una de las desventajas del formato MP3 es que tiene patente. Técnicamente, el tener una patente no significa que su calidad sea inferior ni superior, pero impide que la comunidad pueda seguir mejorándolo y puede obligar a pagar por la utilización del códec; lo cual ocurre en el caso de los dispositivos que lo usan como los teléfonos y las tabletas. Aun así, hoy día, el formato mp3 continúa siendo el más usado y el que goza de más éxito con una presencia cada vez mayor. Algunas tiendas en línea como Amazon venden su música en este formato por cuestiones de compatibilidad.

Un fichero MP3 se constituye de diferentes tramas que a su vez se componen de una cabecera y los datos en sí. Esta secuencia de datos es la denominada “stream elemental”. Cada una de las tramas es independiente, es decir, pueden ser cortadas las tramas de un fichero MP3 y después reproducirlos en cualquier reproductor MP3 del Mercado. La cabecera consta de una palabra de sincronismo que es utilizada para indicar el principio de una trama válida. A continuación siguen una serie de bits que indican que el fichero analizado es un fichero Standard MPEG y si usa o no la capa 3. Después de todo esto, los valores difieren dependiendo del tipo de archivo MP3. Los rangos de valores quedan definidos en la norma ISO/IEC 11172-3.

Formato MIDI

Se tiene que describir el formato y protocolo MIDI, ya que es el principal formato tanto para composición musical, como para establecer una comunicación entre elementos de música electrónicos. Se tendrá que hacer un repaso tanto a la historia del MIDI como a sus principales objetivos.

MIDI (abreviatura de Musical Instrument Digital Interface) es un estándar tecnológico que describe un protocolo, una interface digital y conectores que permiten que varios instrumentos

musicales electrónicos, ordenadores y otros dispositivos relacionados se conecten y comuniquen entre sí. Una simple conexión MIDI puede transmitir hasta dieciséis canales de información que pueden ser conectados a diferentes dispositivos cada uno.

El sistema MIDI lleva mensajes de eventos que especifican notación musical, tono y velocidad; señales de control para parámetros musicales como lo son la dinámica, el vibrato, paneo, cues y señales de reloj que establecen y sincronizan el tempo entre varios dispositivos. Estos mensajes son enviados mediante un cable MIDI a otros dispositivos que controlan la generación de sonidos u otras características. Estos datos también pueden ser grabados en un hardware o software llamado secuenciador, el cual permite editar la información y reproducirla posteriormente.

La tecnología MIDI fue estandarizada en 1983 por un grupo de representantes de la industria de la música llamado MIDI Manufacturers Association (MMA). Todos los estándares MIDI son desarrollados y publicados en conjunto por la MMA en Los Angeles, California, en Estados Unidos; y para Japón, el comité MIDI de la Association of Musical Electronics Industry (AMEI) en Tokio.

Las ventajas del uso de MIDI incluyen su tamaño (una canción completa puede ser codificada en unos cientos de líneas, por ejemplo en algunos kilobytes) y la fácil manipulación, modificación y selección de los instrumentos.

El desarrollo del MIDI Para finales de los setenta, los dispositivos electrónicos musicales se volvieron más comunes y menos costosos en América del Norte, Europa y Japón. Los primeros sintetizadores analógicos eran usualmente monofónicos y controlados mediante el voltaje producido por sus teclados. Los fabricantes usaron este voltaje para conectar instrumentos en conjunto y así un solo dispositivo podría controlar uno u otros más, sin embargo este sistema no era adecuado para los sintetizadores polifónicos y digitales. Algunos fabricantes crearon sistemas que permitían que su propio equipo fuera interconectado, pero los sistemas no eran compatibles, así que los sistemas de otros fabricantes podrían no ser sincronizados con otros.

En junio de 1981, el fundador de Roland, Ikutaro Kakehashi, propuso la idea de una estandarización al fundador de Oberheim Electronics, Tom Oberheim, que en ese entonces habló con el presidente de Sequential Circuits, Dave Smith. En octubre de 1981, Kakehashi, Oberheim y Smith discutieron la idea con los representantes de Yamaha, Korg y Kawai.

Los ingenieros y diseñadores de sintetizadores de Sequential Circuits, Dave Smith y Chet Wood, concibieron la idea de una interface para sintetizadores universal que permitiera una comunicación directa entre el equipo de varios fabricantes. Smith propuso este estándar en noviembre de 1981 a Audio Engineering Society. Por los siguientes dos años, el estándar fue discutido y modificado por representantes de compañías como Roland, Yamaha, Korg, Kawai, Oberheim y Sequential Circuits, renombrado como Musical Instrument Digital Interface. El desarrollo del MIDI fue presentado al público por Robert Moog en octubre de 1982 en la revista Keyboard.

En la exhibición NAMM de enero de 1983, Smith logró presentar la conexión MIDI entre el sintetizador analógico Prophet 600 y el Jupiter-6. El protocolo MIDI 1.0 fue publicado en agosto de 1983. El estándar MIDI fue revelado por Ikutaro Kakehashi y Dave Smith, quienes después recibieron el Grammy técnico en 2013 por su papel en el desarrollo del MIDI.

El impacto del MIDI en la industria de la música El uso del MIDI estaba originalmente limitado a aquellos que quisieran hacer uso de instrumentos electrónicos en la producción musical. El estándar permitió que diferentes instrumentos pudieran comunicarse con otros y con los ordenadores. Esto causó una rápida expansión en las ventas y en la producción de instrumentos electrónicos y software musical. Esta intercompatibilidad permitió que un dispositivo pudiera ser controlado desde otro, lo que ayudó a músicos que tuvieran la necesidad de utilizar distintos tipos de hardware. La introducción del MIDI coincidió con la llegada de los ordenadores personales, los primeros samplers (los cuales permitían reproducir sonidos pre-grabados en presentaciones en vivo para incluir efectos que previamente no eran posibles fuera de los estudios) y los sintetizadores digitales, los cuales permitían almacenar sonidos preprogramados y posteriormente ser utilizados mediante un botón. Las posibilidades creativas que permitió la tecnología MIDI ayudaron a revivir la industria de la música durante los ochenta.

El formato MIDI introdujo muchas capacidades, las cuales transformaron la manera en que los músicos trabajaban. La secuenciación MIDI hizo posible que un usuario sin habilidad para la escritura musical pudiera desarrollar arreglos complejos. Un acto musical con uno o dos miembros, ambos operando múltiples dispositivos MIDI, puede ser una presentación con un sonido similar a grupos con mayor número de músicos. El costo de contratar músicos para un proyecto podría ser reducido o eliminado, y producciones complejas pueden ser realizadas en un sistema pequeño como una estación de trabajo MIDI, un sintetizador con un teclado integrado y un secuenciador. Músicos profesionales pueden realizar esto en un espacio llamado “home recording”, sin la necesidad de alquilar un estudio de grabación profesional con personal. Trabajando la preproducción en tal entorno, una artista puede reducir los costos de grabación llegando al estudio con un trabajo que está parcialmente completo. Las partes rítmica y de fondo pueden ser secuenciadas y posteriormente reproducidas en el escenario. Las presentaciones requieren menor transportación y tiempo de preparación del equipo debido a las diferentes y reducidas conexiones necesarias para reproducir varios sonidos. La tecnología educativa compatible con MIDI ha transformado la educación musical.

En 2012, Ikutaro Kakehashi y Dave Smith ganaron el Grammy técnico por el desarrollo del MIDI en 1983.

Control de instrumentos El MIDI fue inventado para que los instrumentos musicales se pudieran comunicar unos con otros y que un instrumento pudiera controlar a otro. Los sintetizadores analógicos que no tenían un componente digital y que fueron construidos antes del desarrollo del MIDI pueden ser ajustados con kits que convierten los mensajes MIDI a voltajes de control analógicos. Cuando una nota es tocada en un instrumento MIDI, ésta genera una señal digital que puede

ser usada para activar la nota en otro instrumento. La capacidad de un control remoto permite que instrumentos de gran tamaño sean remplazados con pequeños módulos de sonido. Esto permite a los músicos combinar instrumentos para alcanzar un sonido pleno o para crear combinaciones como un piano acústico y cuerdas. MIDI también permite que otros parámetros de los instrumentos sean controlados de manera remota. Los sintetizadores y los samplers tienen varias herramientas para el modelado de un sonido. La frecuencia de un filtro y el ataque de una envolvente, o el tiempo que tarda un sonido en llegar a su valor máximo son ejemplos de los parámetros de los sintetizadores, y pueden ser controlados de manera remota a través de MIDI. Dispositivos de efectos tienen diferentes parámetros, como el tiempo de reverberación o delay. Cuando el número de un controlador MIDI es asignado a unos parámetros, el dispositivo responderá a los mensajes que reciba de dicho controlador. Controles como perillas, switches y pedales pueden ser utilizados para enviar estos mensajes. Un conjunto de parámetros establecidos puede ser guardado en la memoria de un dispositivo como un patch. Estos pueden ser seleccionados de manera remota a través de cambios de programa MIDI. El MIDI estándar permite una selección de 128 programas diferentes, pero los dispositivos pueden permitir más ajustando sus patches en bancos con 128 programas cada uno y combinando el mensaje de cambio de programa para la selección de un banco.

Composición Los eventos MIDI pueden ser secuenciados a través de un editor MIDI o una estación de trabajo especializada. Varias DAW están específicamente diseñadas para trabajar MIDI como componente integral. Las secuencias MIDI han sido desarrolladas en varios DAW para que los mensajes MIDI puedan ser modificados. Estas herramientas permiten a los compositores probar y editar su trabajo con una mayor rapidez y eficiencia que otras soluciones como la grabación multipista, mejorar la eficiencia de compositores y permitir crear arreglos complejos.

Debido a que el MIDI es un conjunto de comandos que crean sonidos, las secuencias MIDI pueden ser manipuladas de diferentes maneras en comparación con el audio pregrabado. Es posible cambiar la tonalidad, la instrumentación o el tempo de un arreglo MIDI y re-acomodar sucesiones de manera individual. La habilidad de componer ideas y escucharlas de manera inmediata permite a los compositores experimentar. Programas de composición algorítmica permiten que ejecuciones generadas por computadora puedan ser utilizadas como ideas para canciones o acompañamiento.

Algunos compositores tomaron ventaja de la tecnología MIDI 1.0 y el General MIDI (GM) que permitía transferir datos musicales entre varios instrumentos utilizando un set de comandos y parámetros estandarizado. Los datos compuestos a través de una secuencia MIDI pueden ser guardados como Standard MIDI File (SMF), distribuidos de manera digital y reproducidos por cualquier computadora o instrumento electrónico que esté adherido al mismo estándar MIDI, GM, y SMF. Los datos MIDI son mucho más pequeños que las grabaciones de archivos de audio.

MIDI y los ordenadores En la época en que el MIDI fue introducido la industria de la computación estaba enfocada en ordenadores mainframe. Los ordenadores personales no eran muy comunes. El mercado de los ordenadores personales se estabilizó al mismo tiempo en que el MIDI

apareció, con ellos los ordenadores se convirtieron en una opción viable para la producción musical. En los años posteriores a la ratificación de la especificación MIDI, las características MIDI fueron adaptadas a varias de las primeras plataformas, incluyendo Apple II Plus, IIe y Macintosh, Commodore 64 y Amiga, Atari ST, Acorn Archimedes, y PC DOS. Macintosh fue la favorita entre los músicos estadounidenses.

Se encontraba a un precio competitivo, y sería años después que la eficiencia y su interfaz gráfica sería igualada por las PC's. El Atari ST fue el favorito de Europa, donde las Macintosh eran más caras. Las computadoras Apple incluían un hardware de audio que era más avanzado que el de sus competidores. La Apple II GS usaba un chip de sonido digital diseñado para el sintetizador Ensoniq Mirage. En posteriores modelos se empleó un sistema especializado de audio con procesadores mejorados, los cuales llevaron a las demás compañías a mejorar sus productos. El Atari ST era preferido debido a que los conectores MIDI estaban integrados directamente en la computadora. La mayoría del software musical de la primera década de publicación del MIDI fue para la Apple o el Atari. Para el lanzamiento de Windows 3.0 en 1990, las PC's habían mejorado su interfaz gráfica junto con sus procesadores, por lo que diferentes softwares comenzaron a aparecer en diferentes plataformas.

Archivos MIDI estándar El formato estándar MIDI permite una manera estandarizada para almacenar, transportar y abrir secuencias en otros sistemas. El compacto tamaño de estos archivos ha permitido que sean implementados de manera numerosa en ordenadores, teléfonos y páginas de Internet. Fueron creados para su uso universal e incluir información como el valor de las notas, tiempo y nombre de las pistas.

La lírica puede ser incluida como metadata, que puede ser visualizada en máquinas de karaoke. La especificación SMF fue desarrollada y mantenida por MMA. Los SMF's son creados como formato para exportar información a secuenciadores software o estaciones musicales de trabajo. Organizan los mensajes MIDI en una o más pistas y en marcas temporales para volver a reproducir las secuencias. Una cabecera contiene la información del número de pistas, tempo y en cuál de los tres formatos SMF está el archivo. Un archivo del tipo 0 contiene la información de una presentación completa en una sola pista, mientras que las de tipo I contienen la información de cada una de las pistas ejecutadas de manera sincronizada. Los archivos de tipo II raramente son utilizados y guardan múltiples arreglos, cada uno tiene su propia pista para ser reproducida en secuencia. Microsoft Windows empaqueta el SMF con Downloadable Sounds (DLS) en un archivo informático para el intercambio de recursos (RIFF), con archivos RMID con extensión .rmi. RIFF-RMID ha sido depreciado a favor de Extensible Music Files (XMF).

Intercambio de archivos Un archivo MIDI no es una grabación de la música. En cambio, es una secuencia de instrucciones, que puede ocupar 1000 veces menos espacio en disco que una grabación. Esto hizo que los arreglos hechos en archivos MIDI se convirtieran en una manera más atractiva de compartir música, antes de la llegada del Internet y dispositivos con un almacenamiento superior.

Los archivos MIDI licenciados se encontraban disponibles en formato de disquete en tiendas de Europa y Japón durante los noventas. La mayor desventaja de esto era la gran variedad que existía entre las tarjetas de audio de los usuarios y las muestras de audio o sonidos sintetizados en la tarjeta que el MIDI retomaba de manera simbólica.

Aun una tarjeta de sonidos con samples de alta calidad puede tener inconsistencias entre la calidad de un instrumento a otro, mientras que diferentes modelos de tarjetas no garantizaban un consistencia en el sonido de un mismo instrumento. Las primeras tarjetas económicas, como AdLib y Sound Blaster, utilizaban una versión simplificada de la tecnología de síntesis por modulación de frecuencias de Yamaha reproducida a través de convertidores digitales-analógicos de baja calidad. La baja calidad de reproducción de estas tarjetas ubicuas se asumía que de algún modo era debido al MIDI. Esto creo la percepción del MIDI como audio de bajo calidad, mientras que en realidad el MIDI no tiene un sonido y la calidad de su reproducción depende totalmente de la calidad el dispositivo que lo reproduzca (y los samples del dispositivo).

Software MIDI La principal ventaja de los ordenadores personales en un sistema MIDI es que puede ser utilizado con diferentes propósitos, dependiendo del software utilizado. La capacidad multitarea de los sistemas operativos permite la operación de varios programas de manera simultánea que puedan compartir los datos unos con otros.

Secuenciadores Un software para secuenciar permite ciertos beneficios para un compositor u arreglista. Permite que el MIDI grabado sea manipulado utilizando a través de las características de edición básicas de un ordenador como cortar, copiar y pegar o arrastrar y soltar. Los atajos de teclado pueden ser utilizados para agilizar el ritmo de trabajo y las herramientas de edición MIDI pueden ser seleccionadas a través de comandos. El secuenciador permite que cada canal sea reproducido por un sonido diferente además de mostrar una pre-visualización gráfica del arreglo. Existen distintas herramientas de edición, incluyendo una visualización en notación musical. Herramientas como loops, cuantización, aleatoriedad y transposición simplifican el proceso de creación de arreglos. La creación de beats es simplificada y el groove puede ser duplicado en la sensación rítmica de otra pista. La expresión realista puede ser agregada a través de la manipulación de controladores en tiempo real. Una mezcla puede ser llevada a cabo y el MIDI puede ser sincronizado con pistas de audio o video grabadas. Los avances pueden ser guardados y llevados a otra computadora o estudio

Los secuenciadores pueden tomar diferentes formas como editores de ritmos de batería que permiten al usuario crear ritmos a través de clics en rejillas de patrones y hacer loops con secuencias, por ejemplo el ACID Pro, que permite al MIDI ser combinado con audios pre-grabados cuyos tiempos y notas son empatados. La secuencia de cues es utilizada para activar diálogos, efectos de sonido y segmentos musicales en transmisiones.

Existen programas que de manera dinámica pueden generar pistas de acompañamientos llamadas programas de “auto-acompañamiento”. Estos crean el arreglo de una banda completa a partir

del estilo que usuario seleccione, los resultados son enviados a un dispositivo de sonido MIDI para generar los sonidos. Las pistas generadas pueden ser utilizadas como herramientas de practica o educativas, también como acompañamiento para presentaciones en vivo o ayuda para la composición de canciones.

Dispositivos MIDI Ahora se expondra una selección de los dispositivos MIDI que existen en el mercado, además de ir exponiendo sus características.

Conector clásico MIDI Los cables terminan en un conector DIN de 180°. Las aplicaciones estándar emplean solo tres de los cinco conductores: tierra y un par de cables balanceados que llevan un señal de 5v. Esta configuración del conector solo puede llevar mensajes en una dirección, así que es necesario un cable para una comunicación de dos vías. Algunas aplicaciones prioritarias como la alimentación phantom para algunos controladores utilizan los pines sobrantes para la transmisión de corriente directa (DC).



Figura 1.9: Conector clásico MIDI

Optoacopladores mantienen los dispositivos MIDI eléctricamente separados de otros conectores, lo cual previene bucles de masa y protege al equipo de picos de voltage. No hay manera de detectar errores en MIDI, así que el tamaño máximo del cable es de 15 metros para limitar interferencias destructivas.

La mayoría de los dispositivos no copian los mensajes de la entrada a su puerto de salida. Un tercer tipo de puerto, el puerto “thru”, emite una copia de todo lo que es recibido en el puerto de entrada, permitiendo que los datos sean transmitidos a otro instrumento en una “Daisy chain”. No todos los dispositivos cuentan con un puerto thru y dispositivos que carecen de la característica de generar datos MIDI, como unidades de efectos o módulos de sonido, pueden no incluir un puerto de salida.

Interfaces La función principal de una interfaz MIDI para ordenador es sincronizar los relees entre un dispositivo MIDI y el ordenador. Algunas tarjetas de sonido de ordenador incluyen un conector MIDI estándar, mientras que en otras se conecta a través de D-sub, USB, Firewire o Ethernet. El creciente uso de conectores USB en los 2000’s ha llevado a una disponibilidad de interfaces MIDI a USB que puedan transferir canales MIDI a computadoras con USB incluido. Algunos controladores MIDI están equipados con conectores USB y pueden ser conectados en ordenadores que empleen software musical.

La transmisión serial MIDI lleva a problemas de sincronización. Músicos experimentados pueden detectar diferencias de $1/3$ de milisegundos (ms) (es el tiempo que tarda el sonido en viajar 4 pulgadas) y un mensaje MIDI de 3 bytes requiere 1ms para transmitirse. Debido a que el MIDI es serial, solo puede ser enviado un evento a la vez. Si un evento es enviado a dos canales a la vez, el evento con número de canal mayor no podrá ser transmitido hasta que el primero haya acabado y será retrasado 1ms. Si un evento es enviado a todos los canales al mismo tiempo, el evento con un número de canal mayor será retrasado por mucho 16ms. Esto ha contribuido a el surgimiento de interfaces MIDI con múltiples entradas y salidas debido a que la sincronización mejora cuando los eventos son enviados en diferentes puertos a diferencia de varios canales en un mismo puerto. El término “tropiezo MIDI” se refiere a los errores audibles resultantes de una transmisión retrasada.

Controladores Existen dos tipos de controladores MIDI: controladores para performance que generan notas y son utilizados para ejecutar música, y controladores que pueden no transmitir notas pero pueden transmitir otros tipos de eventos en tiempo real. Varios dispositivos son la combinación de los dos tipos.

Controladores para performance El MIDI fue diseñado con los teclados en mente y cualquier otro controlador que no posea un teclado es considerado como un controlador “alternativo”. Esto ha sido visto como una limitación para los compositores que no están interesados la música que emplea teclados, la flexibilidad y compatibilidad MIDI fue introducida a otros tipos de controladores incluyendo guitarras, instrumentos de viento y cajas de ritmo.

Teclados Los teclados musicales son el tipo de controlador MIDI más común. Pueden ser encontrados en diferentes tamaños desde 25 teclas, modelos de dos octava, hasta instrumentos de 88 teclas. Algunos solo incluyen el teclado, aunque existen otros controladores en tiempo real como perillas, sliders y palancas. Usualmente existen conexiones para pedales de sustain y de expresión. La mayoría de los controladores con teclado permiten dividir el área de piano en zonas, las cuales pueden ser de diferentes tamaños y sobreponerse. Cada zona corresponde a un canal MIDI diferente y un set diferente de controladores, pueden ser usados para tocar cualquier rango de notas seleccionado. Esto permite a un solo instrumento tocar varios sonidos. Las capacidades MIDI también pueden ser encontradas en instrumentos de teclado tradicionales como pianos y pianos Rhodes. Pedaleros pueden controlar los tonos de un órgano MIDI o pueden tocar un sintetizador como el Moog Taurus.

Baterías y controladores de percusión Los teclados pueden ser utilizados para accionar sonidos de baterías pero no son prácticos para tocar patrones repetitivos como redobles debido a las dimensiones de las teclas. Después de los teclados, los pads de batería son los controladores para performance MIDI más significantes. Los controladores de percusión pueden estar integrados en cajas de ritmo, pueden ser superficies de control independientes o emular y sentirse como instrumentos

de percusión. Los pads integrados en cajas de ritmos usualmente son muy pequeños y frágiles para ser tocados con baquetas, por lo que son tocados con los dedos. Drum pads especializados como el Roland Octapad o el DrumKAT son tocados con las manos o baquetas, están contruidos como un set de batería. Existen otros controladores de percusión con el MalletKAT, parecido al vibráfono, y el Marimba Lumina de Don Buchla. Accionadores MIDI pueden ser instalados en una batería acústica e instrumentos de percusión. Los Pads pueden accionar un dispositivo MIDI que puede ser casero a partir de un sensor piezoeléctrico o un pad de practica.

Instrumentos Un instrumento MIDI tiene puertos para enviar y recibir señales MIDI, un CPU para procesar dichas señales, una interfaz que permita al usuario programarlo, un circuito de audio que genere sonidos y controladores. El sistema operativo y los sonidos de fábrica usualmente están almacenados en una memoria de solo lectura (ROM).

Un instrumento MIDI también puede ser un modulo independiente (sin la necesidad de un teclado) conformado por una tarjeta de sonido General MIDI (GM, GS y /XG) editable dentro de la misma, incluyendo cambios de transport/tono, cambios de instrumento MIDI, ajuste de volumen, panel, niveles de reverberación y otros controladores MIDI. Normalmente, el modulo MIDI incluye una pantalla grande, permitiendo al usuario visualizar la información dependiendo de la función seleccionada. Otras funciones incluyen el visualizar la lírica, usualmente incluida en un archivo MIDI o Karaoke MIDI, listas de pistas, librería de canciones y pantallas de edición. Algunos módulos MIDI incluyen un armonizador y la capacidad de reproducir y reajustar archivos MP3.

Sintetizadores Los sintetizadores pueden emplear cualquier variedad de técnicas para generar sonido. Estos normalmente incluyen un teclado integrado, o pueden existir como “módulos de sonido” que generan sonidos a partir de un controlador externo. Los módulos de sonidos están típicamente diseñados para ser colocados en un rack de 19 pulgadas. Los fabricantes producen comúnmente un sintetizador en versiones independiente y para rack, usualmente la versión con teclado varia de tamaño.

Samplers Un sampler puede grabar y digitalizar audio, almacenarlo en una memoria de acceso aleatorio (RAM) y reproducirlo posteriormente. Los samplers normalmente permiten al usuario editar un sample y guardarlo en un disco duro, aplicarle efectos y modificar su sonido a través de las mismas herramientas usadas en los sintetizadores. También pueden tener un teclado o estar montados en un rack. Los instrumentos que generan sonidos a través de su reproducción pero que no tienen capacidades de grabación son conocidos como “ROMplers”.

Los samplers no se convirtieron en instrumentos MIDI viables tan rápido como lo hicieron los sintetizadores debido al costo de la memoria y el poder de procesamiento en este entonces. El primer sampler MIDI de bajo costo fue el Ensoniq Mirage, lanzado en 1984. Los samplers MIDI

normalmente están limitados debido a sus pequeñas pantallas empleadas para editar las formas de onda sampleadas, aunque algunos pueden ser conectados a un monitor de computadora.

Cajas de ritmos Las cajas de ritmos normalmente son dispositivos especializados que reproducen samples de batería y sonidos de percusión. Usualmente tienen un secuenciador que permite la creación de patrones ritmos para incorporarlos en el arreglo de alguna canción. Comúnmente tienen múltiples salidas que permiten que cada uno de los sonidos sea asignado a cada una. Los sonidos individuales de las baterías pueden ser reproducidos desde otro instrumento MIDI o desde un secuenciador.

Especificaciones técnicas Los mensajes MIDI están conformados de una “palabra” de 8 bits (llamados bytes) que son transmitidos de manera serial a 31.25 kbit/s. Esta tasa fue escogida debido a que es una división exacta de 1 MHz, la velocidad en la que varios de los primeros microprocesadores operan. El primer bit de cada palabra identifica si la palabra es un byte de estatus o de datos, los siguientes siete bits son la información. Un bit de inicio y otro de pausa son agregados a cada byte por cuestiones de sincronización, así que un mensaje MIDI requiere de diez bits para transmitirse.

Una conexión MIDI puede llevar dieciséis canales independiente de información. Los canales son numerados del 1 al 16 pero en realidad corresponden al orden del código binario del 0 al 15. Un dispositivo puede ser configurado para solo escuchar canales específicos e ignorar los mensajes enviados de otros o puede escuchar a todos los canales sin importar su dirección. Un dispositivo puede ser monofónico (el inicio de una nueva señal de “note-on” MIDI implica el final de la nota previa) o polifónico (múltiples notas pueden sonar al mismo tiempo, hasta que el límite de la polifonía del instrumento se haya alcanzado, las notas hayan terminado su envolvente o el comando “note-off” haya sido recibido. Los dispositivos que reciben los mensajes normalmente tienen cuatro combinaciones de los modos “omni off/on” vs. “mono/poly”.

En la especificación técnica hay 128 Instrumentos General MIDI para el proyecto el que nos interesa es el número 10 - Caja de música.

La flexibilidad del MIDI y su gran aceptación ha llevado a varios refinamientos del estándar, además ha permitido que se aplique a propósitos más allá de los que se tenían planeados. Para ello se han diseñado extensiones al general MIDI.

Hardwarees alternativos de transporte Además de la tasa de transmisión 31.25 kbit/s en un conector DIN de cinco pines, otros conectores comunes han sido usados para la misma información eléctrica y la transmisión de señales MIDI en diferentes formas a través de USB, IEEE 1394 o FireWire y Ethernet.

USB y FireWire Los miembros de USB-IF en 1999 desarrollaron un estándar para MIDI a través de USB, el “Universal Serial Bus Device Class Definition for MIDI Devices”. El MIDI sobre el USB ha sido más común que otras interfaces que ha sido empleadas para las conexiones MIDI (serial, joystick, etc.) han desaparecido de las computadoras personales. Los sistemas operativos Microsoft Windows, Macintosh OS X y Apple iOS han incluido drivers para compatibilidad con “Universal Serial Bus Device Class Definition for MIDI Devices”. Los drivers también están disponibles para Linux. Algunos fabricantes decidieron implementar una interface MIDI sobre el USB que está diseñado para operar diferente de la especificación, usando drivers personalizados.

Apple Computer desarrollo la interface FireWire durante los noventas. Comenzó a aparecer en cámaras de video digitales hacia finales de la década y en modelos de la G3 Macintosh en 1999. Fue creado para aplicaciones multimedia. A diferencia del USB, FireWire usa controladores inteligentes que pueden manejar su propia transmisión sin la atención de un CPU principal.

Ethernet La implementación del MIDI en la red de computadoras permite nuevas capacidades de conexiones y permite un canal con gran ancho de banda que las primeras alternativas de MIDI, como ZIPI, trataron de crear. Las implementaciones propietarias han existido desde los ochentas, como lo es el uso de cables fibra óptica para la transmisión. La especificación abierta RTP MIDI del Grupo de trabajo de ingeniería de Internet está adquiriendo el apoyo de la industria debido a que los protocolos propietarios MIDI/IP requieren costos altos de licencias o no ofrecen ninguna ventaja además de la velocidad sobre el protocolo MIDI original. Apple ha apoyado este protocolo desde Mac OS X 10.4 y un driver de Windows basado en la implementación de Apple existe desde Windows XP y para las nuevas versiones.

Versiónes de MIDI Una nueva versión de MIDI, llamada de manera tentativa “Protocolo HD” o “High-Definition Protocol”, fue anunciada como “HD-MIDI”. Este nuevo estándar ofrece retro-compatibilidad con el MIDI 1.0 y está planeado que soporte grandes velocidades de transmisión, permitir la detección de dispositivos con solo conectarlos, enumerarlos y ofrecer un gran rango de datos y resolución.

Los números de los canales y los controladores aumentaran, nuevos tipos de eventos serán agregados y los mensajes serán simplificados. Nuevos eventos serán soportados como Note Update y Direct Pitch que están enfocados a controladores de guitarra. Las capas físicas propuestas incluyen protocolos basados en Ethernet como RTP MIDI y Audio Video Bridging. El protocolo HD y un protocolo de transporte basado en User Datagram Protocol (UDP) están bajo la revisión de High-Definition Protocol Working Group (HDWG) de MMA, el cual incluye a los representantes varias compañías. Prototipos de dispositivos basados en las primeras fases del protocolo han sido mostrados de manera privada en NAMM usando tanto conexiones alámbricas como inalámbricas, sin embargo es incierto si el protocolo será retomado por la industria. En 2015, las especificaciones del protocolo HD están cerca de su finalización y MMA desarrolla las políticas de licencias y certificaciones de productos. Debido a que el costo de almacenamiento de datos ha disminuido, la

música MIDI se ha visto remplazada por audio comprimido en productos comerciales, haciendo nuevamente del MIDI una herramienta para la producción musical.

La conectividad MIDI y un sintetizador de software aún se incluye en Windows, OS X y iOS pero no en Android.

1.1.3. Baterías-MIDI y proyectos en entornos libre

Visto la historia del MIDI, para entender mejor el concepto de una batería-MIDI plantearemos la idea desde crear una batería-MIDI para ir desarrollando esta batería y luego compararla con las profesionales, de esta manera nos ayudara a entender mejor que problemas hay a la hora de desarrollar el reproductor de dtx.

Arduino

Para crear una batería simple, lo más fácil en la actualidad seria coger un microcontrolador como las placas de electrónica de Arduino.

La primera idea más básica es coger una entrada digital de 1 bit por simplicidad y si se activa hacer que el Arduino reproduzca un sonido. De esta manera podemos decir que tenemos una batería de 1 pad.

El primer detalle que se observaría es que como se ha escogido una entrada digital de 1 bit nuestra batería electrónica siempre sonaría con la misma intensidad, independientemente de la intensidad que se toque al pad.

Para solucionar esto desde los años 80 se han usado unos componentes electromecánicos conocidos como sensor piezoeléctrico que es un dispositivo que utiliza el efecto piezoeléctrico para medir presión, aceleración, tensión o fuerza; transformando las lecturas en señales eléctricas.

Por tanto pasando de 1 señal digital de 1 bit, a una entrada analógica ya se tiene la posibilidad de saber la intensidad con que se ha tocado el pad.

Poniendo un sensor piezoeléctricos y conectando este a una entrada analógica, nos faltaría modificar la programación del Arduino para modificar el volumen del sonido dependiendo de la intensidad que se observa en esta entrada analógica.

El añadido siguiente es pasar a “n” entradas. Un detalle a tener en cuenta si la programación esta realizada para que todo sea secuencial, (que es la manera básica de programar el Arduino) si

los sonidos a reproducir son un poquito largos observaríamos que esta batería no puede hacer sonar dos Pads al mismo tiempo.

Para resolver el problema del paralelismo a obtener en los sonidos, hay varias soluciones:

- Ir a una ejecución en paralelo de nuestra idea básica de comprobar las entradas y reproducir un sonido si hay una activa.
- Ir a micro-controladores más potentes, para que sea más rápido a la hora de comprobar las entradas y reproducir el sonido.
- Separar nuestra idea de batería en dos partes una parte sería nuestro Arduino que recoge las señales de los Pads, pero que no reproduce nada, y al tener una gran velocidad de muestreo nos podría parecer que si que esta comprobando las entradas en paralelo, y otra parte la que reproduciría los sonidos.

Como ya se ha observado en el protocolo MIDI, la idea de ir separando la generación de los eventos MIDI, con la reproducción de sonidos asociados a estos eventos MIDI, hace que pasemos a un siguiente paso de la creación de esta batería simple.

Baterías-MIDI

Ahora para continuar mejorando esta batería electrónica, lo que se debería hacer es coger un Arduino, con la posibilidad de programar una salida MIDI.

Si se escoge un Arduino de tipo más bajo se nos daría el siguiente problema, como solo tienen un usb que se emplea para programar, o ponemos un conector extra para que haga de salida MIDI, o perdemos la opción de programar en ese usb, mientras se esta usando como salida MIDI.

Para evitar esta problemática, lo más sencillo es coger un Arduino que si que tenga la opción tener un usb para comunicarse con un ordenador sin perder el usb para programar el Arduino.

Por lo que tenemos es una batería con “n” Pads con intensidad y conectándola a un sintetizador ya tenemos lo que sería una batería electrónica básica.



Figura 1.10: Ejemplo de batería electrónica de 4 pads basada en Arduino

Aquí ya podemos establecer las diferencias con las baterías profesionales.

Baterías-MIDI profesionales

Evidentemente el mundo de las baterías electrónicas profesionales es muy amplio, para los objetivos de este expediente nos centraremos en las grandes diferencias que hay de esta idea básica de batería electrónica, con las baterías profesionales.

Baterías profesionales de entrada Para representar las baterías electrónicas de entrada escogemos a la Yamaha DD-65.



Figura 1.11: Yamaha DD-65

Como se puede observar la mayor diferencia a simple vista con la batería que se ha creado con Arduino solo es el número de Pads, que esto hemos visto que tiene fácil solución.

Otra diferencia son los dos pedales, se podría pensar en la misma solución poner dos Pads más y ponerlos en el suelo para activarlos con los pies. No parece que sea un gran problema a no ser que recordemos como funciona una batería real.

El hihat es un instrumento de batería que tiene un pedal asociado, este instrumento suena distinto según su pedal las dos posiciones basicas del pedal son abierto o cerrado, por lo que en nuestra batería tendríamos que actuar en consecuencia, esto solo es un detalle de programación a la hora de enviar el evento MIDI asociado al pad del hihat, ya que se tiene que enviar un evento distinto si esta el pedal pulsado o no.

En esta batería se pueden conectar otros pedales para hihat y bombo que se asemejen más a unos reales.

Esta batería tiene 254 sonidos y 50 baterías predefinidas con ellos, de todas manera se puede configurar 3 baterías con cualquiera de esos 254 sonidos, pero no se puede modificar ninguno de estos sonidos, Yamaha con este tipo de batería quiere ofrecer al público una batería portátil e independiente que sea configurable por el usuario, y para que sea portable se ha de separar de lo que sería una batería acústica en cuanto al aspecto físico.

Yamaha con esta batería lo que quiere hacer es una batería electrónica por un precio mínimo.

Yamaha indica que esta batería es independiente por lo que se puede tocar sin tener que conectar a un ordenador. Aquí una de las mayores problemáticas que existe a la hora de afrontar que es una batería electrónica, ya que hay gente que entiende que se tiene que simular lo más cercano a una batería acústica, mientras que hay otro razonamiento que ya que se está delante de un instrumento electrónico se ha de aprovechar todas las posibilidades que esto nos da aunque se separe de lo que es una batería clásica.

Baterías semiprofesionales A mayor precio ya tenemos la gama de Roland V-Drums Lite



Figura 1.12: Roland V-Drums Lite HD-3

Se puede observar en esta batería la misma idea que la anterior pero físicamente más parecida a una batería real para que si alguien se acostumbra a tocar en esta batería, el cambio a una batería acústica será mucho menor.

En esta batería tiene 20 baterías predefinidas y no da la posibilidad de modificarlas en su propio sintetizador, Roland supone que si se quiere modificar la batería se hará a través del canal MIDI-out en otro sintetizador(u ordenador).

En esta batería físicamente no se pueden modificar los Pads ni los pedales ya que están soldados los cables que van al sintetizador.

Baterías profesionales A mayor precio ya tenemos la gama de Roland V-Drums T, ponemos por ejemplo la TD-30.



Figura 1.13: Roland TD-30

Se puede observar que han aumentado el número de platillos, el hihat es más parecido a uno real e incluso tiene 3 pedales para tener el doble bombo y el hihat, separados e independientes.

Pero evidentemente hay mucho más, internamente la calidad del sonido que reproduce por ella misma se ha ido aumentando.

Aunque los Pads que hacen de platillos parecen igual que los segundos y podríamos pensar que es la misma idea que nuestra batería construida con el Arduino, esta batería tiene en los platillos varios piezos para poder configurarlos de varias maneras. La más simple es que según se detecte el piezo más superior o el del borde sonara un sonido distinto, siendo más cercano a lo que sonaría en una batería acústica. Otra manera de configurar estos platillos es que si se toca el piezo más cercano al borde para el sonido de los otros piezos, imitando lo que pasa si se cogiera un platillo real con las manos, para hacer que deje de vibrar.

En esta batería se puede modificar tanto los samples que ha de usar cada pad o piezo.

Al tener un sintetizador totalmente independiente, también se puede modificar los pads o pedales, por otros más a gusto del percusionista.

Conclusiones Tal y como se ha ido exponiendo, se va a resumir las principales características por las cuales se puede diferenciar una batería-MIDI de otra y de esta manera tener un concepto más claro de que se podrá conectar a nuestro proyecto.

La principal idea es que se ha de aproximar en la medida de lo posible a una batería acústica, por lo que independientemente o no, se tiene que poder escuchar sonidos asociados a los impactos

que hagamos en los pads.

Las principales características de una batería electrónica a tener en cuenta este proyecto son:

- Todas tienen salida MIDI.
- Número de Pads y colocación de estos.
- Número de entradas dedicadas a los pedales.
- La capacidad de modificar los samples internos que tienen.
- La capacidad de modificar su funcionamiento interno.

1.1.4. Actualidad de Reproductores DTXMania en entornos libres

Se va a exponer el estado actual de los juegos DTXmania y DTXmaniaXG en entornos libres.

Todos estos programas están programados para la plataforma PC, pero entendiendo PC como Windows evidentemente plataforma x86 y como es un japonés el que los está programando usando características de Windows en Japón.

Desde hace bastante tiempo dejé el programa como software libre, la página web del programa se puede consultar en:

<https://osdn.jp/projects/dtxmania/>

El mismo autor tiene el programa DTXmaniaXG, para emular los programas DrumMania XG y Gitadora.

<https://osdn.net/projects/dtxmanixg-verk/>

La gran mayoría de la documentación estando en perfecto japonés.

Aunque siempre está la opción de máquinas virtuales, estamos hablando de juegos musicales donde tanto el tema de la latencia en eventos MIDI hacia el programa como la sincronización del sonido es muy importante, por lo que esta opción es válida mientras estemos pensando en ejecutarlo en una máquina bastante potente.

La otra gran opción es usar WINE, pero aunque se ponga las opciones de lenguaje en japonés, parece que no llega a crear bien o acceder a los archivos por problemas de codificación del lenguaje

japones y sistema de ficheros por lo que no se llega ni acceder al menú inicial, de cualquiera de estos programas (Versión probada de wine 2.5).

Pero también hay que pensar para otros entornos (arm/MIPS/RISCv), y ninguna de estas dos opciones son viables fácilmente para otras plataformas que no sean x86.

Hubo un proyecto antiguo llamado DigiBand el cual era un simulador/juego de baterías en entornos libres, que podía funcionar con canciones con el formato DTX, este programa tiene varios inconvenientes.

- En el 2005 fue la última vez que alguien modifico el código fuente.
- Emplea librerías que ya en 2005 estaban deprecated.

Se ha intentado compilarlo tanto en entorno x86-64, usando un fedora 25 para ello, como en un entorno arm usando una Raspberry. Y no se ha podido compilar, tanto por las dependencias antiguas que tiene como por los cambios en las librerías que se han sucedido en estos 12 años.

Otros juegos musicales

Como muchos juegos musicales, comparten la base, por lo que vamos a mostrar que existe en la actualidad para poder afianzar la base en la que trabajar.

Los juegos de Rock Band salieron para consolas pero no llegaron a salir para PC, por lo que ya no digamos plataformas en entornos libres.

Hay un juego que se considera el nuevo sucesor espiritual del Rock Band para PC, que es el PhaseShift misma idea pero gráficos más actualizados, usando como base:

- Windows XP/ Vista / 7
- DirectX 9.0c
- VC++ Redist

Y cuando se les pregunta a los autores de PhaseShift por MAC o linux, contestan:

- Q. Will you add support for Mac / Linux

- A. We do not have the resources to develop native versions, but the game is now compatible with Wine / Crossover.

Por lo que se deduce es que se podría llegar a tener en entornos “libres” usando WINE por lo que se puede entender que se pueden ejecutar si la maquina tiene la potencia suficiente.

El juego FoFiX como salio de un proyecto (Frets on fire) que era libre, no fue muy difícil modificarlo para que funcionar en entornos libres. En entornos x86 no tiene mayor inconveniente el problema esta en los entornos arm que al no soler tener OpenGL, no funciona. Indicar que en el entorno de la Raspberry pi, tanto en la 2 como en la 3 activando el driver beta OpenGL, comienza a funcionar, pero no se llega a ver todo bien y todavía hay muchos flecos a resolver en el driver de OpenGL.

El juego Dance Dance Revolution como hemos comentado antes si que tiene una versión libre, el StepMania en esa versión le pasa lo mismo que al FoFiX en plataformas x86, funciona bien pero cuando nos pasamos a entornos arm es cuando hay problemas con toda la parte gráfica ya que emplea OpenGL.

Aunque existe un juego llamado PyDance que emplea librerías de muy bajo nivel, lo que hace que tenga menús muy simplificados y poca carga gráfica mientras se juega a DDR se puede hacer funcionar tanto en entornos x86 como arm sin necesidad de tener OpenGL. Eso si esta pensado para canciones de tipo DDR no DTXMANIA. Emplearemos este programa como base para poder entender que tenemos que hacer para crear un reproductor de canciones .dtx en entornos libres.

Por lo que vemos el estado de los juegos musicales en entornos libres no es totalmente malo, pero son proyectos en los que la plataforma principal y casi se podría decir que única ha sido la x86, aunque dejando la puerta abierta a otras plataformas.

1.1.5. Raspberry

Como se ha observado casi siempre la plataforma principal ha sido la plataforma x86 para esto tipos de juegos musicales, ahora vamos a hacer un repaso a otra plataforma que esta cogiendo mucha fuerza en la actualidad, para ello expondremos el principal ordenador, con plataforma tipo arm que a nivel educativo esta cambiando la manera de entender los ordenadores.

Raspberry PI es un ordenador de placa reducida o (placa única) (SBC) de bajo coste, se podría decir que es un ordenador de tamaño reducido, del orden de una tarjeta de crédito, desarrollado en el Reino Unido por la Fundación Raspberry PI (Universidad de Cambridge) en 2011, con el objetivo de estimular la enseñanza de la informática en las escuelas, aunque no empezó su comercialización hasta el año 2012.

Aunque no se indica expresamente si es hardware libre o con derechos de marca, explican que disponen de contratos de distribución y venta con dos empresas, pero al mismo tiempo cualquiera puede convertirse en revendedor o redistribuidor de las tarjetas RaspBerry Pi, por lo que se entiende que es un producto con propiedad registrada pero de uso libre. De esa forma mantienen el control de la plataforma pero permitiendo su uso libre tanto a nivel educativo como particular.

En cambio el software sí es open source, siendo su sistema operativo oficial una versión adaptada de Debian, denominada RaspBian, aunque permite otros sistemas operativos, incluido una versión de Windows 10.

El concepto es el de un ordenador desnudo de todos los accesorios que se pueden eliminar sin que afecte al funcionamiento básico. Está formada por una placa que soporta varios componentes necesarios en un ordenador común y es capaz de comportarse como tal.

El diseño de la Raspberry Pi 1 u original incluye:

- Un Chipset Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz (el firmware incluye unos modos Turbo para que el usuario pueda hacerle overclock de hasta 1 GHz sin perder la garantía).
- Un procesador gráfico (GPU) VideoCore IV.
- Un módulo de 512 MB de memoria RAM (aunque originalmente al ser lanzado eran 256 MB).
- Un conector de RJ45 conectado a un integrado lan9512 -jzx de SMSC que nos proporciona conectividad a 10/100 Mbps
- 2 conexiones USB 2.0
- Una Salida analógica de audio estéreo por Jack de 3.5 mm.
- Salida digital de video + audio HDMI
- Salida analógica de video RCA
- Pines de entrada y salida de propósito general
- Conector de alimentación microUSB
- Lector de tarjetas SD

Historia

Este proyecto fue ideado en 2006 pero no fue lanzado al mercado febrero de 2012. Ha sido desarrollado por un grupo de la Universidad de Cambridge y su misión es fomentar la enseñanza

de las ciencias de la computación los niños. De hecho, en enero de este año Google donó más de 15.000 Raspberry Pi para colegios en Reino Unido. La Raspberry Pi, es una excelente herramienta para aprender electrónica y programación.

Los primeros diseños de Raspberry Pi se basaban en el microcontrolador Atmel ATmega644. Sus esquemas y el diseño del circuito impreso están disponibles para su descarga pública.

La fundación Raspberry Pi surge con un objetivo en mente: Desarrollar el uso y entendimiento de los ordenadores en los niños. La idea es conseguir ordenadores portables y muy baratos que permitan a los niños usarlos sin miedo, abriendo su mentalidad y educándolos en la ética del “ábrelo y mira cómo funciona”. El ideólogo del proyecto, David Braven, un antiguo desarrollador de videojuegos, afirma que su objetivo es que los niños puedan llegar a entender el funcionamiento básico del ordenador de forma divertida, y sean ellos mismos los que desarrollen y amplíen sus dispositivos. El co-fundador de la fundación es Eben Upton, un antiguo trabajador de la empresa Broadcom, el cual es el responsable de la arquitectura de software y hardware de la raspberry pi.

Eben Upton, se puso en contacto con un grupo de profesores, académicos y entusiastas de la informática para crear un ordenador con la intención de animar a los niños a aprender informática como lo hizo en 1981 el ordenador Acorn BBC Micro.

La fundación da soporte para las descargas de las distribuciones para arquitectura ARM, Raspbian (derivada de Debian), RISC OS y Arch Linux; y promueve principalmente el aprendizaje del lenguaje de programación Python, y otros lenguajes como Tiny BASIC, C y Perl.

El primer prototipo basado en ARM fue montado en un paquete del mismo tamaño que una memoria USB. Tenía un puerto USB en un extremo y un puerto HDMI en el otro.

En agosto de 2011, se fabricaron cincuenta placas Alpha del modelo inicial, el Model A (o modelo A). En diciembre de 2011, 25 placas Beta del modelo B fueron ensambladas y probadas de un total de 100 placas vacías.

Durante la primera semana de diciembre de 2011, se pusieron a subasta diez placas en eBay. Debido al anticipado anuncio de puesta a la venta a final de febrero de 2012, la fundación sufrió colapso en sus servidores web debido a los refrescos de páginas desde los navegadores de gente interesada en la compra de la placa.

El primer lote de 10.000 placas se fabricó en Taiwán y China, en vez de Reino Unido, con esto se conseguía un abaratamiento en los costes de producción y acortar el plazo de entrega del producto, ya que, los fabricantes chinos ofrecían un plazo de entrega de 4 semanas y en el Reino Unido de 12. Con este ahorro conseguido, la fundación podía invertir más dinero en investigación y desarrollo.

Las primeras ventas comenzaron el 29 de febrero de 2012 (Modelo B). Las dos tiendas que vendían las placas, Premier Farnell y RS Components, tuvieron una gran carga en sus servidores

inmediatamente después del lanzamiento. En los seis meses siguientes llegarían a vender 500.000 unidades.

El 16 de abril de 2012 los primeros compradores empezaron a informar que habían recibido su Raspberry Pi. El 22 de mayo de 2012 más de 20.000 unidades habían sido enviadas.

El 6 de septiembre se anunció que se llevaría la producción de placas al Reino Unido, a una fábrica de Sony y que en ella se producirían 30.000 unidades cada mes, y se crearían 30 nuevos puestos de trabajo.

El 4 de febrero de 2013, se lanzó el modelo A, que venía con solo 256Mb de RAM y sin puerto ethernet a un precio más asequible que el modelo B.

En diciembre de 2015 se pueden comprar modelos con mejores prestaciones; Raspberry Pi 2 Model B - Placa base (ARM Quad-Core 900 MHz, 1 GB RAM, 4 x USB, HDMI, RJ-45) de Raspberry Pi, manteniendo el precio de la Raspberry original.

En febrero de 2016 sale a la venta un nuevo modelo, la versión 3 con las siguientes características: ARM Quad-Core 1,2 GHz, 1 GB RAM, 4 x USB, HDMI, RJ-45 y una conectividad inalámbrica integrada de 802.11 b/g/n LAN y Bluetooth, se continua manteniendo el precio original.

Hardware

El modelo A solo tiene un puerto USB, carece de controlador Ethernet y cuesta menos que el modelo B, el cual tiene dos puertos USB y controlador Ethernet 10/100. En 2014 se lanzó el modelo Raspberry Pi 2 B. El modelo lanzado en 2015 es el Raspberry Pi Zero. Y en 2016 se ha lanzado el modelo Raspberry Pi 3 B.

A pesar que el Modelo A no tiene un puerto RJ45, se puede conectar a una red usando un adaptador USB-Ethernet suministrado por el usuario. Por otro lado, a ambos modelos se puede conectar un adaptador Wi-Fi por USB, para tener acceso a redes inalámbricas o internet. El sistema cuenta con 256 MB de memoria RAM en su modelo A, y con 512 MB de memoria RAM en su modelo B. Como es típico en los ordenadores modernos, se pueden usar teclados y ratones con conexión USB compatible con Raspberry Pi.

El Raspberry Pi no viene con reloj en tiempo real, por lo que el sistema operativo debe usar un servidor de hora en red, o pedir al usuario la hora en el momento de arrancar el ordenador. Sin embargo se podría añadir un reloj en tiempo real (como el DS1307) con una batería mediante el uso de la interfaz I2C.

La aceleración por hardware para la codificación de vídeo (H.264) se hizo disponible el 24 de

agosto de 2012, cuando se informó que la licencia permitiría su uso gratuitamente; antes se pensó en anunciarlo cuando se lanzara el módulo de cámara. También se puso a la venta la capacidad para poder usar el codificación-decodificación de MPEG-2 y Microsoft VC-1. Por otro lado, se hizo saber que el ordenador soportaría CEC, permitiendo que pudiera ser controlado mediante un mando a distancia de televisión.

El 5 de septiembre de 2012, se anunció una revisión 2.0 de la placa, que ofrecía un pequeño número de correcciones y mejoras, como unos agujeros de montaje, un circuito para hacer reset, soporte para depuración JTAG, etc.

El 15 de octubre de 2012, la fundación anunció que todos los Raspberry Pi Modelo B serían enviados a partir de ese momento con 512 MB de RAM en vez de 256 MB.

El “system on a chip” (SoC) usado en la primera generación de la Raspberry Pi es equivalente más o menos a los chips usados en los smartphones viejos (como el iPhone, 3G, 3GS). La Raspberry Pi esta basada en el SoC de Broadcom BCM2835 , que incluye un procesador ARM1176JZF-S a 700 MHz, una cpu gráfica VideoCore IV (GPU), y RAM.

La Raspberry Pi 2 tiene un Broadcom BCM2836 SoC con un procesador quad-core ARM Cortex-A7 a 900 MHz de 32-bit, con 256 KB de cache L2 compartida.

La Raspberry Pi 3 tiene un Broadcom BCM2837 SoC con un procesador quad-core ARM Cortex-A7 a 1.2 GHz de 64-bit, con 512 KB de cache L2 compartida.

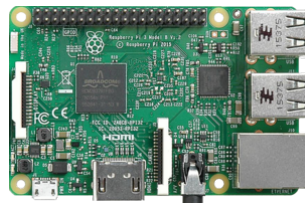


Figura 1.14: Raspberry Pi 3 B

Software

El Raspberry Pi usa mayoritariamente sistemas operativos basados en el núcleo Linux. Raspbian, una distribución derivada de Debian que está optimizada para el hardware de Raspberry Pi, se lanzó durante julio de 2012 y es la distribución recomendada por la fundación para iniciarse.

A la GPU se accede mediante una imagen del firmware de código cerrado, llamado blob binario, que se carga dentro de la GPU al arrancar desde la tarjeta SD. El blob binario está asociado a los

drivers Linux que también son de código cerrado. Las aplicaciones hacen llamadas a las librerías de tiempo de ejecución que son de código abierto, y estas librerías hacen llamadas a unos drivers de código abierto en el kernel de Linux. La API del driver del kernel es específica para estas librerías. Las aplicaciones que usan vídeo hacen uso de OpenMAX, las aplicaciones 3D usan OpenGL ES y las aplicaciones 2D usan OpenVG; OpenGL ES y OpenVG hacen uso de EGL y éste último, del driver de código abierto del kernel.

El 24 de octubre de 2012, Alex Bradbury, director de desarrollo Linux de la fundación, anunció que todo el código del driver de la GPU Videocore que se ejecuta en ARM sería de código abierto, mediante licencia BSD modificada de 3 cláusulas. El código fuente está disponible en un repositorio de la fundación en GitHub.

El 5 de noviembre de 2012, Eben Upton anunció el lanzamiento del sistema operativo RISC OS 5 para Raspberry Pi a la comunidad, pudiéndose descargar la imagen de forma gratuita desde la web de la fundación. Su relación con la comunidad RISC OS se remontaba a julio de 2011, cuando habló en ella de una hipotética versión.

El 24 de noviembre de 2012, se anunció en la Minecon de París, el juego Minecraft: Pi Edition para Raspberry Pi, basado en la versión Minecraft: Pocket Edition para teléfonos inteligentes y tabletas. La descarga se hizo disponible de forma oficial y gratuita por primera vez el 12 de febrero de 2013 desde el blog del juego, como versión 0.1.1 alpha, junto a instrucciones para ejecutarlo en Raspbian Wheezy. Una de las características principales de este lanzamiento es poder interactuar con el juego mediante programación, con la intención de motivar a los niños a aprender a programar.

El 25 de mayo de 2013, la fundación informó de que se estaba trabajando en una versión del servidor gráfico Wayland para Raspberry Pi, para sustituir al sistema de ventanas X. Con este cambio se lograría suavidad al usar la interfaz gráfica del escritorio, ya que el procesamiento lo realizaría la GPU Videocore y no la CPU, sin interferir en el renderizado 3D.

El 3 de junio de 2013, fue lanzado en la web de la fundación para su descarga la aplicación NOOBS (New Out of Box Software), utilidad que facilita la instalación de diferentes sistemas operativos para Raspberry Pi. Se distribuye en forma de archivo zip que se copia descomprimido a una tarjeta SD de 4 GB o superior, y una vez arrancada la placa con la tarjeta por primera vez, aparece un menú en que se da la opción de instalar una de las diferentes distribuciones en el espacio libre de la tarjeta de memoria, o acceder a internet con el navegador Arora integrado.

El 26 de septiembre de 2013, se añadió a los repositorios de Raspbian una versión oficial de Oracle Java JDK ARM con soporte para coma flotante por hardware, que ofrece bastante más rendimiento que la versión OpenJDK ARM ya existente hasta ese momento y más compatibilidad con aplicaciones. También se anunció que esta versión de Oracle Java JDK se incluiría dentro de la distribución en futuras versiones de Raspbian.

Accesorios

Cámara El 14 de mayo de 2013, la fundación y los distribuidores RS Components & Premier Farnell/Element 14 lanzaron la cámara para la Raspberry Pi con una actualización del firmware para que funcionara. Se conecta al puerto CSI de la placa mediante un cable plano flexible. El precio es de unos 20€, puede capturar video a 1080p, 720p y 640x480p. Sus dimensiones son de 25 mm x 20 mm x 9 mm.

Gertboard Este accesorio aunque no es de la fundación, ha tenido el visto de bueno de ellos, diseña con propósitos para la educación, expande los GPIO para poder interactuar con LED's, switches, señales analógicas, sensores y otros dispositivos. También incluye un controlador opcional para Arduino para poder interaccionar con el Raspberry Pi.

Cámara de infrarrojos A finales de octubre de 2013 se puso a la venta un módulo de cámara de infrarrojos, llamada la Pi NoIR.

HAT HAT (Hardware Attached on Top) , placas de expansión, junto al modelo B+, inspiradas en las placas de expansión del Arduino, el interface para las HAT fue definido por la fundación Raspberry Pi. Cada HAT tiene que tener una pequeña EEPROM(suelen llevar la CAT24C32WI-GT3) conteniendo los detalles importantes de la placa, de tal manera que el sistema operativo de la Raspberry Pi tiene el conocimiento del HAT, y de sus aspectos tecnicos importantes para el sistema operativo de como interactuar con el HAT. Las especificaciones mecanicas de un HAT, es que tiene que usar los cuatro agujeros en su formación rectangular.

De esta manera la fundación dejo como se podía ir ampliando el proyecto Raspberry.

Aunque hay muchas placas para expandir la Raspberry vamos a comentar brevemente tres que vemos muy interesante para este proyecto.

DAC La calidad del sonido obtenido por la Raspberry depende directamente de la tarjeta de sonido a la que conectemos, la Raspberry Pi tiene dos grandes opciones:

La primera y por defecto es la salida HDMI de esta manera es el monitor y su amplificador el responsable de la calidad final del sonido.

La segunda es la salida analogica que tiene la Raspberry, esta opción no tiene demasiada calidad, pero se ha de pensar en el precio de la Raspberry. Mucha gente se quejaba de que el sonido no

era de gran calidad cuando era la Raspberry la encargada de procesar el sonido para ello salieron varios HATs para que la Raspberry pudiera procesar sonido de alta calidad.

Estas soluciones suelen ser mejores que poner una tarjeta de sonido por USB ya que además de estar seguros que funcionara en la Raspberry, suelen emplear el protocolo I2S para reducir el uso de la CPU.

PianoHAT Aquí tenemos un HAT que integra 13 teclas en la disposición de un piano de una octava completa. Y 3 teclas más para modificar este piano.

Juntando esta HAT con una Raspberry se puede tocar el piano en la misma Raspberry. Los fabricantes de este HAT han publicado unos programas en el lenguaje Python para que este HAT saque comandos MIDI a través del USB y conectarlo a cualquier sintetizador.

Este HAT respeta las libertades del usuario, y tiene tanto el hardware como el software abierto y documentado para modificarlo como uno quiera.



Figura 1.15: Piano HAT

DrumHAT Pimoroni los fabricantes del PianoHat viendo su exito se decidieron a hacer lo mismo pero con una batería en vez de un piano de esta manera nació el DrumHAT.

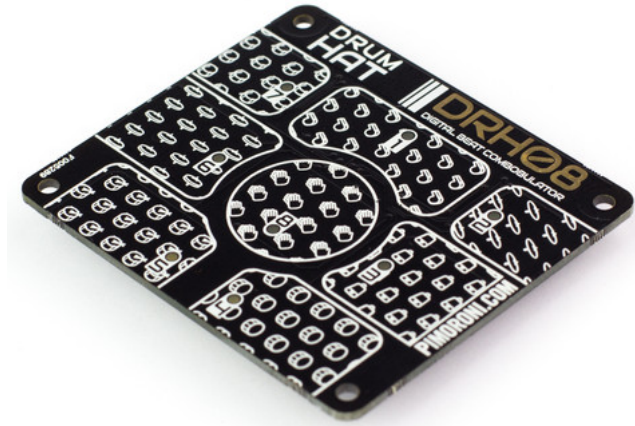


Figura 1.16: Drum HAT

Aunque ha tenido un relativo éxito, no ha tenido tanto como el pianoHat, recordar que el instrumento-MIDI por defecto es el piano, por lo que aunque en teoría es posible los fabricantes no han dado la opción para que el DrumHAT pueda sacar eventos MIDI.

Comunidad

La comunidad de Raspberry Pi fue descrita por Jamie Ayre de la compañía AdaCore ligada a FLOSS como una de las partes más excitantes del proyecto. El blogger de la comunidad Russell Davis, dijo que gracias a la fuerza de la comunidad, la fundación puede concentrarse en la documentación y la enseñanza.

Una serie de eventos llamados Raspberry Jam, dirigidos por Alan O'Donohoe han sido organizados en el Reino Unido, y en otras partes del mundo, para involucrar a la gente con la comunidad.

Desde mayo de 2012 es publicada mensualmente una revista gratuita llamada MagPi, que se dedica divulgar información acerca del Raspberry Pi, e incluye proyectos que se pueden desarrollar con él y cursos de programación de diferentes lenguajes. Es publicada mediante licencia Creative Commons BY-SA-NC.

El 2 de agosto de 2012, se añadió al foro oficial de la fundación los subforos en español, portugués y alemán (el de francés ya llevaba tiempo creado).

El 17 de diciembre de 2012, junto a la versión 2012-12-16-wheezy-raspbian de Raspbian, se lanzó la tienda de aplicaciones “Pi Store”, que en el momento de salida incluía desde aplicaciones como LibreOffice o Asterisk a juegos como Freeciv o OpenTTD. En esta plataforma se puede poner a disposición de todos los usuarios de Raspbian, mediante moderación y posterior lanzamiento, contenidos gratuitos o de pago, como archivos binarios, código python, imágenes, audio o vídeo. Además se quiere incluir documentación acerca del Raspberry Pi como la revista MagPi y tutoriales de proyectos.

Recepción e influencia

El escritor sobre tecnología, Glyn Moody, describió el proyecto en mayo de 2011 como un “potencial BBC Micro 2.0”, no para reemplazar a los ordenadores personales sino como algo suplementario. Alex Hope, coautor de Next Gen Report, sentía esperanzas de que el Raspberry Pi animaría a los niños a aprender a programar, en vez de a usar aplicaciones ya creadas. El coautor Ian Livingstone sugirió que la BBC podría involucrarse en el proyecto, con la posibilidad de hacerlo llamar “BBC Nano”. Chris Williams que escribe en The Register, ve la inclusión de lenguajes de programación como Kids Ruby, Scratch y Basic como un “buen comienzo” para dar a los niños habilidades que necesitarán en el futuro, pero que habrá que ver cómo de efectivo será su uso. El Centro de la historia de la computación da un fuerte apoyo al proyecto y sugiere que podría “marcar el comienzo de una nueva era”. Antes del lanzamiento del Raspberry Pi, la placa fue mostrada por el CEO de ARM, Warren East, en un evento en Cambridge, haciendo referencia a la filosofía de Google de mejorar la enseñanza de ciencias y tecnología de la computación en el Reino Unido.

Harry Fairhead sugiere que se tendría que dar más énfasis en mejorar el software educacional en el hardware actual, usando herramientas como Google App Inventor, para volver a enseñar programación en las escuelas, en vez de sacar nuevos sistemas.

En octubre de 2012 el Raspberry Pi ganó el premio T3 “Innovación del año”.

SonicPi

Aunque el formato MIDI es el más usado por los músicos me gustaría añadir un pequeño aporte, a nuevas formas de crear, editar y escuchar música en entornos libres.

Se ha creado el lenguaje musical SonicPi en el cual es más parecido a programar que a enfrentarse a una partitura clásica. El proyecto Raspberry Pi lo ha cogido como una de sus ramas para que los jóvenes aprendan música y esto se merece un pequeño punto en este proyecto.

Usos

En enero de 2012, encuestas hechas en el Reino Unido acerca de la penetración en las aulas de Raspberry Pi concluyeron que por cada placa que había en un colegio público, había cinco en colegios privados. Por ello se espera que en un futuro empresas patrocinen la adquisición de placas en colegios públicos.

El CEO de Premier Farnell declaró que el gobierno de un país de medio oriente expresó interés en proveer una placa a cada chica estudiante, con el objetivo de mejorar sus expectativas de empleo en el futuro.

A finales de enero de 2013, se dio a conocer que Google, junto con la ayuda de otros 6 socios, repartiría 15.000 placas entre estudiantes del Reino Unido que mostraran motivación por las ciencias de la computación.

Visto todo esto vemos ahora podemos definir el proyecto en mejores condiciones.

Capítulo 2

Proyecto de canciones DTXMANIA entorno libre

2.1. Planteamiento general

El planteamiento general es crear un reproductor de canciones del DTXMANIA, DTXMANIA HD y GITADORA en entornos libres.

Para ello se ha decidido usar tanto la Raspberry Pi 3 como principal plataforma arm con la distribución Raspbian y también la plataforma x86-64 con la distribución Fedora.

Para que el proyecto tenga más interés para los baterías reales este proyecto deberá mantener la principal características del DTXMANIA original, como poder conectar Baterías-MIDI y usar estas para tocar las canciones.

Ya que se ha optado para crear el reproductor de canciones dtx en arquitecturas x86 y arm, por lo que no deberíamos usar características específicas de una plataforma de esta manera se dejara abierto para que en otras posibles plataformas ya sea MIPS o RISCv respetando las librerías que se emplean, debería poder funcionar en ellas.

Todo el proyecto estará bajo la licencia GPL3.0, y se intentara que todas las librerías usadas también respeten estas libertades.

También se hará compatible con el Drum HAT de Pimoroni, para poder tener el proyecto conservando la filosofía de educación de la Raspberry Pi.

Por lo que los principales objetivos del proyecto se podrían resumir en los siguientes.

- Dar una posibilidad a los baterías a usar estas en entornos abiertos, como simulación/juego.
- Conseguir que la comunidad de los entornos libres continúe creciendo.

El reproductor de canciones dtx tendrá que conservar algunas de sus principales características.

Muy importantes:

- Poder conectar baterías-MIDI y configurar estas para su uso en el reproductor.
- Poder configurar la velocidad de sincronización de las notas.
- Poder configurar el reproductor para poder poner en automático algunas lanes por si la batería-MIDI no tuviera suficientes pads o por si no queremos/sabemos tocar esa parte.
- Poner una base para estudiar y mejorar la percusión en entornos abiertos.

Importantes:

- Si una canción tiene varias dificultades poder escoger que dificultad se quiere reproducir.
- Tener temas de interfaz, por si el interfaz por defecto no gusta.
- Tener una puntuación para saber lo bien o mal que se ha tocado.
- Se tiene que poder modificar el uso del reproductor de tal manera que si nuestra batería-MIDI tiene muchos pads y pedales se tiene que poder dejar de tal manera que el reproductor nos obligue a tocar lo más cercano a una batería acústica.

Se ha de tener en cuenta que al ser un proyecto muy enfocado a aspectos musicales, se tendrá que tener mucho cuidado con las latencias que se pueden observar en tiempo de ejecución, tanto a nivel de reproducción de sonidos, como a nivel gráfico, de todas maneras se pondrá siempre el nivel sonoro delante del nivel gráfico, ya que al ser un programa para percusionistas es el sonido lo más importante.

Se tendrá que ver hasta que punto la Raspberry puede con toda la carga y en alta calidad de algunas canciones del DTXManiaHD.

Entorno de desarrollo

Pasamos a definir que entorno de desarrollo emplearemos para el proyecto.

Parte física

Como componentes físicos para poder probar que el proyecto es funcional.

Para el proyecto se emplearan una Raspberry 3, como principal plataforma army y se probara sobre una distribución Fedora en arquitectura x86-64. Se tendrá un drumHAT para extender la capacidad de las Raspberry y de esta manera tener una batería muy simple en la Raspberry que se puede tocar con los dedos.

Como batería-MIDI se usara la Roland HD-1.

Python3

Se escoge Python y más concretamente Python3 como lenguaje para programar este proyecto y se exponen las ventajas y desventajas que nos aportara.

Es el principal lenguaje recomendado y apoyado por la fundación de Raspberry por su simplicidad, por lo que si lo escogemos continuaremos expandiendo la comunidad.

Como es un lenguaje interpretado, ya funcionara en plataformas x86 y arm entre otras sin tener que estar compilando para esa plataforma en concreto. Al ser un lenguaje interpretado en general sera más lento que si compiláramos el proyecto para la plataforma en cuestión.

Recordar que python3 no es un lenguaje paralelo aunque tiene algunas librerías para poder ejecutar código en paralelo. Por lo que como la Raspberry modelo 2 y 3 ya tiene cuatro cores, seria interesante ya intentar integrar el paralelismo en parte del código para mejorar latencias y sincronismos.

La gran ventaja de usar un lenguaje interpretado es la rapidez de prototipado de este, ya que fácilmente se puede tener un prototipo funcional para luego más adelante ir optimizando el código.

Recordar que hay implementaciones alternativas de Python3 para que el código sea compilado en tiempo de ejecución (JIT) para la plataforma en concreta, por lo que se programara todo el código de tal manera que se pudiera usar tanto el interprete de Python3 como estas implementaciones, la más famosa de ellas es PyPy (No confundir con PyPI el repositorio de paquetes de python).

Ninja-IDE

Como IDE para programar se usará el Ninja-IDE ya que además de darnos las herramientas de un IDE más o menos clásico, coloreado de syntaxis, facilidad para acceder a todos los ficheros del proyecto, etc.

Nos ofrecerá una gran ventaja característica de python ya que mostrará como warnings si no cumplimos un PEP de python (buenas maneras en el código). De esta manera se integrará con la Guía de estilo python que emplearemos, esta guía está como apéndice.

2.1.1. Comparaciones funcionales Stepmania/DTXMania

Como se ha expuesto anteriormente el pydance puede funcionar en una Raspberry por lo que cogeremos parte como base para nuestro proyecto, para ello compararemos primero las similitudes y luego las diferencias que podremos emplear.

Comparando los juegos musicales originales, para luego comparar las versiones de ordenador.

DDR/Drummania

Similitudes generales en los dos juegos Como se ha comentado los dos juegos son juegos musicales donde en cada nota se mira la sincronización para dar una puntuación. Hay una barra de vida asociada a la sincronización con cada nota, mientras esta barra de vida no baje hasta su mínimo se continúa con la canción.

En los dos juegos buscaron una manera fácil de poner una dificultad a las canciones, en DDR se usó la idea de indicar los BPM (Beats per minute) e indicar además una dificultad con un valor numérico del 1 al 100, en Drummania se indicó la dificultad de la canción con un valor numérico también del 1 al 100.

En los dos juegos puede haber videos, para hacer más ameno la reproducción aunque no es una característica que se tenga en cuenta en este proyecto.

Diferencias en los dos juegos En DDR como se ha observado hay 4 Lanes mientras que en la versión más básica del Drummania ya salió con 6 Lanes.

En versiones avanzadas del Drummania (Gitadora) recordar que hay Lanes que pueden ser dobles, el hi-hat es el mejor ejemplo ya que se tiene que tocar o cerrado o abierto, para no tener otro

lane se muestra en el mismo (ya que nunca se toca el hihat al mismo tiempo cerrado y abierto).

El DDR se puede juntar a otras maquinas DDR de la misma versión para poder jugar 2 personas al mismo tiempo o un modo especial de 1 jugador bailando en las 2 máquinas.

El DTXManiaHD en cambio se puede juntar con los GuitarFreaks para tocar varias personas al mismo tiempo una canción, ya en el Gitadora si que se pudo conectar contra otros Gitadora para poder competir dos percusionistas en la misma canción.

El DDR por ser un juego de baile tiene sentido tener que estar manteniendo los pies en alguna posición en concreto, cosa que en una batería no tiene sentido mantener alguna baqueta en contacto con el pad.

En el DDR normalmente solamente suena la canción principal y no hay sonidos extras o silencios dependiendo de lo bien o mal que se este bailando, mientras que en el Drummania toda la parte de la batería de la canción esta totalmente sincronizada a lo que se toca en la batería.

StepMania/DTXMania

En las versiones libres de estos juegos musicales tienen otras similitudes que las pasaremos a comentar a continuación. En los dos juegos hay opción de modificar toda la interfaz mediante temas.

La comunidad que se monto cuando por fin la gente tuvo estos juegos en su ordenador personal, de las primeras modificaciones que se pidieron a los dos programas era desactivar la opción de la barra de vida, ya que ahora no existía el interés económico, la gente quería aprender a bailar/tocar la canción y para ello les interesaba que aunque hubiera un momento muy dificil en la canción poder continuar la practica hasta el final para ir mejorando. Una segunda modificación referente a la manera de aprender fue pedir, la posibilidad de bajar la velocidad de las canciones para aunque no fuera correcto a nivel de sonido, aprender y una vez se estuviera cómodo en esta velocidad subirla hasta reproducir la canción tal y como tiene que sonar (Esta velocidad no es la misma que la de sincronización de las notas).

También se puso el detalle de tener canciones con varios niveles de dificultad, que es la misma canción pero una mayor complejidad para tocarla/bailarla, en baile es más fácil de entender, pero en las canciones para batería se puede entender como canciones sin arreglos solamente tocando los ritmos básicos y la canción con todos los arreglos posibles.

2.1.2. pydance

De la implementación libre del DDR, el pydance, y ya que esta implementada de tal manera que funciona directamente en la Raspberry, se empleara para tener una base sobre que librerías se pueden usar para este proyecto.

<https://icculus.org/pyddr/index.php>

Ya en la página oficial del proyecto indica que se ha realizado con python, Pygame , libsndl y Vorbis.

Explicaremos brevemente las tres librerías que acabamos de nombrar:

Pygame

Pygame es un conjunto de módulos del lenguaje Python que permiten la creación de videojuegos en dos dimensiones de una manera sencilla. Está orientado al manejo de sprites.

Gracias al lenguaje, se puede prototipar y desarrollar rápidamente. Esto se puede comprobar en las competiciones que se disputan online, donde es cada vez más usado. Los resultados pueden llegar a ser profesionales.

También puede utilizarse para crear otros programas multimedia o interfaces gráficas de usuario. Se basa en las librerías libsndl.

libsndl

Simple DirectMedia Layer (SDL) es un conjunto de bibliotecas desarrolladas en el lenguaje de programación C que proporcionan funciones básicas para realizar operaciones de dibujo en dos dimensiones, gestión de efectos de sonido y música, además de carga y gestión de imágenes. Fueron desarrolladas inicialmente por Sam Lantinga, un desarrollador de videojuegos para la plataforma GNU/Linux.

Pese a estar programado en C, tiene wrappers a otros lenguajes de programación como C++, Ada, C#, BASIC, Erlang, Lua, Java, Python, etc. También proporciona herramientas para el desarrollo de videojuegos y aplicaciones multimedia. Una de sus grandes virtudes es el tratarse de una biblioteca multiplataforma, siendo compatible oficialmente con los sistemas Microsoft Windows, GNU/Linux, Mac OS y QNX, además de otras arquitecturas y sistemas como Sega Dreamcast, GP32, GP2X, etc.

La biblioteca se distribuye bajo la licencia LGPL, que es la que ha provocado el gran avance y evolución de SDL.

Aunque a partir de la versión 2.0 esta librería se encuentra bajo la Licencia ZLib

Ogg Vorbis

Ogg vorbis forma parte del proyecto Ogg y fue llamado Ogg Vorbis o tan sólo ogg por ser el códec más comúnmente encontrado en el contenedor Ogg.

Vorbis es un códec de audio perceptivo de fines generales previsto para permitir flexibilidad máxima del codificador, permitiéndole escalar competitivamente sobre una gama excepcionalmente amplia de bitrates.

Aun cuando el algoritmo de compresión con pérdida produce una menor cantidad de información, es un procedimiento de codificación que tiene como objetivo eliminar una cierta cantidad de información considerada como irrelevante para disminuir el volumen de datos. Esta idea fue utilizada anteriormente y el ejemplo más famoso es la compresión de mp3.

En la escala de nivel de calidad/bitrate (CD audio o DAT-rate estéreo, 16/24 bits) se encuentra en la misma liga que MPEG-2 y Musepack (MPC) y comparable con AAC en la mayoría de bitrates. Similarmente, el codificador 1.0 puede codificar niveles de calidad desde CD audio y DAT-rate estéreo hasta 48kbps sin bajar la frecuencia de muestreo. Vorbis también está pensado para frecuencias de muestreo bajas desde telefonía de 8kHz y hasta alta definición de 192kHz, y una gama de representaciones de canales (monoaural, polifónico, estéreo, cuadrafónico, 5.1, ambisónico o hasta 255 canales discretos).

Ogg Vorbis es totalmente abierto y libre de patentes; la biblioteca de referencia (libVorbis) se distribuye bajo una licencia tipo BSD por lo que cualquiera puede implementarlo ya sea tanto para aplicaciones propietarias como libres. Aplicaciones de streaming audio como Spotify utilizan el formato Ogg en versión premium.

Código de pydance

Se puede observar que el pydance puede reproducir varios formatos ya que son muy parecidos, 3 Lanes (DDR Solo beginner), 4 Lanes (DDR), 5 Lanes (Pump It Up), 6 Lanes (DDR Solo), 8 Lanes (Technomotion), 9 Lanes, Dance ManiaX, EZ2 Dancer, EZ2 Real, y Parapara-style.

Por lo que estaría muy bien hacer lo mismo en nuestro proyecto, y coger los tres grandes estilos de juegos musicales para una batería-MIDI, el DTXMania ,el DTXManiaHD y el Gitadora como

principales y si se puede intentar soportar el Rockband y Rockband Pro. Más adelante veremos en detalle el estilo del Rockband que esta orientado hacia un juego y el estilo del DTXMania que es un simulador.

Si miramos las versiones que emplea son Python 2.4 y Pygame 1.8.1 , por lo que esta empleando las versiones de libSDL y de Ogg que estaban enlazadas en ese momentos con esas librerías.

Actualmente estamos en la versión Python 2.7.11 (2015-12-05) y las versiones de Python 3 son Python 3.4.4 (2015-12-21) y 3.5.1 (2015-12-07) considerándose la versión beta. Se aconseja ya no programar en Python 2 a no ser que se tenga mucho código en él.

Mientras que pygame a continuado avanzando y esta en la versión 1.9.2

En su manual <https://icculus.org/pyddr/manual.php> indica cosas que el pydance no puede hacer por lo que las tendremos que tener en cuenta, solo exponemos las generales que nos pueden afectar, no diremos ninguna en referencia a los juegos DDR.

- Reproducir MP3, indican que algunos se pueden reproducir pero otros dan fallos, aconsejan pasar estos a .ogg o a .wav.
- Algunos videos no funcionan.

Como podemos observar son fallos fácilmente derivados de DRM.

Algo que se ha observado mientras se analizaba el código del pydance es que indica los archivos que puede reproducir y para evitar los problemas con el DRM ha creado un formato nuevo que no tendrá este problema.

2.1.3. DTXMania

Ahora tenemos que pasar a comentar el principal objetivo del proyecto que seria reproducir las canciones .dtx, para ello se tiene que observar en detalle el reproductor DTXMania ya que este es el principal reproductor de estos archivos, y es la base para el DTXManiaHD.

Configuraciones

Recordar que el DTXMania inicialmente salio al inicio para solamente para las baterías Yamaha de la serie DTX, de hay su nombre DTXMania, ya que su programador tenia una de ellas y quería tener un emulador del DrumMania en su propia casa.

Cuando la gente lo vio fue pidiendo para que cogiera baterías-MIDI en general y que no implementara detalles de las Yamaha. Para ello se tuvo que dar la opción de poder configurar cada lane con una señal MIDI.

Una opción que se implemento en los inicios fue que su pudiera tocar independiente de la barra de vida, para convertirlo en un “medidor” de lo bien o mal que se tocaba una canción, justo con esto se añadió la opción de tener que tocar la canción exactamente como estaba escrita en el archivo o dejar la libertad de poder tocar cualquier Pads y que no fuera considerado un fallo.

Una opción interesante que añadió el programador del DTXMania, fue como su batería-MIDI en el hihat tenia la opción de dar dos señales, según tuviera pulsado el pie izquierdo, añadió la opción que algunos Lanes pudieran estar físicamente en la misma columna para ello el .dtx tenia que definir por separado esos dos sonidos diferentes, esto a los baterías les gusto mucho ya que les obligaba a tocar como se hace en una batería acústica, a la gente que jugaba con las baterías del Rockband no les gusto ya que no tenían esa posibilidad para ello el programador del DTXMania dio la posibilidad de configurar estos Lanes “dobles” con la misma señal, de esta manera si tu batería era más simple se podía continuar jugando y la mayor diferencia es que visualmente la nota que bajaba era distinta, pero se podía tocar esa canción, aunque no fue correcto según una batería acústica.

Los profesionales con baterías-MIDI con mayor calidad fueron viendo que este programa era lo que estaban buscando pero tenían un problema principal, como sus baterías-MIDI tenían muchos Pads y algunos con varias señales integradas en un mismo pad, cuando configuraban el DTXMania para poder usarlo con una canción, les molestaba mucho que aunque ellos sabían que tenían que tocar un pad, el DTXMania les obligaba a tocar solo una zona, ya que si tocaba la otra zona el DTXMania se lo indicaba como fallo y no sonaba nada, para ello se tuvo que dar la opción de poder tocar varias señales para un lane, de esta manera ellos si conocían la canción y tenían más Pads podían configurar varios de ellos y tocarlo exactamente igual que con una batería acústica.

A partir de estos primeros años los “grandes” añadidos fue ir añadiendo Lanes pero de tal manera que se podía configurar para juntarlos en un lane y dejarlo como en las versiones más antiguas, de esta manera se tenia contento a los baterías y a los jugadores con sus controles básicos.

Cuando llego por fin el DTXManiaHD además de actualizar las resoluciones gráficas empleadas, aquí se rompió un poco la compatibilidad hacia atrás por lo siguiente, el DTXManiaHD emulaba la máquina DrumManiaXG y como cambiaron el mueble físico de la máquina ya por definición se paso a 9 Lanes, pero existe un pequeño conflicto, se pusieron dos pedales, por el derecho no hay problema ya que siempre es el bombo, pero el izquierdo dependiendo de la canción cumple la función de abrir o cerrar el hihat o de doble bombo. En el DTXMania original como no tenia posibilidad de doble bombo, como ya se había implementado la opción de poner varias señales en la misma lane, la gente creaba canciones y se ponían todas las notas del bombo en la lane correspondiente, cierto es que no obligaba a usar los pasos correctos mientras se marcaran en el momento correcto, no le importaba que señal le venia. Para aclarar este concepto no es lo mismo Derecha-Derecha-Izquierda

que Izquierda-Derecha-Derecha. Ahora en el DTXManiaHD si que se codifica que pie se tiene que usar.

Si es una canción del DTXMania con bombo simple no hay problema el DTXManiaHD la pone en el pedal derecho y el resto de los Lanes donde correspondan. El problema esta cuando se coge un archivo codificado del DTXMania con doble bombo en el DTXManiaHD, las bases son las mismas pero al existir el problema anterior se puede optar por varias soluciones:

Duplicar el lane del bombo y ponerlo tanto en el izquierdo como en el derecho, ya que no podemos saber en que lane exacta debería ir. Esto puede tener problemas en las canciones que también se han de usar el hihat al mismo tiempo que el pie derecho, y además continua sin tener el concepto de izquierda o derecha ya que no obliga al usuario a diferenciarlos.

Poner el lane del bombo como esta codificado y configurar el lane del pedal derecho con dos señales tanto la del pedal izquierdo como la del derecho, aquí salta otro problema se acaba de duplicar la entrada del pedal izquierdo por lo que cuando se toca el pedal izquierdo para tocar el hihat, también se tiene que comprobar el lane del bombo, de esta manera estamos perdiendo la opción de obligar al batería a tocar exactamente un pad por lane, de esta manera estamos perdiendo la opción que seria más cercana a una batería acústica.

La opción que tomo el programador fue comentarlo y continuar manteniendo dos programas, la idea era que la gente fuera pasando las codificaciones al DTXManiaHD.

Menús fuera de la reproducción de las canciones

Una vez configurada la batería-MIDI todos los menús estaban pensados de tal manera que con los pads se podía navegar por todo el programa.

Un detalle que gusto bastante a la hora de navegar por las canciones es que los usuarios insertaban las canciones en una carpeta especifica y ya el reproductor miraba recursivamente todas estas carpetas.

Para tener agrupadas las canciones se añadió la posibilidad de poner un pequeño fichero de configuración en las carpetas para modificar varias cosas: El nombre que salia en el menú, tener una imagen para esa carpeta, y ya una opción muy poco usada fue que se podía configurar toda una carpeta con opciones de sincronización propios que eran más prioritarios que la configuración del DTXMania, de esta manera era muy típico tener una carpeta con ritmos de practica con necesidad de tocarlos con un mayor sincronismo y luego ya poder tocar las canciones más tranquilo sin tener que estar modificando todo el tiempo las configuraciones.

2.2. Formatos a nivel básico de las canciones en juegos musicales

Para continuar afianzando las bases de nuestro proyecto pasamos a explicar como están codificadas las canciones para baterías, se comenzara con los otros programas (FoFiX /PhaseShift), por simplicidad, luego se pasara al pydance ya que se puede entender como un .dtx primitivo ya que no tiene la parte del sintetizador del dtx y ya luego se pasara a explicar la base de un .dtx.

2.2.1. FoFiX / PhaseShift

Ahora se mirara el formato que ha empleado FoFiX y el PhaseShift para sus canciones con parte de batería para ver tanto la dificultad de integrarlo en el proyecto como sus ventajas/desventajas.

Si cogemos una canción del FoFiX que tenga parte de batería observaremos que en su directorio hay varios archivos.

Los más importantes son el song.ini, el notes.mid los .ogg, primero miraremos el song.ini

```
[song]
artist = Daisuke Ishiwatari
name = Holy Orders (Be Just or Be Dead)
album = Guilty Gear X (Heavy Rock Tracks)
delay = 0
year = 2001
diff_guitar = 5
diff_bass = 4
diff_guitar_coop = -1
diff_rhythm = -1
diff_drums = 4
diff_vocals = -1
diff_keys = 5
diff_bass_real = -1
diff_guitar_real = -1
diff_dance = -1
diff_bass_real_22 = -1
diff_guitar_real_22 = -1
diff_drums_real_ps = -1
diff_drums_real = 4
diff_vocals_harm = -1
diff_band = 4
multiplier_note = 116
```

```
pro_drums = True
genre = Metal
song_length = 191193
charter = GhostByob
banner_link_a = http://jcharting.wix.com/jrockband
link_name_a = J-Rock Band Project site
icon = jrb
```

Igual que en otros formatos vemos que en este formato hay un archivo donde indica la dificultad de la canción, en este caso para varios instrumentos, una duración de la canción y detalles como autor o título de la canción.

En el notes.mid esta la notación musical de esta canción para que el programa la interprete en notas que tiene que pulsar el jugador, en este .mid lo que se ha realizado es separar la canción en varias pistas que ya con sus nombres podemos deducir para que son:

- PART DRUMS - Notación para la batería.
- PART BASS - Notación para el bajo.
- PART GUITAR - Notación para la guitarra.
- PART VOCALS - Notación para la voz.
- EVENTS - Eventos asociados al juego.
- BEAT - Ritmo que ha de marcar el juego, se podría entender como los BPM o puntos de sincronización para que todo suene en el mismo momento.

El song.ogg que es obligatorio es el archivo musical donde tiene que ir la voz, el teclado y los instrumentos que no están definidos como tocarlos en esta canción.

Para los canciones de los instrumentos disponemos de las siguientes posibilidades:

- guitar.ogg - La guitarra en la canción.
- rhythm.ogg - El bajo en la canción
- drums.ogg - La batería en la canción.
- preview.ogg - Es la canción que se tocara en el menú de selección

Vemos que este formato ha usado el midi adaptándolo a las necesidades del juego, lo bueno es que se puede coger este midi ponerlo en un sintetizador y sonaría la canción, recordar que el midi “no tiene voz humana”, en el sentido de que el midi se define en base a bancos de sonido e indicamos que banco ha de sonar y en que nota.

En este formato para simplificarlo no han definido los bancos de sonido de los instrumentos si no que usan estos .ogg, como se han incluido estos .ogg que sonaran todos (menos el preview) y el del instrumento que estamos tocando, lo único que ocurre cuando se falla la nota es que bajan el volumen momentáneamente del ogg de nuestro instrumento.

Para una batería esto es un problema mayor ya que, la batería tiene muchos Pads por lo que no es raro tener que pulsar dos o tres Pads al mismo tiempo, si nos dejamos alguno sin tocar ese en particular no tendría que sonar, pero los demás si, como el sonido del instrumento se lo hemos indicado en un .ogg lo único que se puede hacer es bajar momentáneamente el sonido de este .ogg pero bajamos el sonido de toda la batería no solamente del pad que hemos pulsado mal. También esta el problema que no estamos educando al oído correctamente ya que independientemente de cuando pulsemos los pads, estarán sincronizados musicalmente con el resto de canción aunque el programa visualmente nos indique que es un fallo.

Como ventajas indicar que es relativamente fácil pasar cualquier canción ya que solamente poniendo el .ogg de la canción y haciendo un midi con la notas de la batería ya lo tendríamos, si queremos hacerlo mejor tendríamos que tener dos .ogg uno para la canción sin la batería y otro para la batería de la canción, pero al no separar todas las notas de la batería, no se puede independizar que suena o que no debe sonar justo en el momento de la ejecución.

2.2.2. MIDI

Hemos observado que el formato MIDI es el mayor empleado y estandarizado a nivel musical, como formato de notación para las canciones, aunque en esta parte tendremos que hacer un pequeño inciso. Y ser más específicos en la parte de percusión, ya que se va a exponer las desventajas para utilizarlo como notación para este proyecto.

Se ha de recordar que el formato MIDI para que fuera un formato pequeño tanto en memoria usada como en cpu, lo habitual es que un instrumento solo tenga un sample de sonido y según la nota tocada modificar ese sample mientras suena. Para sintetizadores potentes se usa un sample de sonido para cada nota, de esta manera la calidad es máxima.

MIDI ha pasado varias implementaciones, en su primera especificación ya en el canal 10 que es donde esta asignada la batería o percusión, tiene 8 posibles voces las cuales ya son distintas no como otros instrumentos, por lo que tocar el bombo ya es diferente en sonido que tocar el hihat o el snare. Esta primera implementación ya dicta cuales son estos 8 sonidos, en un sintetizador clásico

tiene los siguientes sonidos:

- 113 Tinkle Bell
- 114 Agogo
- 115 Steel Drums
- 116 Woodblock
- 117 Taiko Drum
- 118 Melodic Tom
- 119 Synth Drum
- 120 Reverse Cymbal

Por lo que vemos que estos sonidos no son los típicos de una batería de rock con múltiples piezas, aunque si se modifica el sintetizador tanto se puede modificar estos samples, como muchos otros detalles.

En la especificación “General MIDI level 2” en el canal 10 ya hay 128 voces disponibles.

Aquí es cuando vemos la problemática de usar el MIDI clásico para usarlo directamente en un simulador de una batería de 9 Lanes.

El MIDI 1.0 se queda justo en sonidos, cogiendo el MIDI 2.0 se tendría el siguiente problema, como hay 128 voces disponibles, quien haya creado la canción habrá usado la voz más cercana al instrumento que tiene que sonar, por lo que en nuestro simulador tendríamos varias opciones.

- Identificar solo 9 sonidos de estos 128 sonidos y asignarlos a nuestros lanes, y el resto de sonidos se tendría que sintetizarlos como un sintetizador MIDI normal, puede pasar que en la canción muchos sonidos los tendría que estar haciendo el batería y no sonando automáticamente.
- Hacer una relación de estos 128 sonidos a 9 lanes, lo que nos pasaría ahora es que muchas veces coincidirían varios sonidos en el mismo lane y no sabríamos que sonido tendríamos que reproducir cuando se tocara el pad asociado.

Para ello como ya veremos más adelante el DTXMania implementa al mismo tiempo la parte de reproducción de un sintetizador y nos obliga a detallar parte de esta información en el formato. Esta parte es que hace que el DTXMania sea un buen simulador de baterías ya que tiene la parte de notación, y de un sintetizador todo en el mismo programa.

2.2.3. pyDance

Cogiendo el formato específico del pydance, que ya está un poco saneado comparando con los del DDR clásicos, podremos ver algunas de las características de los formatos, en juegos musicales que ha realizado Konami.

Para ello ya en su página web hay canciones para bajar.

<https://icculus.org/pyddr/download.php>

Se puede observar que una canción está separada en varios archivos.

Un .ogg que como hemos comentado será la canción para que se escuchara mientras se está bailando.

Dos ficheros de imágenes un Background y un Banner.

Y el fichero que más nos preocupa que es un fichero .dance.

Abriendo el fichero forkbomb.dance, que es el más sencillo, ya vemos que tiene un preámbulo donde indica datos de la canción:

```
filename forkbomb.ogg
title Forkbomb
subtitle pydance training song
artist Pajama Crisis
author P2E and piman
banner forkbomb-banner.png
background forkbomb-bg.jpg
cdtitle pydance.png
bpm 136.0
end
```

Que gracias a la documentación se obtiene los siguientes datos de la canción.

- filename - Donde pone el archivo musical de la canción.
- title - Título de la canción.
- subtitle - Subtítulo de la canción.

- artist - Artista de la canción.
- author - El autor de la canción en pydance.
- banner - Imagen que se empleara en el menú de navegación de archivos.
- background - Imagen que se empleara ya cuando se esta jugando como fondo de pantalla.
- cdtitle - Imagen para el menú de navegación de archivos.
- bpm - Beats per minute de la canción, muy importante ya que nos definirá lo rápido que va la canción, recordar que los bpm es una unidad empleada para medir el tempo en música. Equivale al número de pulsaciones que caben en un minuto.
- end - Finaliza la zona del archivo para datos genéricos.

El archivo continua con los siguientes datos:

```
SINGLE
BEGINNER 0
R
D 8
L 7 Welcome to the pydance beginner tutorial.
L 6 (instructions will be here!)
D 6
L 7 See the platform under your feet?
D 6
L 7 When moving arrows cross over the ones at the top....
D 6
L 6 ....you hit corresponding arrows on the platform.
D 8
L 7 So, get ready to tap the left arrow..
D 3.75
L 7 ...Now!
D 0.25
h 1000
```

Se limpia la parte de avisos al jugador ya que en el DTXMania no existe y queda algo parecido a lo siguiente:

```
SINGLE
BEGINNER 0
```

```
h 1000
o 0001
o 0110
q 1000
q 0001
q 1000
q 0001
q 1000
q 0001
q 0010
q 1000
q 0100
q 0001
q 0010
q 1000
q 0100
q 0010
q 0001
end
```

Hay algunos datos más de la canción pero más referidos a como esta escrita la canción, por lo que esto se puede entender como el nivel de dificultad.

Vemos que juntando los bpm, la canción en formato .ogg y esta especie de partitura se comienza a entender lo que seria una versión simplificada del DTXMania. Siempre tendremos datos de la canción en un preambulo, datos que más concretos del juego musical y ya al final del archivo lo que se podría entender como una partitura de notas, o acciones a realizar en el juego musical.

2.2.4. DTXMania

Ya estamos en condiciones de estudiar el formato del DTXMania para ello, se va a plantear la siguiente manera de trabajar:

Se comenzara con una canción simple ignorando as dificultades del formato, para cuando ya tengamos un concepto general, pasar a mirar toda la documentación del formato y poder definir claramente como implementar un reproductor en entornos libres.

Usamos un .dtx que es de una canción para practicar ritmos simples, se pasa a indicar solo una parte del fichero, ya que luego se detallara las especificaciones completas del formato.

Como idea inicial se puede decir que indicando el tempo de la canción y poniendo una “partitura” en los lanes, se tendría lo que seria un canción delTXMania básica.

Esta canción de ejemplo no tiene una canción con otros instrumentos por lo que no se observa ningún .ogg.

Como antes ya vemos que se puede subdividir el fichero en dos grandes partes un preámbulo de datos y lo que seria la notación musical. En la primera parte ya podemos observar dos cosas importantes:

```
; Created by DTXCreator 019

#TITLE: Shuffle
#PANEL: Simple Shuffle
#BPM: 60.00
#DLEVEL: 45

#WAV01: bass_rock3.XA
#WAV02: snare_ambient.xa
#WAV03: snare_tap.xa           ;Snare Tap
#WAV04: ride_rock.XA          ;Ride - Hvy
#WAV05: ride_light.XA         ;Ride - Light
#WAV06: count1.xa
#WAV07: count2.xa
```

Ya se puede indicar dos detalles del formato .dtx:

- Con el punto y coma indicamos que son comentarios que el DTXMania ignora.
- y si usamos la almohadilla es un comando para que el DTXMania lo interprete.

vemos que se define:

- ; Created by DTXCreator 019 - Indica el programa que se ha usado para crear esta “partitura”
- #TITLE: Shuffle - Titulo de la canción
- #PANEL: Simple Shuffle - Subtitulo de la canción.

- #BPM: 60.00 - BeatsPerMinute de la canción
- #DLEVEL: 45 - Dificultad de la canción

Y una segunda parte donde define los sonidos que empleara en los chips.

Se observa que va asignando archivos .xa, estos archivos para definir los de una manera rápida son archivos de sonido codificados en menor calidad, ya que están pensados para ser pequeños tanto en tiempo como en tamaño.

Aquí uno de los grandes problemas de este reproductor en entornos libres, este formato de sonido es un formato japonés muy poco usado y casi sin documentación pero se ha mantenido desde los inicios del DTXMania, con el principal uso de que los samples de sonidos de las baterías ocuparan lo mínimo siendo todavía aceptables a nivel de calidad sonora, recordar que el DTXMania original es de cerca de 1999, y tanto la memoria para poner samples de gran calidad en la RAM del ordenador como para luego bajar las canciones de Internet eran cuestiones a tener en cuenta.

Más adelante ya se indicara como tratar este formato en entornos libres.

Se continua con la siguiente parte que es la notación musical:

```
#00061: 06070707
#00113: 01010101
#00116: 040000040005040000040005
#00161: 06070707
#00112: 00020002
#00216: 040000040005040000040005
#00261: 06070707
#00213: 01010101
#00212: 00020002
#00316: 040000040005040000040005
#00361: 06070707
#00313: 01010101
#00312: 00020002
#00461: 06070707
#00416: 040000040005040000040005
#00413: 01010101
```

Para entenderla mejor vamos a separarla, por datos tal y como los interpreta el DTXMania

```
#000 61: 06 07 07 07

#001 13: 01 01 01 01
#001 16: 04 00 00 04 00 05 04 00 00 04 00 05
#001 61: 06 07 07 07
#001 12: 00 02 00 02

#002 16: 04 00 00 04 00 05 04 00 00 04 00 05
#002 61: 06 07 07 07
#002 13: 01 01 01 01
#002 12: 00 02 00 02

#003 16: 04 00 00 04 00 05 04 00 00 04 00 05
#003 61: 06 07 07 07
#003 13: 01 01 01 01
#003 12: 00 02 00 02

#004 61: 06 07 07 07
#004 16: 04 00 00 04 00 05 04 00 00 04 00 05
#004 13: 01 01 01 01
```

Se definen los tres primeros números como números en sucesión que se tocaran en orden, (el número del compas) tendrá un tiempo de duración según los bpm que le hemos indicado antes, para el ejemplo supongamos cuatro segundos.

Los segundos números son los lanes del DTXMania y por último los números son los samples que se han definido antes, que se distribuirán en los cuatro segundos que tiene cada compás.

Ejemplo para concretar ideas.

```
#000 61: 06 07 07 07
```

Con esta instrucción el DTXMania lo que interpreta es que nos pondría en el lane asociado al canal 61, supongamos hihat el sonido 6 una vez y el sonido 7 tres veces. Por tanto mostraría cuatro barras bajando por ese lane espaciadas 1 segundo entre ellas y si pulsamos en orden correcto, escucharíamos el count1.xa una vez y el count2.xa tres veces.

Como idea básica ya tenemos lo que es el formato DTXMania/DTXManiaHD.

Capítulo 3

Posibles configuraciones en el DTXMania

Ahora que ya tenemos claro que es un .dtx, se tiene que indicar las opciones que se podían configurar en el DTXMania independientemente de las canciones, para indicar si lo tendremos que implementar:

Para ello separaremos en tres tipos las configuraciones de interfaz.

- Configuraciones a implementar
- Configuraciones para mejorar en un futuro
- Configuraciones que se escapan al objetivo del proyecto

Separarlas en tres grupos sera una norma en todo el proyecto tanto cuando se exponen características de los formatos a implementar o de las opciones que se tenían en los reproductores de estos formatos.

Para ello nos fijamos en las opciones que se indica en la página web http://www.dtxmania.net/wiki.cgi?page=qa_options_e

También indicar que ya se implementara una copia del interfaz de gitadora ya que se hizo un gran trabajo de limpieza comparado con el DTXManiaHD.

3.1. Configuraciones de Interfaz a implementar

3.1.1. DARK

Dan las opciones de OFF, estaran activos todos los elementos gráficos durante la reproducción: lanes, fondos de pantalla, video de fondo, barra de vida e iconos de instrumentos. Half donde no se vera el wallpaper, los Lanes coloreados se verán en negro y la barra de vida tampoco. Full donde además desaparecen los iconos de los instrumentos.

Esta opción también se podría indicar como a mejorar en un futuro, ya que solamente se implementa el estilo Full, por ser la más empleada.

3.1.2. Autoplay

En el DTXMania se podía configurar cada lane como AUTO, de esta manera el DTXMania se haría cargo de tocar el sonido cuando hiciera falta, sin aumentar la puntuación ni bajar la barra de vida, esto era muy útil para no tener que tocar el bombo que al principio es lo más difícil y preocuparse solamente del ritmo de las baquetas, y además ofrecía la posibilidad de en baterías con menos pads poder jugar ignorando algunos lanes.

3.1.3. ScrollSpeed

Esta es una opción obligatoria a implementar ya que esta es la velocidad con que se mide la sincronización a mayor velocidad pongamos más precisos tenemos que ser.

3.1.4. Tight

Marca un miss si no hay Chips en el lane concreto que se haya tocado, con esta opción nos obliga a tocar lo que muestra el DTXMania, es una opción que estaría bien implementar en un futuro pero entra en conflicto con el número de Pads que se tenga en la bateria-MIDI y que versión de canción estamos tocando, esto se detallara más adelante.

3.2. Configuraciones de Interfaz interesantes para un futuro

3.2.1. InputAdjust

Ajusta el tiempo de entrada para modificar el posible lag algunas baterías-MIDI, es una opción interesante pero no la veo obligatoria.

3.2.2. PreviewSoundWait

Se puede configurar el tiempo cuando comenzaran a sonar la música de preview de las canciones en el menú de selección, esta opción la dejamos como opcional para un futuro.

3.2.3. Fullscreen

Opción que se dejara para poder escoger si la aplicación funciona en fullscreen o en modo ventana.

3.2.4. Sudden

Quita los Chips de los Lanes hasta que están justo encima de las barras de esta manera obliga al batería a conocerse la canción de memoria.

3.2.5. Hiden

Quita los Chips cuando están a punto de caer en la zona de hit, desaparecen de esta manera el batería tiene que tocar de oído y no estar pendiente de la pantalla.

3.3. Configuraciones de Interfaz que se escapan al proyecto

3.3.1. ComboPosition

Donde estará el marcador de combo, encuentro que no hace falta implementarlo ya que se limpio la interfaz en el Gitadora.

3.3.2. Reverse

Las notas en vez de ir de arriba a abajo van al revés, no se observa el intereses en implementar esta opción, ya que en entornos abiertos suele haber bastantes opciones para girar la pantalla.

3.3.3. Position

Indica donde saldrán las notas de juicio (Perfect, Great, Good, Miss) con la limpieza de la interfaz del Gitadora ha perdido mucho sentido.

3.3.4. Guitar/Bass

El DTXMania tenia la opción de tocar la guitarra o el bajo estilo Konami, son opciones que para el batería no tienen sentido.

3.4. Configuraciones de uso a implementar

Detallamos las opciones de como se podía comportar el DTXMania cuando este reproducía una canción, se indicaran las más importantes.

3.4.1. Drums Key Assign

Opción indispensable para poder asignar teclas/Pads a los Lanes.

3.4.2. HH Group

Opción con cuatro modalidades que configuran la manera de trabajar en los Lanes del hihat, esta opción es para la gente que tiene baterías-MIDI simples con menos Pads que los Lanes en la canción.

- HH-0 Opción en donde los Lanes son independientes.
- HH-1 El lane Left Cymbal se integra en los lanes HiHat cerrado/abierto de esta manera hemos quitado un lane.
- HH-2 El lane asignado al hihat cerrado o abierto se puede tocar indistintamente.
- HH-3 Los tres lanes anteriores se pueden tocar indistintamente.



Figura 3.1: Grupo HH

Obligatoria. Ya se detallara como se ha resuelto esto en el proyecto.

3.4.3. HH Priority

Aquí definimos que ha de sonar cuando tenemos Lanes (HH-1, HH-2 o HH-3) integrados el Chip que esta en la canción o el pad que hemos tocado. .

3.4.4. FT Group

Opción con dos modalidades que configuran la manera de trabajar en los Lanes del tom de la izquierda, esta opción es para la gente que tiene baterías-MIDI simples con menos Pads que los Lanes en la canción.

- FT-0 Opción en donde los Lanes son independientes.
- FT-1 Los dos Lanes anteriores se pueden tocar indistintamente.



Figura 3.2: Grupo FT

Obligatoria. Ya se detallara como se ha resuelto esto en el proyecto.

3.4.5. FT Priority

Aquí definimos que ha de sonar cuando tenemos Lanes (FT-1) integrados el Chip que esta en la canción o el pad que hemos tocado.

Obligatoria. Ya se detallara como se ha resuelto esto en el proyecto.

3.4.6. CY Group

Opción con dos modalidades que configuran la manera de trabajar en los Lanes del platillo de la derecha, esta opción es para la gente que tiene baterías-MIDI simples con menos Pads que los Lanes en la canción.

- CY-0 Opción en donde los Lanes son independientes.
- CY-1 Los dos Lanes anteriores se pueden tocar indistintamente.



Figura 3.3: Grupo CY

Obligatoria. Ya se detallara como se ha resuelto esto en el proyecto.

3.4.7. CY Priority

Aquí definimos que ha de sonar cuando tenemos Lanes (CY-1) integrados el Chip que esta en la canción o el pad que hemos tocado.

Obligatoria. Ya se detallara como se ha resuelto esto en el proyecto.

3.5. Configuraciones de uso para un futuro

3.5.1. BGA

Opción para activar o desactivar las animaciones de Back ground.

3.5.2. FillIn

Opción para añadir efectos gráficos cuando se da a una nota en el momento justo.

3.5.3. DebugInfo

Opción que muestra información de frames per second, tiempo trascurrido y total de la canción.

3.5.4. HitSound

Opción que activa o desactiva el sample configurado en cada lane, se suele emplear para cuando el batería ha de escuchar el resto del grupo y no lo que esta tocando él.

3.5.5. SaveScore

Opción para guardar la puntuación si es la mejor que hemos realizado hasta la fecha, esta opción es interesante ya que nos hace ver la evolución que se va teniendo a medida que se va tocando la batería, aunque sería más interesante poder tener un historial para poder analizar la mejora, y en que parte de una canción solemos fallar o estar menos sincronizados.

3.5.6. D-MinCombo

A partir de que número de combo se muestra el numero de combo actual.

3.6. Configuraciones de uso que no se implementaran

3.6.1. AVI

Para ver los videos o desactivar esta opción, se considera que se escapa al objetivo del proyecto ya que la Raspberry con tema de codecs puede tener mucha problemática, y los problemas relacionados con los DRM.

3.6.2. Guitar/Bass

El DTXMania tenia la opción de tocar la guitarra o el bajo estilo Konami, son opciones que para el batería no tienen sentido.

3.6.3. StageFailed

Es una opción que se tendría que mantener para respetar la dualidad juego/simulador si la activamos se comporta como un juego y si baja mucho la barra de vida, sale al menú de selección

de canciones.

La ignoramos ya que no estamos centrando en el DTXMania como simulador/entrenador más que como juego.

3.6.4. BGMSound

El DTXMania se podía configurar de tal manera que no había música de fondo y de esta manera solo se escuchaba la batería, es una opción que no suele tener mucho sentido ya que la mayoría de baterías-MIDI tiene una salida de altavoces donde se escucha la batería en sí.

3.7. Otras configuraciones

También existía la opción de crear un archivo con unas características especiales en los directorios de las canciones para conseguir los siguientes beneficios.

Estructura de un archivo box.def de ejemplo:

```
#TITLE: FROM DTX Collection
#ARTIST: FROM
#COMMENT: Dazzling floor-tom sequences.

#PREIMAGE: preimage.jpg (or #PREMOVIE: premovie.avi)
#PREVIEW: preview.wav
#FONTCOLOR: #FFFFFF
#PERFECTRANGE: 34          (Release 068 (080427) or later)
#GREATRANGE: 67           (Release 068 (080427) or later)
#GOODRANGE: 84            (Release 068 (080427) or later)
#POORRANGE: 117           (Release 068 (080427) or later)
```

Se podía obtener varios beneficios:

- Tener una imagen y información para el directorio de algunas canciones.
- Una sincronización específica para un directorio, de esta manera era muy típico tener una carpeta con ritmos básicos para practicar.

Seria interesante en un futuro analizar estos archivos.

Capítulo 4

Formato xa

Anteriormente se vio que el DTXMania emplea el formato xa, se tiene que explicar que es este formato y dar una posible solución para el proyecto. El formato .xa es un formato de sonido que usaron en BandJAM (programa japonés de música, hace mucho en desuso) pero desde el inicio del DTXMania ha usado este formato hasta la actualidad.

Para ello el autor del DTXMania contacto con el autor original para usar este formato, añadió una pequeña documentación en inglés y no ha habido cambios desde entonces.

Las fuentes de este formato están disponibles en http://www.dtxmania.net/wiki.cgi?page=qa_utility_e

Más concretamente donde se indica donde se puede bajar el xadec.dll asociado a este formato E indica:

“The original one is made by Shinichi Nakamoto (the author of BandJAM), and I (yyagi and staff) add English documentations.”

Aquí ya indica que esta librería es para uso del formato xa, usado en GDAC2 DTXC y DTXMania. Aunque solo hemos pasado por encima del formato del DTXMania ya hemos podido observar que es un archivo de texto, pero para dar facilidad a los músicos para que usaran este formato hay herramientas para creación de canciones en el formato DTXMania, esto sale de la idea de este proyecto. Estas utilidades para crear canciones para facilidad de los creadores de canciones también usan este .dll.

4.1. Problemas

Como primera opción para resolver este problema, observemos que hizo el único programa que existe para entornos libres, que actualmente no se puede compilar ya que usaba librerías que actualmente están deprecated, el Digiband mencionado anteriormente dentro de su código existen zonas del código que se compilaran dependiendo de la plataforma donde estemos compilando linux o Windows, pasamos a detallar esto.

Dentro del Digiband en el archivo decodexa.C nos encontramos el siguiente código.

```
/*Oh... nothing like using a library that you have no idea how it works
and no idea what to do with the data it gives you and the only documentation
you have is a example program.
Well at least this way I was able to 'hack' it and figure out how it works by
poking its header file. Thank you bandjam.net for allowing public use of the
codec. It's just to bad the codec has a round about usage.
*/
#ifndef LINUX
bool decodexa(char soundfile[300])
```

Por lo que se podía decodificar estos archivos si se estaba en Windows pero no en Linux.

En la documentación que se puso en aquellos tiempos para poder compilar y crear el xadec.dll se observa lo siguiente:

```
#ifndef      _XADEC_H_
#define      _XADEC_H_

/*-----*/
#include      <windows.h>
#include      <windowsx.h>
#include      <mmsystem.h>

/*-----*/
#define      _XAID (ULONG)(( '1' << 24) | ( 'D' << 16) | ( 'W' << 8) | 'K' )
/*-----*/
typedef struct _XASTREAMHEADER {
    UCHAR *pSrc;
    ULONG nSrcLen;
    ULONG nSrcUsed;
    UCHAR *pDst;
    ULONG nDstLen;
    ULONG nDstUsed;
} XASTREAMHEADER;

typedef struct _XAHEADER {
    ULONG id;
    ULONG nDataLen;
```

```

        ULONG nSamples;
        USHORT nSamplesPerSec;
        UCHAR nBits;
        UCHAR nChannels;
        ULONG nLoopPtr;
        SHORT befL[2];
        SHORT befR[2];
        UCHAR pad[4];
    } XAHEADER;

typedef HANDLE HXASTREAM;

```

Donde el autor emplea código asociado a la plataforma Windows para definir bastantes cosas de este formato, ya que aunque se podría pensar que solo poniendo los tamaños correctos se podría compilar, también usa detalles internos del Windows para tener detalles del sonido.

Esta opción se valoro para hacer que el reproductor de DTXMania en entornos libres, funcionara indistintamente de los formatos internos del DTXMania pero tal además de los problemas citados anteriormente no queda bien integrar formatos cerrados en entornos libres.

4.2. Solución aportada al proyecto

Para dar una opción viable en entornos libres, se paso a analizar las utilidades para crear canciones con el formato DTXMania, estas utilidades obligan a tener este xadec.dll (todas las utilidades para crear .dtx están en Windows y japones) y también indican que podemos tener la utilidad xa.exe para poder transformar de .wav a .xa.

El formato DTXMania desde los inicios acepta .xa o .wav indistintamente lo que antes por consumo de memoria a la hora de reproducir la canción y por tamaño a la hora de pasarlo vía Internet se aconsejaba usar los .xa.

Para no perder todas las canciones que si que tienen un o varios .xa lo que se ha comprobado es que esta utilidad para transformar de wav a xa, también puede hacer el sentido inverso.

Se vuelve a a tener un problema esta utilidad evidentemente esta pensada para Windows. Lo bueno es que al ser tan simple (es una utilidad vía linea de comandos), se ha probado mediante wine y ha funcionado.

Por lo que con un pequeño script, podemos pasar de canciones DTX “no libres” a canciones totalmente “libres”.

```

find . -type f -iname *.dtx -exec sh -c 'iconv -f $(file -bi "$1" | sed -e "s/.*[ ]
charset=//") -t utf-8 -o converted "$1" && mv converted "$1" ' -- {} \;

```

```

find . -iname '*.xa' -execdir wine ../xa122/xa.exe -d -u '{}' \;
find . -iname '*.dtx' -execdir sed -i -e 's/.xa/.wav/g' '{}' \;
find . -iname '*.dtx' -execdir sed -i -e 's/.XA/.wav/g' '{}' \;
find . -iname '*.dtx' -execdir sed -i -e 's/\\\\\\\\/g' '{}' \;
find . -type f -iname '*.mp3' -exec bash -c 'FILE="$1"; ffmpeg -i "${FILE}" -acodec
libvorbis "${FILE%.mp3}.ogg";' _ '{}' \;
find . -iname '*.dtx' -execdir sed -i -e 's/.mp3/.ogg/g' '{}' \;

```

Los trabajos realizados mediante este script son los siguientes:

- Convertir los ficheros .dtx desde su codificación de origen a UTF-8.
- Usar el programa xa de tal manera que convierta los .xa a .wav, con el comando find ya coge tanto los .xa como los .XA.
- Transformar internamente los .dtx para que apunte a los nuevos .wav.
- Transformar el caracter especial de directorio de windows al de linux.
- Convertir los ficheros de .mp3 a .ogg, y canviar los .dtx en consecuencia.

Indicar que esto rompe un poco el espíritu del proyecto ya que esta transformación solo la podemos hacer o en un Windows o con este script en un linux x86, luego una vez transformado si que funcionaria en cualquier plataforma tanto x86 como arm.

Capítulo 5

Formato DTXMania al completo

Pasamos a indicar las especificaciones formales dentro del formato DTXMania, para ello vamos a la página web http://dtxmania.net/wiki.cgi?page=qa_dtx_spec_e

5.1. Estructura básica

Como ya se ha visto anteriormente la estructura básica de un archivo .dtx se compone de dos parte cabecera y objetos.

En la cabecera ira la información genérica de la canción y en la parte de los objetos toda la notación musical.

Todos los comandos se definen en lineas.

- DTXMania ignora todas las lineas que no comienzan por #
- DTXMania ignora las lineas que no tengan un descriptor definido.

Los comentarios dentro de los DTX son a partir del carácter ; hasta el final de la línea.

Se aconseja que no se usen a mitad de la linea ya que antiguamente un creador de dtx no lo soportaba por lo que se podía tener problemas de abrir archivos .dtx en ese creador y que hubiera problemas.

Los formatos de sonido soportados son .wav, .xa, .mp3 y .ogg, ya se ha solucionado el posible

problema con los .xa y los .mp3 desde hace un tiempo ya aconseja el propio autor del DTXMania por tema de licencias usar el .ogg antes que el .mp3.

Los ficheros de imágenes soportados por el DTXMania son el .bmp, .png y .jpg.

Los ficheros de video indica que soporta tanto los .VFW (video for Windows) o AVI, recordar que no es el objetivo principal de este proyecto el tratar ficheros de video.

Se tiene que detallar todos los comandos documentados para nuestro reproductor de .dtx, se separan en tres grandes bloques: a implementar, detallar para un futuro o no implementar por salir de la idea del proyecto.

5.2. Cabecera

Cuando en la documentación se indica zz lo que se define es decimal más letras en mayúsculas por lo que se tiene las siguientes posibilidades (01-ZZ)

5.3. Comandos en cabecera a implementar

#TITLE

#TITLE <song title>

#PANEL

#PANEL <comments on playing screen>

Es un comentario que se vera con un scroll mientras se juega la canción, se suele usar en las canciones para entrenamiento indicar como se tienen que tocar.

Si no existe este parametro se usa #TITLE

#DLEVEL

#DLEVEL <level value>

Dificultad de la canción en batería.

#BPM

#BPM <the value of BPM>

El valor del BPM al inicio de la canción, se puede usar decimales.

#BPMzz

#BPMzz <the value of BPM>

Posible valor de BPM, durante la canción se modificara mediante el canal 08 apuntando al valor de la “nota de cambio” zz.;

#BASEBPM

#BASEBPM <the value of BASEBPM>

El valor base de las BPM al cual se añadirán o restaran valores para modificar los BPM durante la canción.

#PREIMAGE

#PREIMAGE <preview image filename>

Imagen que se mostrara en la pantalla de selección.

#WAVzz

#WAVzz (WAV, MP3, XA or OGG) filename>

Define el sonido del banco zz. Para que nuestro reproductor del DTXMania sea lo más libre posible recomendamos solamente WAV o OGG.

Cada canal de batería tiene dos-polifonías posibles.

5.4. Comandos en cabecera para un futuro

Define el nombre de la canción.

#ARTIST

#ARTIST <artist name>

Define el nombre del artista.

#GENRE

#GENRE <genre name>

Define el género de la canción.

#COMMENT

#COMMENT <comments of yours>

Pone un comentario.

#STAGEFILE

#STAGEFILE <now loading image filename> Se puede usar archivos en BMP, JPEG o PNG. Y se mostraran en el momento de cargar la canción.

#BACKGROUND

#BACKGROUND <wallpaper filename>

Wallpaper que se usara mientras se reproduce la canción.

#WALL

#WALL <wallpaper filename>

Lo mismo que background.

#PREVIEW

#PREVIEW <preview sound filename>

La música que sonara como muestra en el menú de selección.

#BMP

#BMP <BGA image file>

Pone la imagen de fondo inicial.

#SOUND_NOWLOADING

#SOUND_NOWLOADING <nowloading sound filename>

Define el sonido que se escuchara mientras estamos en la pantalla de “now loading”.

#SOUND_STAGEFAILED

#SOUND_STAGEFAILED <stage failed sound filename>

Define el sonido que se escuchara mientras estamos en la pantalla de “stage failed”.

#SOUND_FULLCOMBO

#SOUND_FULLCOMBO <full combo sound filename>

Define el sonido que sonara si hacemos un full combo.

#RESULTIMAGE

#RESULTIMAGE <result screen image filename>

Imagen que se mostrara al final de la canción. Si no esta definido se usara #PREIMAGE. También se puede usar #RESULTIMAGE_xx que puede cambiar la imagen resultante dependiendo de que rango se obtenga al tocar la canción.

```
#PREIMAGE resimage0.jpg
#RESULTIMAGE_A resimage1.jpg
#RESULTIMAGE_D resimage2.jpg

;Si tu rango es...
;-SS, S or A: resimage1.jpg
;-B, C or D: resimage2.jpg
;-E: resimage0.jpg
```

#RESULTIMAGE_xx

#RESULTIMAGE_xx <result screen image filename>

Mirar #RESULTIMAGE

#RESULTSOUND

#RESULTSOUND <result screen sound filename>

Sonido que se usara en la pantalla de resultados.

#RESULTSOUND_xx

#RESULTSOUND_xx <result screen sound filename>

Sonido que se usara dependiendo del rango obtenido, para el uso mirar la sección RESULTIMAGE.

#VOLUMEzz

#VOLUMEzz <volume percentage>

Ajusta el volumen en el banco de sonido zz.

#PANzz

#PANzz <panning position parameter>

Ajusta el posicionamiento del stereo en el banco de sonido zz, el uso va de -100 a 100.

#SIZEzz

#SIZEzz <chip display size percentage>

Sirve para definir el tamaño de las notas en el canal zz. Puede ser útil para mostrar visualmente que las notas aunque esten en el mismo lane, tienen volumen distinto o realmente se tendria que tocar con varios instrumentos.

#BMPzz

#BMPzz <image filename>

Define la imagen que se empleara para el chip del canal zz.

5.5. Comandos en cabecera que no se implementaran**#GLEVEL**

#GLEVEL <level value>

Dificultad de la canción en guitarra.

#BLEVEL

#GLEVEL <level value>

Dificultad de la canción en bajo.

#PREMOVIE

#PREMOVIE <preview movie filename>

Video que se pondría en el menú de selección. No es el objetivo del proyecto manejar ficheros de video.

#HIDDENLEVEL ON

#HIDDENLEVEL ON

Pone la dificultad como ??.

#BACKGROUND_GD

#BACKGROUND_GD <wallpaper filename>

Wallpaper que se usara en el modo guitarra.

#RESULTMOVIE

#RESULTMOVIE <result movie filename>

La película que se vera cuando se acaba la canción.

#RESULTMOVIE_xx

#RESULTMOVIE_xx <result movie filename>

La película que se vera dependiendo del rango.

#MIDINOTE

#MIDINOTE <ON|OFF>

Se puede usar para que salga en el canal 10 de MIDI-OUT. Lo que se esta tocando. No se implementara ya que las dos principales baterías-MIDI que se han tomado por defecto en este proyecto ya tienen formas de tener MIDI-OUT.

5.6. Objetos

Descripción de los objetos

#nnncc object list

Se define la notación de la siguiente manera:

- nnn Compas número nnn en decimales (000-999)
- cc El canal en hexadecimal (01-FF)
- object list - lista de objetos

Formato de lista de objetos

Se define la lista de objetos como donde y que Chips están una barra y un canal distribuidos uniformemente en la barra. Cada objeto esta definido como una expresión de 2 dígitos en 36-decimales (números y letras) por lo que cada objeto va entre (00-ZZ).

El objeto 00 esta definido como el silencio, y los demás objetos tendrán su banco de sonido asociado mediante el comando en cabecera de #WAVzz

Para entender esto lo mostraremos con un ejemplo, definiremos la siguiente partitura:

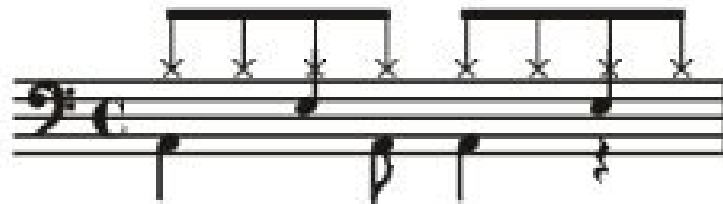


Figura 5.1: Explicación lista objetos

```
#WAV01 hihat.wav
#WAV02 snare.wav
#WAV03 bass.wav

#00111 0101010101010101
#00112 00020002
#00113 0300000303000000
```

Vamos a separar explicar cada línea.

```
"#00111 0101010101010101" -> "#00111 01 01 01 01 01 01 01 01 01"
```

Significa que en el primer compás (001), en el canal (11) que es el del hihat ponemos ocho objetos distribuidos uniformemente de tipo (01) que también se ha definido como el sonido hihat.wav.

De la misma manera:

```
"#00112 00020002" -> "#00112 00 02 00 02"
```

Significa que en el primer compás (001), en el canal (12) definido como el snare ponemos cuatro objetos distribuidos uniformemente (00 02 00 02).

```
"#00113 0300000303000000" -> "#00113 03 00 00 03 03 00 00 00"
```

En el primer compás (001) en el canal (13) bombo, ponemos la siguiente lista de objetos (03 00 00 03 03 00 00 00).

Se puede insertar los caracteres ____ que no afectan a la semántica pero pueden ayudar a la lectura humana de la partitura.

```
#00111 01010101_01010101    / or 01010101_01010101
#00112 0002____0002         / or 00__02____00__02__
#00113 03000003_03000000    / or 03000003_03000000
```

Si existe varias lista sobre el mismo tiempo y el mismo canal se mezclaran.

Ejemplo:

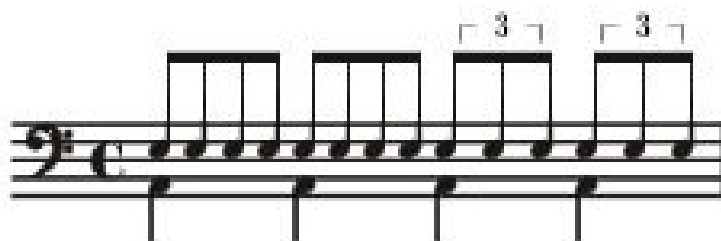


Figura 5.2: Explicación varias lista objetos

#00112 02020202_02020202_00000000_00000000	-> primera parte (dieciséis notas)
#00112 000000__000000__020202__020202	-> siguiente parte (tripletas)
#00113 03____03____03____03	

No se permitirá en este proyecto la mezcla en el compás, lane y canal ya que no esta definido que hacer si se mezclan varias notas, y no es una opción muy usada en los .dtx.

5.7. Canales definidos por el DTXMania

Se pasa a definir los canales del DTXMania, para ello se usara la misma manera de separarlos en tres tipos: canales a implementar, canales para un futuro y canales que se escapan al objetivo del proyecto.

5.8. Canales a implementar

5.8.1. 01 - BGM Back chorus

Canal empleado para los coros este canal es polifónico, se podrían emplear los canales de 61 al 92 pero esas son monofónicas.

5.8.2. 03 - BPM

Se cambiara el número de beats por minuto, esta especificado en hexadecimales (01-FF) Pensar que este valor se añade a la BASEBPM

#BPM 220
#BASEBPM 200
 #03103 0042

Al principio, BPM=220. A partir del compás 31, 03 mitad de la barra se cambiar los BPM a 266 (42 hexadecimal = 66 + 200)

Esto es lo indicado en la documentación se ha probado y lo que realmente se tiene que hacer es, si esta en la primera nota del compás cambiar los BPM del compás actual y si no cambiar los BPM para el siguiente compás.

5.8.3. 08 - Extended tempo(BPM)

En el canal 03 se ha definido el tempo como un objeto directo, que se tenía que sumar a un valor base, aquí se define el tempo con los valores zz.

Se suele usar el canal 03 para BPM enteros y para BPM reales se usa este canal.

```
#BPM 123.45
#BPM01 234.56
#00303 00007000
#00603 00010000
```

BPM=123.45 Desde el principio de la canción hasta el tercer compás y segundo beat.

BPM=112.00 desde el tercer tiempo, tercer beat.

(112 decimals = 70 hexadecimal)

BPM=234.56 desde el sexto tiempo segundo beat hasta el final de la canción.

En caso de BASEBPM=0

Esto es lo indicado en la documentación, se ha probado y lo que realmente se tiene que hacer es, si esta en la primera nota del compás cambiar los BPM del compás actual y si no cambiar los BPM para el siguiente compás.

5.8.4. 11-1A - Notas de la batería

Canales visuales donde se pondrán los Chips que veremos, los canales y los Lanes están relacionados de la siguiente manera:

```
11 = HiHatClose
12 = Snare
```

```

13 = BassDrum
14 = HighTom
15 = LowTom
16 = Cymbal
17 = FloorTom
18 = HiHatOpen
19 = RideCymbal
1A = LeftCymbal (DTXMania Release 064b061229 or later)
1F = Fill sound drums

```

No se implementara el canal 1F e indicar también que se tiene que implementar los canales 1B y 1C para tener compatibilidad con las canciones del DTXManiaHD y del gitadora, esto no esta en ninguna documentación del DTXManiaHD o del gitadora se ha tenido que obtener vía ingeniería inversa..

El canal 1B es para el pie izquierdo para el doble bombo y el 1C también para el pie izquierdo para el sonido del hihat.

5.8.5. 61-62- Sonidos en autoplay

Canales que están definidos en autoplay, se usan para poner sonidos adicionales.

5.9. Canales a implementar en un futuro

5.9.1. 02 - BAR length

Este canal controla la longitud de la barra. El valor puede ser decimal o flotante, pensar que se cambia la longitud de toda la barra.

- Valor de 1 es el de defecto es equivalente a un ritmo 4/4.
- 0.75 significaría un ritmo 3/4
- 1.0625 significaría un ritmo 17/16.

```
#01902 0.75
```

```
#01911 010101
#01912 010101010101010101
```

Este valor se mantiene, hasta que haya otro cambio o final de la canción.

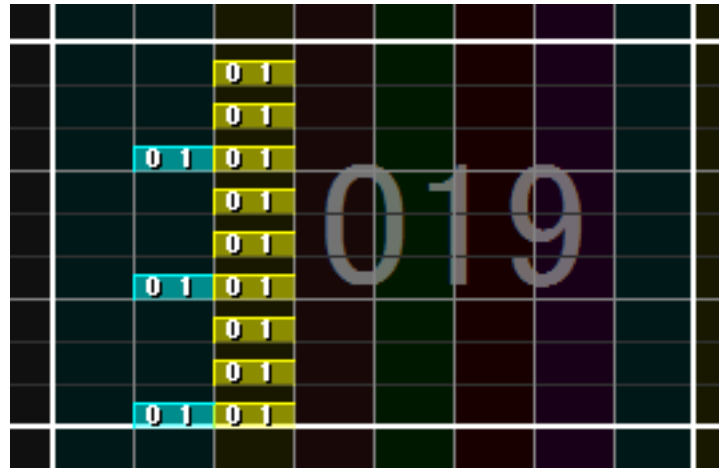


Figura 5.3: Explicación barra reducida

5.9.2. 04 - BGA

Canal donde se define el BackGround Animation.

5.9.3. 07 - BGA/ 2 capa

Muy parecido al canal 04 pero con dos capas.

5.9.4. 31-3A - Drums invible object

Este canal se emplea para que cuando se falle el chip de un canal de batería tener otro sonido definido, tienen una relación directa con los canales de la batería.

```
31 = HiHatClose
32 = Snare
```

33 = BassDrum
34 = HighTom
35 = LowTom
36 = Cymbal
37 = FloorTom
38 = HiHatOpen
39 = RideCymbal
3A = LeftCymbal (DTXMania Release 059 or later)

5.9.5. 50 - BAR line

Aunque los tiempos son manejados internamente por el DTXMania se puede hacer que se vean los BAR line en la canción explícitamente.

5.9.6. 51 - Beat Line

Aunque los tiempos son manejados internamente por el DTXMania se puede hacer que se vean los beats en la canción explícitamente.

5.9.7. 53 - Fill in

Define una zona 01 al inicio 02 al final, para escuchar el sonido de ánimos cuando se comienza un combo o se acaba.

5.9.8. 55-59,60 - BGA / capas

Lo mismo que el canal 04, pero más capas para poder definir varios fondos.

5.9.9. 63-69, 70-79,80-89,90-92 - Sonidos en autoplay

Canales que están definidos en autoplay, se usan para poner sonidos adicionales.

5.9.10. B1-B9, BC No-chip default sound

Parecido a los canales 31-3A Aquí se define que suena si no hay chip, pero este canal se puede ir definiendo hasta el final de la canción. por lo que se puede ir modificando estos sonidos.

```
B1 = HiHatClose  
B2 = Snare  
B3 = BassDrum  
B4 = HighTom  
B5 = LowTom  
B6 = Cymbal  
B7 = FloorTom  
B8 = HiHatOpen  
B9 = RideCymbal  
BC = LeftCymbal (DTXMania Release 059 or later. Note: it's NOT BA!)
```

5.9.11. C1 [BEAT-line shifting]

La línea de BEAT comenzara en ese BAR donde se haya definido en este canal C1 con el valor 02, no afecta al siguiente BEAT.

5.9.12. C2 [hide BEAT/BAR lines]

Se puede ocultar los chips para ello hay que poner el valor 02 en este canal. Para volverlos a mostrar hay que poner el valor 01.

5.9.13. C4, C7, D5-D9, E0 [Swapping BGA bitmap]

Canales para ir cambiando la imagen de fondo.

5.10. Canales que no se implementaran

5.10.1. 00

Tempo esta obsoleto.

5.10.2. 05 - Extend object

No implementado en el DTXMania.

5.10.3. 06 - Cambiar la animación de MISS

No implementado en el DTXMania.

5.10.4. 09-10 - Reservados por BMS

Reservado para otros juegos musicales.

5.10.5. 20-29 y 2F- Notas de la guitarra

Estos canales son para la guitarra

5.10.6. 30 - flow-speed drums

No estaba soportado por el DTXMania por lo que nosotros tampoco lo implementaremos.

5.10.7. 40 - Reservado por BMS

Reservado para otros juegos musicales.

5.10.8. 41-46 Invisible object for the 2nd player

No implementado en DTXMania.

5.10.9. 47-49 - Reservado por BMS

Reservado para otros juegos musicales.

5.10.10. 52 - MIDI drum chorus

Para enviar la nota a través de canal MIDI-10.

5.10.11. 54 - Playback movie

Para ver una película especificada en #AVIzz.

5.10.12. 93-99 - Reservados

Reservados.

5.10.13. A0-A7 - Bass notes

Canales para el bajo, no se tendrá en cuenta para este proyecto

5.10.14. A8 - Wailing (Bass)

Canal para el bajo, no se tendrá en cuenta para este proyecto

5.10.15. AF - Wailing sound (Bass)

Canal para el bajo, no se tendrá en cuenta para este proyecto

5.10.16. BA - No chip default sound (guitar)

Canal para la guitarra, no se tendrá en cuenta para este proyecto

5.10.17. BB - No chip default sound (bass)

Canal para el bajo, no se tendrá en cuenta para este proyecto.

5.11. Comentarios sobre el proyecto

Ahora que ya se tiene el concepto de que es un .dtx y debido a todos los posibles canales se dejara en la salida del terminal todos los objetos que no se han implementado, tanto objetos o canales de esta manera si un .dtx no suena correctamente se podrá observar que características de los .dtx se están intentando emplear y no están implementadas.

También se tendrá que mirar el formato .dtx usado en las canciones del DTXmaniahd y gitadora, ya que se ha ido ampliando el formato, y no se ha ido ampliando la documentación en ingles, se ha implementado dos canales que no estaban en la documentación original.

Capítulo 6

Especificaciones generales de diseño

6.1. Consideraciones generales a python3

Ahora ya se tiene una idea más clara de todo lo que se tendrá que implementar en el proyecto, por lo que se pueden evaluar las posibles librerías de python a usar.

Primero se detallaran unas consideraciones generales a todo el código python que se escribirá para el proyecto.

El texto empleado en las pantallas estará en ingles, para un mayor acceso a la comunidad de baterías a nivel mundial.

Se ha escogido python y la versión python3 más concretamente para este proyecto, ya que es un lenguaje interpretado y gracias a ello, sin tener que recompilar se tendrá el proyecto tanto en las plataformas x86 como en las plataformas ARM (se tendrá para otras plataformas pero no estará probado).

Ya que python es un lenguaje que tiene muy buen control sobres listas y diccionarios se integrará este control en el proyecto, principalmente para obtener tres objetivos al mismo tiempo:

- Al usar los iterators en las listas para recorrer estas el código es más rápido.
- Al usar las keys de los diccionarios para buscar la información de estos, el tiempo es optimo.
- Lectura y mantenimiento del código.

Para mantener este tercer punto de manera integrada en el proyecto, las variables de tipo lista

siempre comenzaran sus nombres como:

```
list__
```

Y las variables de tipo diccionario empezaran por:

```
dict__
```

Se emplearan listas cuando el orden importe, y diccionarios cuando no. Para los diccionarios la llave de estos normalmente sera el nombre de la lane asociada de esta manera estará optimizado para su uso, ya que la gran mayoría de búsquedas que se harán en el código se buscara por lane.

Se escogerá la versión de python 3.4 debido a que es la soportada en las principales distribuciones en la raspberry, por lo que no se tendrá que usar ninguna instrucción para versiones superiores.

6.1.1. Posibles librerías gráficas y de sonido: pygame / tkinter / Qt5

Este proyecto tiene una gran carga tanto en sonido como en gráficos, no por complejidad de estos si no más bien debido al tratamiento en tiempo real de estos.

Recordar que en música se indica que un instrumento musical se puede tocar si tiene menos de diez milisegundos de retardo entre la acción y que suene.

En python3 hay tres grandes librerías para tratar los gráficos de una manera rápida y cómoda.

Pygame, librerías especializadas en creación de videojuegos 2d.

PyQT5, wrapper a las librerías Qt5, con toda su documentación y su control multiplataforma.

TkInter, la librería “casi” standard de python.

Tkinter tiene el problemas que para hacer las animaciones se debería programar todo el control de tiempo sobre ellas, al mismo tiempo que se esta teniendo el control del sonido, por lo que se deshecha por demasiado simple.

PyGame tiene la ventaja que al estar pensada para videojuegos tiene un buen control sobre los tiempos, pero tiene dos grandes inconvenientes para este proyecto, el primero y más sencillo es que debido a su bajo nivel, el programador tiene que empezar casi de cero para crear toda una interfaz gráfica, el segundo punto es que no tiene una fácil escalabilidad si la plataforma soporta OpenGL.

PyQT5, tiene una muy buena documentación y escala bien con el OpenGL, por lo que se puede programar de tal manera que si las librerías Qt5 detectan que la plataforma soporta OpenGL

automáticamente acelerar toda la interfaz gráfica.

Una gran ventaja de las librerías Qt5 es que en su integración con Python, las llamadas para animar gráficos o reproducir sonidos no bloquean el interprete de python, se puede entender que son llamadas en paralelo, aunque que por el GIL (Global Interpreter Lock) de python esto no es del todo cierto.

Aunque se podría usar unas librerías gráficas para los menús y otras librerías gráficas para la parte mientras se esta reproduciendo la canción en sí, es mejor evitarlo para no tener problemas de que librerías recogen los eventos de teclado, ratón, foco de la pantalla.

Para el proyecto se ha decidido usar las librerías Qt5.

6.1.2. Configuraciones en disco duro: configparser

Como en este proyecto no hay necesidad de usar unas configuraciones muy complicadas se ha optado por usar la librería de python configparser, para cargarlas y guardarlas en el disco duro. Ya que són las librerías aconsejadas por la comunidad de python para estos casos.

6.1.3. MIDI: pygame / Cython y rtmidi2

Observando las librerías disponibles para tratar el MIDI en python3 se observo que existen solamente dos grandes librerías en python, pygame y rtmidi

El problema con pygame es el comentado anteriormente de usar unas librerías para los menús y otras para la parte de la reproducción de la canción en sí.

Las librerías rtmidi2 es una librería basada en Cython (que es la manera que tiene python de llamar a posibles librerías en C) el detalle es que esta librería es muy poco usada por lo que habitualmente se tendrá que descargar, tanto la librería en C, como compilar para el python instalado en el sistema las librerías Cython y rtmidi2.

La gran ventaja de usar este estilo de librerías es que al estar a tan bajo nivel son muy rápidas y se consume poca memoria ya que casi no hay aumento de memoria comparado con las librerías en C.

Viendo las posibilidades se ha optado por rtmidi2 que mediante Cython, se podrán usar los comandos de la librería rtmidi de C dentro de python.

6.2. Gráficos en qt5

Una de las grandes ventajas de emplear Qt5 como la base para los gráficos es que en todos los menús, se tiene separado la parte gráfica de la parte funcional, para ello se tienen los archivos .ui, que mediante el creador que da Qt, se puede crear de manera gráfica todos los elementos estilo ventanas, botones, imágenes, y luego mediante un sistema de eventos y señales les podremos dotar una cierta funcionalidad.

Los ficheros .ui estarán en una carpeta llamada ui, dentro del código fuente.

Cada vez que se quiera hacer un cambio gráfico en el proyecto se tendrá que generar los .py asociados a estos .ui mediante pyuic5, de esta manera en entornos de poco recursos se optimizara ya que estos python ya estarán generados, como buena practica de programación se aconseja no tocar estos ficheros generados por Qt, por lo que se tendrá que tener clases que se deriven de estos python generados.

6.3. Animaciones en qt5

Qt5 tiene un motor para hacer sus animaciones, dentro de los widgets gráficos, estan animaciones tanto pueden ser animaciones secuenciales como paralelas entre ellas.

Recordar que el lenguaje python es interpretado y tiene un garbaje collector por lo que tendremos que tener especial cuidado en no dejar de apuntar a los objetos a los que todavía tenemos que hacer referencia, aunque sea de las librerías de Qt, ya que si el garbage collector de python elimina esos objetos se podrán observar fallos gráficos.

El mismo Qt5 indica que sus timers y animaciones son aproximadas, en el aspecto que depende de la CPU su buen funcionamiento, siempre entendiendo que el refresco gráfico tiene que tener la CPU libre, para ello el mismo QT indica que lo mejor es dejar que sea su maquina de estados la encargada de actualizar todos los elementos gráficos, si se han de animar estos elementos gráficos Qt da la manera de ir agrupandolos en animaciones tanto paralelas como secuenciales, y la maquina de estados de Qt, animara de una manera o de otra dependiendo de si están en pantalla estos elementos o no.

Como python3 tiene el GIL, Global Interpreter Lock, se entiende que aunque hay llamadas a otras librerías que si que podrían estar ejecutándose en paralelo, no es del todo así, si no que para evitar fallos de concurrencia solo hay 1 thread vivo al mismo tiempo y solo se emplea una cpu. Si queremos tener un python realmente paralelo se debería cambiar de interprete y no usar el interprete por defecto.

En referencia a los timers de Qt5 en la documentacion de Qt indican que estos tiempos no son exactos y que su exactitud depende de la plataforma y como se hayan compilado las librerias Qt, para que se pinte la pantalla se tiene que parar el interprete de python y que este deje la CPU cierto tiempo libre.

Esto puede derivar que en algunas canciones con muchas notas y mucha velocidad, dependiendo de la cpu se pueden observar pequeñas diferencias visuales entre notas. A nivel de sonido y de comprobación de sincronización esto no afecta ya que esta lógica se hará mediante python por lo que al tener más prioridad el tiempo sera correcto.

Ahora entendiendo esto se puede plantear la animación de una canción de cuatro maneras distintas:

6.3.1. Ningún timer, una gran animación

Una posible implementación, de una canción y dada una velocidad de sincronización en concreto, es pensar en analizar la canción y crear todas las notas visuales en su lane correspondiente y en su tiempo asignado creando una animación para cada nota y que cada nota llegue al punto cero en el momento que debería ser tocada, evidentemente esto tiene el gran inconveniente de que tenemos que tener todos estos elementos gráficos en memoria aunque no estén en pantalla.

Ya que no tiene que tener la canción durante toda su duración el mismo tiempo, esto junto a que se entiende como tiempo de sincronización el tiempo que tarda una nota en recorrer la pantalla visible, complica el colocar las notas en su tiempo correspondiente, ya que no todos los compases tienen la misma longitud (Dependiendo que en la canción haya cambios de tempo).

Esto en la raspberry al tener una cpu lenta, se observo que no era viable.

6.3.2. Todas las notas con su timer

Otra posible implementación es hacer que cada nota tuviera un timer asociado de cuando debería aparecer en pantalla, de esta manera una vez creado todos los timers, el reproductor solo tendría que crear una animación para esa nota, cuando el timer se disparase.

El problema de esta implementación es que los timers no son lo suficientemente precisos, ya que aunque se puede indicar una precisión menor a milisegundos cuando se han de disparar se nota que es una aproximación, esto sobretodo se observa en la raspberry ya que en notas dobles, que deberían sonar al mismo tiempo pero visualmente se observaban como notas separadas.

6.3.3. Un timer para el siguiente compás

Como se han comprobado con los dos casos anteriores, tener mucha información gráfica que no estamos viendo o tener muchos timers nos puede llevar a tener retardos tanto gráficos como a nivel sonoro.

Para ello se va a reducir en lo posible estos dos elementos.

Entendiendo las animaciones como bloques se puede juntar como una animación para cada compás, creando un timer para el siguiente compás, se puede entender la reproducción como animar el compás, y esperar a este timer del siguiente compás.

El problema de esta implementación es que aunque solo tengamos un timer este va acumulando retrasos en tiempo de ejecución, y eran distinguibles en una canción un poco larga ya en un ordenador en la plataforma x86 por lo que no merece la pena comentar los retrasos que se producían al probar el código en una raspberry.

6.3.4. Un timer único para el control del tiempo

Viendo que los timers proporcionados por Qt, no tienen los requerimientos de tiempo real, que si que se tienen en este proyecto, se van a implementar las animaciones y el control del tiempo usando un solo timer que sera el control de tiempo, de esta manera si que se puede saber cuando tiempo llevamos reproduciendo la canción.

Por tanto se pasa a tener un programa que en muchas partes de su código dependerá del tiempo actual que llevamos reproduciendo.

Se codificara una lista de compases por analizar, un compás analizandose en el momento actual, y una lista de chips separadas en lanes que se están animando o se ha de reproducir sin lane visual.

Por tanto se crearan los chips visuales lo más cercano posible al momento que se han de comenzar a ver.

Comentar que siempre se tratara con dos tiempos, el tiempo según el .dtx original y el tiempo donde realmente tiene sonar la nota, que sera el tiempo original más el tiempo de sincronización. Una vez realizado un análisis del .dtx, se obtiene todas las notas en sus compases y se calcula el tiempo en el que debería sonar cada nota, y los tiempos de inicio de estos compases. Por tanto el bucle principal a la hora de reproducir, es comprobar el tiempo que llevamos de canción reproducida, si hemos alcanzado el tiempo del primer compás de la lista de compases analizar, quitar este compás y pasar a estar analizando este. En el compás que se esta analizando tiene notas con los tiempos originales en el .dtx, cuando se llega al tiempo de una nota, se tiene que pasar esta nota a su lane

correspondiente aumentado el tiempo correspondiente con el tiempo de sincronización.

Gracias a esta implementación se tiene solamente como memoria gráfica lo que tiene que estar en pantalla o que en breve tiene que desaparecer, recordar que sera el garbage collector el encargado de liberar esta memoria, lo que si que hay que indicar es que tenemos la cpu muy activa para tener un control de sonido muy preciso y cuando se ha acabo de comprobar las tareas de sonido y de control del reproductor se puede hacer la pausa para que el qt, pinte la pantalla.

6.4. Sonidos en qt5

Qt5, ya da la facilidad de reproducir sonidos, pero se tendrá que tener en cuenta si son .ogg o .wav ya que Qt5, no los trata igual, ya que uno tiene que descomprimir el archivo y reproducirlo mientras que en el otro solo tiene que reproducirlo.

Para un mejor funcionamiento del programa todos los sonidos los pondremos en un banco de memoria, de esta manera una vez ya comenzada la reproducción del .dtx solo tener que reproducir el sonido sin tener que hacer la carga a memoria.

Por definición de los sonidos en los dtx, los lanes visuales tendrán sonidos asociados estilo .wav, ya que los sonidos de la batería tienen que tener la menor latencia posible. Lo que no se puede asegurar de los sonidos asociados a los lanes no visuales, ya que se suelen poner los .ogg en canciones donde hay más música aparte de la batería, pero en las canciones de practica se suele poner los sonidos de un metrónomo con el formato .wav.

Capítulo 7

pyDTX

7.1. Ideas generales

Ahora que se esta definido el lenguaje a usar, las librerías que se emplearan y el formato .dtx se detallaran las ideas generales, para efectuar el proyecto.

7.2. Pantalla principal

Se trabaja con un Main Aplication dentro de Qt5, se pondrá un MainWindow ya que usaremos entorno gráfico, en este MainWindow se creara un stackedwidget, dentro del cual se tendra unos widgets independientes, de esta manera se aumenta el tiempo inicial pero luego como ya están todos los menús creados ahorramos en tiempo de computación. Como todos los widgets son bastante independientes seria muy fácil coger este código y modificarlo para otros proyectos de estilo musical, o modificar los gráficos manteniendo la estructura y lógica de python.

Dentro de este stackwidget se crearan los siguientes widgets:

- Widget Inicial
- Widget para configurar bateria-MIDI
- Widget para configurar pyDTX
- Widget Pantalla Navegación
- Widget Reproducción

7.3. Pantalla Inicial

Para la pantalla inicial se empleara un widget muy básico donde se pondrán cuatro botones centrados, y estos botones activaran el widget correspondiente de esta manera podemos cambiar de widgets.

Qt5 se encargara automáticamente de que solo el widget actual reciba los eventos de teclado y ratón.

El widget de la pantalla inicial tendrá una imagen de fondo dependiendo de la configuración del pydtx, aquí decir que se hará mediante .css ya que es lo más independiente del sistema y es lo que aconseja Qt5 actualmente.

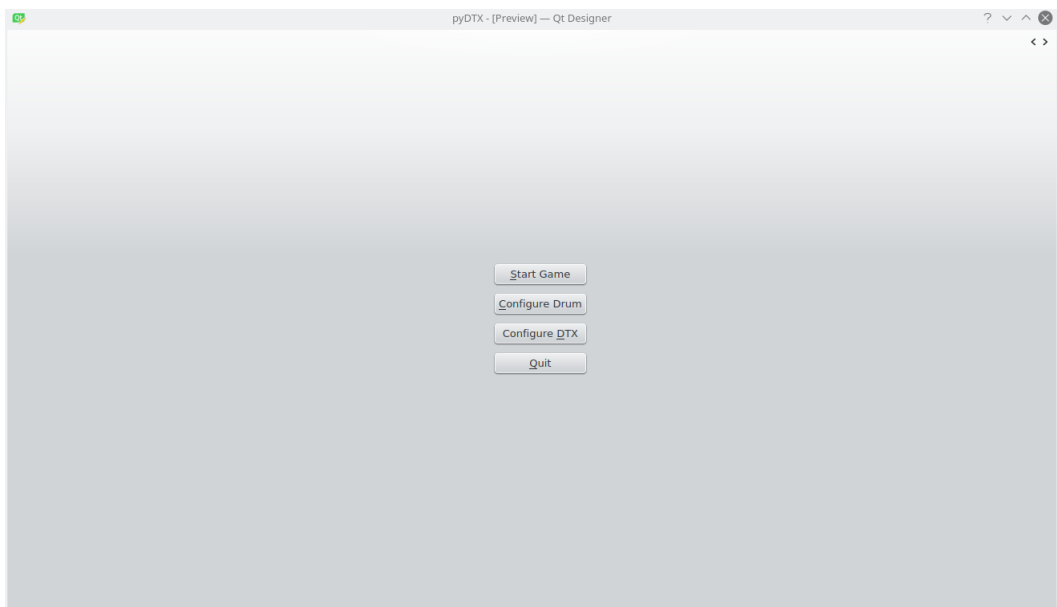


Figura 7.1: Pantalla Inicial

7.4. Pantalla configuración batería-MIDI

El widget que se usara para configurar la batería-MIDI tiene que poder dejar al usuario definir las posibles teclas, para cada lane visual de este pyDTX.

Como pantalla de configuración se tiene que leer el fichero de configuración, poner los elementos

gráficos en concordancia y un posible de botón para guardar para que todos estos valores gráficos transformarlos en un fichero de configuración.

Además se podrá definir los lanes que configuraremos en Auto para que el mismo pyDTX, haga sonar todos los chips asociados a este lane pero continúen siendo visuales.

En esta pantalla ya hay dos cosas relacionadas con la batería-MIDI, la primera un botón para poder ver cuantos instrumentos-MIDI esta viendo el sistema operativo y de esta manera poder configurar el que queremos usar para tocar los .dtx.

Para que sea más cómodo para el usuario se pondrá un botón, para poder tocar la batería-MIDI y que en la pantalla se pueda ver que nota asociada esta emitiendo la bateria-MIDI. De esta manera el usuario podrá configurar los lanes visuales con los eventos que emite su batería-MIDI.

	Note MIDI	or Note MIDI	Key
Left Cymbal	0	0	
Hi Hat Close	0	0	
Hi Hat Open	0	0	
Left Bass Drum	0	0	
Hi Hat Foot	0	0	
Snare	0	0	
High Tom	0	0	
Right Bass Drum	0	0	
Low Tom	0	0	
Floor Tom	0	0	
Cymbal	0	0	
Ryde Cymbal	0	0	

Figura 7.2: Widget Configurar Drum

7.4.1. MIDI

Antes de continuar se tiene que hacer una pequeña introducción a la comunicación MIDI enfocada a instrumentos-MIDI puros.

En la comunicación MIDI de un instrumento, hay mensajes diferenciados en 3 tipos:

- Nota-on
- Variación de nota
- Nota-off

y en cada mensaje hay la nota en sí, y su volumen.

Lo habitual es una Nota-on, y luego al cabo de X tiempo dependiendo del instrumento un Nota-off, dependiendo del instrumento se puede meter una o varias variaciones de la nota antes de emitir el Nota-off.

Estos mensajes según el instrumento puede ser múltiples, ya que ese instrumento esta pensado para estar sonando varias notas al mismo tiempo, esto se suele determinar por el instrumento en sí o por limitaciones del sintetizador MIDI que normalmente tienen un limite de notas al mismo tiempo.

En el proyecto por simplicidad y por como suele funcionar una batería, solo miraremos los mensajes tipo Nota-on.

Cada Intrumento-MIDI no emite sonidos, emite estos mensajes y es el sintetizador MIDI que teniendo sus bancos de sonidos configurados, puertos y notas es el encargado de reproducir los sonidos.

Pensar que este pyDTX, es la mezcla de un sintetizador enfocado a estos instrumentos-MIDI y a estos ficheros .dtx como partitura.

7.4.2. Configuración MIDI en pyDTX

Se tiene que exponer uno de los primeros problemas con el MIDI y su tratamiento.

Gracias a usar las librerías Qt, estas mismas librerías se encargan de que los eventos del teclado y ratón solo disparen los objetos gráficos visibles, pero si queremos implementar eventos MIDI, el

programa tiene que saber que widget esta activo o solo implementar la escucha de los mensajes MIDI en parte de reproductor.

La manera que se ha resuelto es la siguiente, cada vez que estamos en un widget que tiene que escuchar los mensajes MIDI, se tiene una función que se llama al entrar en el widget (recordar que los widgets están creados todos al inicio para mejorar la fluidez del programa). En este widget de configuración de la batería-MIDI se ha resuelto esta problemática con un botón que es el que activa el escuchar los mensajes-MIDI y lo tenemos que desactivar siempre cuando volvemos al menú principal.

7.4.3. Baterías-MIDI y Lanes

Aunque lo mejor seria que el número de lanes visuales de los .dtx a tratar fueran los mismos que el número de Notas MIDI a configurar, ya que se podría hacer una relación uno a uno. Se tiene que tratar varios casos distintos bastantes comunes.

El número de lanes que se trata como lanes visuales son 12. Aunque visualmente solo tenemos 9 columnas visuales ya que hay lanes que comparten columna visual y los chips de estos lanes visualmente son de colores distintos para diferenciarlos.

La lista completa de los lanes visuales tratados es la siguiente:

- 1A Left Cymbal
- 11 Hi Hat Close
- 18 Hi Hat Open
- 1C Left Bass Drum
- 1B Hi Hat Foot
- 12 Snare
- 14 High Tom
- 13 Right Bass Drum
- 15 Low Tom
- 17 Floor Tom
- 16 Cymbal
- 19 Ride Cymbal

Conviene hacer un pequeño repaso de como se fue ampliando los 6 lanes visuales del inicio del formato .dtx hasta los 12 actuales (recordar que la maquina actual del gitadora solo tiene 7 pads y doble pedal).

Los primeros cambios ya fue separa el Hi-hat en posición abierto o cerrado, dependiendo del número de pads de la batería-MIDI y su capacidad, se necesitaría tocar de manera distinta o solo era una marca visual de que son notas distintas, como nunca se puede tocar el hi-hat al mismo tiempo en abierto y en cerrado se puso en la misma columna visual.

El siguiente cambio fue en los platillos, originalmente solo existía el lane 16 colocado a la derecha, se añadieron los lane 1A y 19 se definió el lane 1A como el platillo izquierdo y el 19 Como el platillo derecho. Para no perder la compatibilidad con los .dtx anteriores a este cambio, se dejo el lane 16 a la derecha de todo. Aunque la maquina de Gitadora solo tiene 1 pad simple para el platillo izquierdo y otro pad simple para el platillo derecho.

Si se reproduce un .dtx, anterior a este cambio se observaran el sonido de los platillos siempre a la derecha, pero el usuario no vera ninguna diferencia.

Como este lane no se ha desactivado, son los autores de los .dtx quienes tiene que escoger como usar este lane 16.

Hay tres principales maneras de entender este lane antiguo en canciones más complicadas, la primera y mas sencilla es ignorarlo y emplear solamente los lanes de platillo izquierdo o derecho, la segunda opción es usar este lane antiguo como un tercer platillo adicional y la tercera opción es como este lane esta a la derecha junto al platillo derecho, emplearlo para tener dos notaciones en este lane de esta manera se puede simular y mostrar con una marca visual el tocar arriba del platillo o el tocar más al borde.

En este proyecto por simplicidad se han juntado en la misma columna el lane 16 y 19, se marcan con notas de colores distintos pero no esta contemplado el caso de que haya al mismo tiempo dos notas simultaneas para estos dos lanes.

Cuando salio el gitadora se amplio la máquina árcade un doble pedal para ello se insertaron los Lane 1B y el 1C que serian los dos para el pie izquierdo, tanto para el hi-hat como para el doble bombo, como no es habitual tener una batería-MIDI con triple pedal y ya que lo más común es tocar canciones con doble bombo o usar el pie izquierdo para el Hi-hat, aunque a nivel visual comparten la misma posición y en el gitadora comparten color, se ha decidido igual que en el caso anterior hacer que las notas visualmente serán de colores distintos para las canciones donde si que hay tanto doble bombo como hi hat,y de esta manera saber visualmente que son notas distintas y si se tiene una batería-MIDI más cercana a una batería acústica con estas características poder tocar exactamente como en la batería acústica.

Ya que hay multitud de baterías-MIDI en el mercado se ha de tratar dos principales problemas:

Cuando hay menos pads que lanes o al revés y los dos casos tienen sus complicaciones, que ahora se expondrán.

Pocos Pads para muchos lanes

El primer caso se solventa dejando que se pueda configurar una nota-MIDI en múltiples lanes, esto complica saber la sincronización de cuando se activa esta nota-MIDI con los lanes asociados. Ya que esta configuración obliga a cuando se dispara la nota-MIDI comprobar en varios lanes si hay notas, para ello se tendrá que comprobar en todos los lanes asociados, y como suponemos la mejor sincronización posible solo se pondrá a hit el mejor tiempo de todos los lanes asociados, también hay que tener cuidado ya que hay canciones donde puede ser que se tengan que tocar estos dos lanes simultáneamente, y si hemos configurado estas dos lanes en la misma nota-MIDI serán dos hits.

Muchos Pads para pocos lanes

El otro caso es tener una batería-MIDI muy completa, y querer usarla para tocar los .dtx, si solo se puede configurar una nota-MIDI para cada lane, esto obliga al batería a tener que ser muy exacto ya que todos los pads con zonas múltiples, solamente se deberían tocar en una zona concreta, la solución más simple es dejar que en cada lane, se puedan configurar dos notas-MIDI al mismo tiempo.

Casos especiales por respetar los dos al mismo tiempo

Estos casos independientemente tienen su sentido pero el tener que respetar los dos al mismo tiempo se tiene una serie de ventajas y desventajas.

La desventaja clara es que nos complican el comprobar la sincronización entre tocar notas-MIDI y su relación con los chips de la partitura.

La ventaja de tener estas opciones además de las comentadas es que con el mismo programa no tendremos problemas a la hora de poner canciones del dtxmania original, dtxmaniaHD o del gitadora.

Ya que el principal inconveniente era el doble bombo, si se tiene una batería-MIDI con el doble bombo. Se puede configurar de la siguiente manera: el hihat y el bombo izquierdo se configuran en el pie izquierdo, pero además este pie izquierdo se puede poner también en el bombo derecho, con esto se ha conseguido que en la misma configuración poder llamar a canciones antiguas del dtxmania

original, con doble bombo y se puede tocar con los dos pies, aunque no se respeta izquierda o derecha ya que esta asociado al mismo lane y debe ser el que toca saber que pie debería ser tocado.

Con la misma configuración si la usamos para un dtx del dtxmaniaHD o del gitadora como estos ya diferencia pie izquierdo y derecho, la podemos usar y el pydtx indicara visualmente que pie se debería tocar, aunque con esta configuración en concreto se puede tocar el lane del bombo derecho correctamente o usar el pie izquierdo. Por lo que se observa que no hay manera de respetar todos los casos posibles y con una sola configuración obligar al mismo tiempo que la pulsaciones en los pads sean las que tendrían que ser en una batería acústica.

7.5. Pantalla configuración pyDTX

En este widget se dejara las opciones para que usuario configure los valores por defectos del pyDTX.

El usuario puede configurar el directorio donde se buscara el tema gráfico del pyDTX.

También se puede configurar el directorio inicial donde se buscaran los subdirectorios que anteriormente tienen las canciones de dtx, detallar que por comodidad a la hora de navegar si queremos ir a un directorio superior de este directorio inicial se volvera a ir al menú principal.

Aunque se dejara cambiar la velocidad de sincronización antes de reproducir cualquier canción, se tiene una velocidad por defecto para ya tenerla configurada cada vez que se arranque el pyDTX. Para tener claro que relación hay entre esta velocidad, que es el tiempo que tardara un chip en recorrer la pantalla desde el punto más alto a la parte inferior, y los tiempos de sincronización se pondrán unas indicaciones en pantalla para saber los márgenes que se aplicaran para las posibles sincronizaciones Perfect, Great, Good y Poor.

Se creara la opción para seleccionar si para las animaciones de los chips gráficos durante la reproducción de la canción usaremos OpenGL, ya que esto es una asignatura pendiente en muchas plataformas arm y si obligamos a usar OpenGL, como se crean todas los widgets al inicio para mejorar los tiempos, el programa fallaría sin ni siquiera ver la pantalla de inicio (de todas maneras se puede cambiar el fichero de configuración ya que es un fichero de texto con extensión .ini)

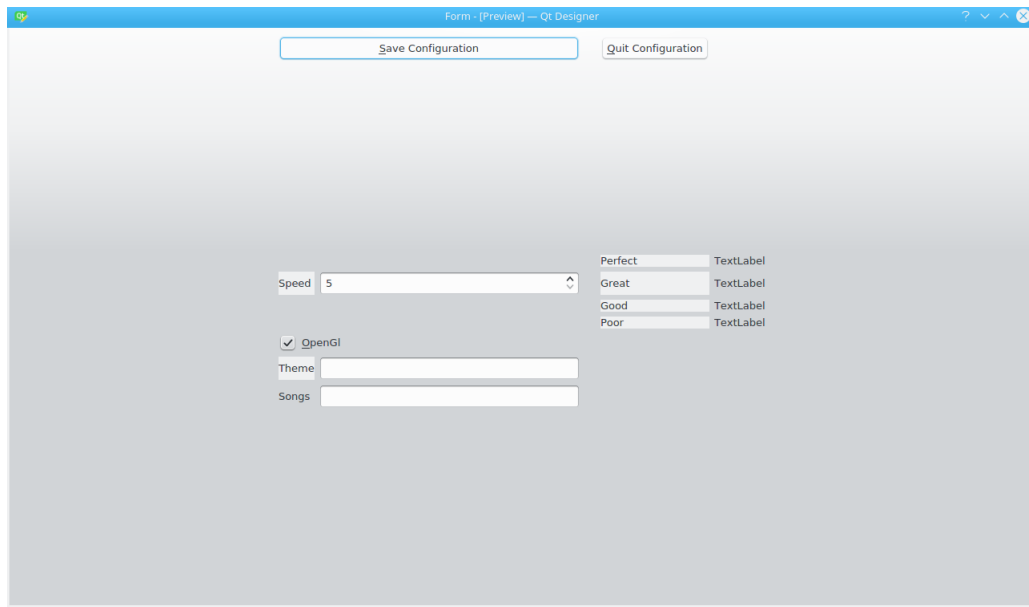


Figura 7.3: Widget Configurar DTX

7.6. Pantalla Navegación

Se tiene que definir el widget que el usuario empleara para seleccionar la canción a reproducir. Se copiará en medida de lo posible la navegación que se usaba en el dtxmania original, para ello este widget se dividirá en tres grandes partes, información de la canción actualmente seleccionada, una botonera para cambiar entre canciones o directorios, y una lista de los directorios y canciones del directorio seleccionado actualmente.

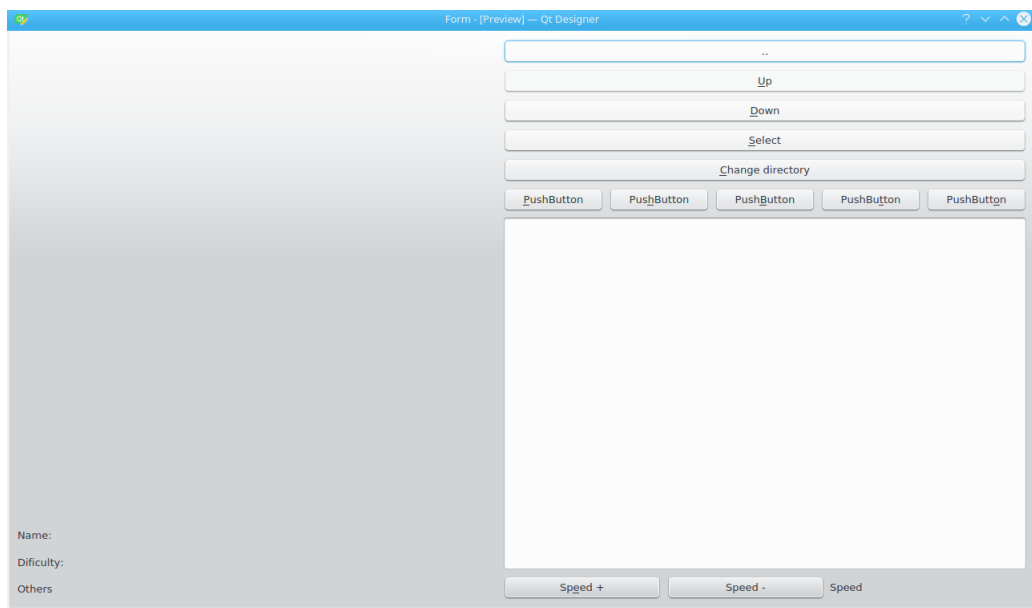


Figura 7.4: Widget Navegacion

Tenemos que definir más en detalles estas tres partes:

Información de la canción

Sera una columna a la izquierda donde se pondrá la información de la canción seleccionada, hay espacio para una imagen y varios textos (etiquetas gráficas en Qt5) para indicar autor de la canción, dificultad y uno de información extra.

Se tendrá en cuenta cada vez que se cambie el directorio o la canción seleccionada actualizar estos cuatro elementos gráficos.

Lista de las canciones y directorios actuales

A la derecha de la información gráfica y abajo de la botonera.

Se obtendrá la información de todos los subdirectorios que estén dentro del directorio actual de canciones, cuando se crea el widget sera del directorio indicado en la configuración.

Para una mayor comodidad en la navegación se entenderá que un directorio que tenga un `set.def` o un único `.dtx` contiene al menos una canción (esto se indica por el posible caso de varios `.dtx` pero ningún `set.def`).

Si solo hay un `.dtx` es tan sencillo como obtener la información del `.dtx` y ponerlo en la lista de selección, y si tiene un `set.def` obtener la información de los posibles cinco labels y de sus cinco posibles `.dtx` asociados.

Esta parte sera la encargada de tener una lista de los directorios actuales, cada ítem de esta lista sera uno de los siguientes casos:

- Si dentro del directorio hay un `set.def`, se obtendrá una lista de 5 elementos, las 5 posibles etiquetas de estos elementos y la información de los `.dtx` asociados.
- Si no tiene un `set.def` se buscara un `.dtx` dentro del directorio y el primero que se encuentre se pondra la información de esta canción (la información para los 4 elementos gráficos).
- Si es un directorio sin `set.def` o `.dtx`, guardaremos su path.

De esta manera tendremos una lista con toda la información del directorio actual ya guardada en memoria para no tener que estar pidiendo la información al sistema operativo, mientras que el usuario no se mueva de este directorio.

Botonera para cambio de directorios o seleccionar canciones

A la derecha de la información gráfica y en la parte superior se colocaran los botones para poder navegar en la estructura de directorios.

En esta parte se colocaran los botones para realizar acciones sobre la canción seleccionada, velocidad de sincronización o sobre los directorios.

Comentar que no siempre veremos todos los botones ya que es más intuitivo si los botones que no tienen sentido, se esconden y solo se muestran los botones que si pueden realizar acciones sobre el directorio o canción seleccionada.

El primero y más sencillo es el botón de ir a un nivel de directorio superior, si lo usamos y ya se esta en el directorio configurado como directorio de las canciones volveremos al menú principal, si no subiremos un nivel el path de directorio y volveremos a crear toda la lista de directorios y canciones del directorio actual.

Se crearan dos botones para subir o bajar en la lista de directorios/canciones, ya que de esta manera se deja abierto la manera de seleccionar estos directorios, se dejara seleccionar vía ratón pero también se puede seleccionar vía teclado, mediante la tabulación entre botones, recordar que de esta manera sera sencillo hacer que una batería-MIDI simule estas ordenes si solo dejáramos la vía del ratón, se debería cambiar mucho código para que funcionara mediante la batería-MIDI.

Abajo se crearan dos botones para variar la velocidad de sincronización actual y un elemento gráfico para ver la sincronización actual. De esta manera el usuario no tiene que ir hasta al menú de configuración para cambiar el valor de la sincronización.

Se creara un botón para reproducir la canción actual, este botón solo se mostrara en el caso de que estemos seleccionando un directorio que no tiene un set.def y al menos tiene un .dtx, cuando se pulsa empezaremos a reproducir la canción con la velocidad de sincronización actual.

Se creara un botón para cambio de directorio, solo se mostrara si estamos seleccionado un directorio sin set.def o .dtx, si se pulsa cambiaremos de directorio y se tendrá que rehacer toda la lista de posibles directorios/canciones

También se tiene cinco posibles botones para los directorios que tienen un set.def, como el programa original no indicaba nada en las canciones que hay por internet estos cinco posibles .dtx no tienen un orden ya que dependía del creador de los .dtx y del set.def cuantos y como se empleaban estas cinco posibilidades (hay canciones en internet que solo definen el segundo label y el tercero por ejemplo), en cada una de estas posibilidades se puede apuntar a un .dtx y poner un texto para información, además de la información que tiene el .dtx. Para ello cuando se crea la información de un directorio con un set.def se obtendrá toda la información disponible y también indicar donde no se ha podido obtener, para mostrar solamente los botones que tienen asociada información.

Se actualizara la información mostrada de la canción con el primer botón que tiene asociada información.

Se ha tenido que usar una opción del Qt5 para cuando hay focus en uno de estos botones ya que se tiene que actualizar toda la información mostrada de la canción seleccionada.

Esto se ha realizado de esta manera para integrar el uso del ratón y del teclado o posible batería-MIDI ya que si solo se tiene en cuenta el ratón siempre se pulsaba directamente los botones y no se podía ver la información de los .dtx, o se tendría que haber puesto dos botones más para poder navegar entre los posibles .dtx de un set.def, se ha preferido para simplificar el menú mediante esta reprogramación de los eventos si hay focus en uno de estos botones, poner la información del .dtx relacionado.

7.7. Reproducción de la canción

7.7.1. Inicialización básica y detalles de Qt5

El widget donde se reproducirá la canción seleccionada, sera un widget gráfico ya que sera el encargado de tener los lanes visuales y dentro de ellos los chips visuales, además del widget de puntuación.

Este widget gráfico ya estará creado, y según la configuración se usara OpenGL o no, se tiene que desactivar las opciones del qt5 de las barras de scroll del QGraphicsView, ya que por defecto si se añaden elementos gráficos en los limites de la visión, aparecerían barras de scroll para poder ver todo.

Otra cosa a evitar es el evento `resizeEvent`, ya que si se redimensiona la pantalla, se ha pensado que siempre estaremos usando la misma resolución interna, de esta manera sera el Qt5 el encargado de mantener la relación de aspecto.

No se puede inicializar nada más ya que se depende de la canción seleccionada y del tiempo de sincronización.

7.7.2. Inicialización con una canción

Cuando se llama a este widget con un .dtx concreto y una velocidad de sincronización. Es cuando se tiene que reproducir el dtx, para ello se tiene que hacer varias tareas que paso a describir.

Se tiene que leer los ficheros de configuración, para tenerlos en memoria.

Como toda la parte gráfica sera mediante Qt5, se crea una scene y se ira añadiendo los elementos gráficos a esta scene para que Qt5 sea quien los pinte.

Se tiene que configurar el teclado y el instrumento MIDI.

7.7.3. Teclado

Configurar teclado

Se crea un diccionario donde se guardara la relación de teclas con su lane correspondiente.

Recorriendo todas las lanes visuales, se comprueba si la lane esta configurada sin automático, si fuera de esta manera se inserta su tecla asociada a su lane en este diccionario de teclas.

Evento de teclado

Se configura el evento `KeyPressEvent` de Qt5, para que cuando se pulse una tecla lance este evento.

Se comprueba si es la tecla especial `Escape`, si fuera esta se tiene que parar la reproducción actual y volver a el menú de selección de canciones.

Si no fuera así, se comprueba que tecla se ha pulsado y si esta en el diccionario de teclas que se ha configurado, se tiene que comprobar el tiempo actual y comprobar la puntuación en el lane asociado, esto se explicara más concretamente cuando definamos los lanes visuales y que funciones tienen.

7.7.4. MIDI

Configurar MIDI

Se crea un diccionario donde se guarda las notas midis y la lista de sus lanes correspondientes.

Recorriendo todas las lanes visuales, si la lane esta configurada sin automático, se comprueba su dos posibles MIDI y entonces se añaden a este diccionario de MIDI, con cuidado ya que una nota MIDI puede activar múltiples lanes, por eso en este caso son listas dentro de un diccionario y no un diccionario simple.

Evento MIDI

Mediante `rtmidi2` se define una función que se dispara cada vez que vea un evento MIDI, del instrumento MIDI que esta en la configuración.

Cuando se dispara este función es comprobar el tipo de mensaje MIDI que tiene, los eventos tipo '153' `Note-On` serán los que se traten.

Si fuera de esta manera, se comprueba si la nota de este evento esta definida dentro de nuestro diccionario.

Aquí hay que tener mucho más cuidado que con el teclado ya que hay dos posibles casos: Que haya solamente un lane asociado a esta nota-midi por lo que es lo mismo que el caso con el teclado o, puede ser que haya varios lanes asociados a esta nota-midi por lo que se tendrá que hacer una doble pasada sobre los lanes asociados: una primera para obtener el tiempo menor comparando con el tiempo actual, para saber a que lanes iría esta nota, y luego ya ir a los lanes y marcarlos como hit, recordar que pueden existir notas en varios lanes en el mismo tiempo.

7.7.5. Lanes

Una vez configurados los métodos de entrada teclado y midi. Se tiene que crear los lanes tanto automáticos como visuales.

7.7.6. Ideas básicas de los lanes

Se entiende un lane como una lista de chips, los lanes se separan en dos grandes grupos lanes automáticos y lanes visuales, los cuales tendrán a su vez asociados chips auto y chips visuales, pasamos a describir los chips ya que son más sencillos y luego se describirán con más exactitud los lanes.

Chips Auto

Se tiene que crear una clase Chip Auto.

Los chips auto son los chips que tendrán los lanes automáticos puros, que son los que se emplean para o poner la música de fondo o para ir añadiendo sonidos simples en la canción, ejemplo clásico metrónomo para canciones de practica.

Un chip auto es una nota (un sonido que estará en un banco de sonidos definido en el dtx) y un tiempo donde debería ser tocada.

Se crearan dos funciones en esta clase para leer el tiempo y su nota en sí.

Chips Visuales

Se crea una clase que se extiende de QGraphicsWidget, para que Qt5 pueda manejar toda la parte gráfica y se pondrán los mismos datos y funciones que la clase Chip Auto.

Esta clase es un tipo de Chip pero visual, donde además de la información anterior si tiene que tener la información de si el chip ya ha sido pulsado, una animación, un color de chip y una posición horizontal en pantalla.

Los datos de color y posición son dependientes del lane donde estará este chip.

Las funciones que se definen además de las de un chip auto son:

paint: función que usara Qt5 para pintar este chip en concreto donde se indica que pinta un rectangulo, con un color.

animar: función donde se tienen que crear una propiedades de animación para que qt5 lo trate, básicamente es poner un chip, arriba de la pantalla con su posición horizontal de la lane donde está e indicar que tiene que acabar esta animación en la misma posición horizontal pero al final de la pantalla, indicando que tiene que tardar el tiempo de sincronización.

Esta función tiene que devolver la animación en sí, ya que sera la scene de Qt5 donde se tiene que tratar.

Lanes Auto

Se creara una clase lane Auto donde se tendrán dos datos una lista de chips, y un banco de sonidos.

Las funciones asociadas a esta clase son las siguientes:

insertar_chip: Para poder insertar chips en la lista de chips.

insertar_sonidos: Para poder insertar el banco de sonidos en la lane auto.

reproducir: Donde se buscara el primer chip de la lista y si su tiempo es igual o menor al tiempo actual, se tendrá que reproducir su nota asociada y eliminar este chip.

Lanes visuales

Extendiendo la clase de Lane Auto se definirá la clase Lane visual estos lanes, tendrán una lista de chips y una función para reproducir los sonidos por si estuvieran configurados como automático, se tendrá que añadir toda la información gráfica y funciones para comprobar la puntuación obtenida según el sincronismo.

Los datos de un lane visual son los de un lane automático y además color, posición horizontal y el tiempo de sincronización, de este último se calcula el tiempo máximo a mantener una nota en memoria, ya que una nota que ya ha pasado su tiempo donde tendría que haber sido tocada y el margen de puntuación en “poor”, esa nota es un “miss” y se puede borrar de memoria.

Además de funciones para leer datos del estilo leer_color y leer_pos.

Hay tres funciones adicionales para estos lanes visuales:

comprobar_y_poner_hit:

Función para buscar la nota más cercana al tiempo actual, y ponerla a hit, se devolverá la diferencia del tiempo actual al tiempo de la nota, para poder puntuar.

comprobar_tiempo:

Función que se usara para cuando una nota-midi esta en varios lanes visuales, se comprueba en la lista la nota más cercana al tiempo actual y se devolverá el tiempo de diferencia; de esta manera se podrá saber el lane donde esta la nota más cercana.

eliminar_nota_antigua:

Con el tiempo actual se comprueba si la primera de la lista ya ha superado su tiempo y el tiempo de margen de hasta una nota en calificación “poor”, se borra esta nota de las lista de notas si ha superado este tiempo. Y se devuelve true si se ha eliminado una nota que no haya sido pulsada, esto es para poder indicar a la puntuación que hay una nota más en miss.

Reproducir una canción

Ahora que se han definido tanto los chips como los lanes, se puede indicar tranquilamente que se tienen que crear los lanes automáticos, 01, 61 y 62, que son los lanes automáticos más comunes en los .dtx. y los se ponen en un diccionario de lanes automáticos.

Se crean los lanes visuales 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B y 1C con sus colores y posiciones, siempre mirando si en la configuración esta puesto como lane automático, si fuera de esta manera además de en el diccionario de lanes visuales, se tendría que añadir también al diccionario de lanes automáticos.

Se pone el número de compases a cero, y se tiene que obtener la información básica del .dtx, para ello se tendrá que hacer varios pasos.

En el primer paso se obtendrán los siguientes datos:

Todos los bancos de sonidos que están definidos como ordenes WAVxx, si el sonido es un .ogg o un .wav se creara un banco de sonido con un reproductor asociado al tipo del sonido; de esta manera cuando se tenga que reproducir el sonido ya estará cargado en memoria y preparado.

Toda la información de los posibles cambios de BPM en la canción y de los BPM por defecto.

Se obtendrá la información del nombre de la canción para ponerlo en el widget de la puntuación.

Se ira tratando la información de los compases poco a poco para ello, primero obtendremos sus notaciones, en que lane son estas notas y en que compás.

Como todas las notaciones deberían estar colocadas por compás y no hace falta que estén ordenadas por lane, lo que se hará mientras se recorren todas las notaciones en el archivo .dtx es lo siguiente, si es un compás mayor que el actual se pondrá en esta lista de compases en el sitio del compás un diccionario con la key 'lanes' vacía, y luego se insertara los datos en el compás, y en este diccionario ya se pone usando la lane como key, las notaciones u ordenes sin tratar todavía(tal y como están en el archivo .dtx), también recordar que en los ficheros .dtx se pueden saltar compases si no hay ninguna nota en esos compases, pero para el proyecto ya se crea en esta lista de compases, compases que internamente tendrá una key 'lanes' pero no llegaran a tener notaciones internas.

Ahora ya se tiene una lista de compases y en cada compás un diccionario con una key 'lanes', y con las notaciones separadas por lanes.

Ahora creamos la puntuación.

Puntuación

Usamos un widget por comodidad, donde tendremos los siguientes datos:

Datos visuales: Titulo de la canción Número de notas en perfect Número de notas en great
Número de notas en good Número de notas en poor Número de notas en miss Número de notas en
combo Número de notas máximas en combo

Datos a guardar en este widget: Los tiempos de margenes calculados a partir del tiempo de sincronización.

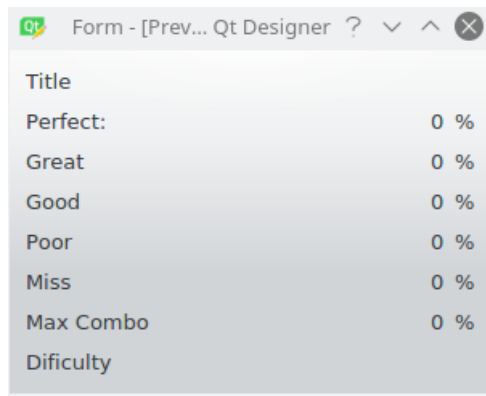


Figura 7.5: Widget Puntuación

Con las siguientes funciones:

`comprobar_puntuacion`: Con el tiempo de margen que nos pasan, se miran los tiempos de sincronización y se actualizan los datos en consecuencia.

`actualizar_puntuacion`:

Función para actualizar los elementos gráficos de la puntuación.

`aumentar_miss`: Función que se empleara cuando hay una nota que se tiene que eliminar por tiempo, ya directamente se sabe que es un miss.

Continuamos actualizando los compases

Insertamos los bancos de sonidos en todos los lanes.

Ahora se tiene que poner en todos los compases los tiempos, para ello se ira recorriendo la lista de compases y si hay una nota en los lanes 03 o 08 ya que estos modifican los BPM de la canción; lo más habitual es que se cambie al inicio del compás o justo en la última nota del compás para el compás siguiente, se ha implementado tanto el lane 03 como el 08, ya que aunque parece que se quiere dirigir el formato hacia cambios absolutos, para tener retrocompatibilidad se ha implementado los cambios de BPM con sumas o restas al BPM actual, para ello en el diccionario que es el compás actual ahora añadimos dos keys, que son la key 'bpm' asociada al bpm del compás actual y la key 'tiempo' que es el tiempo donde se debería iniciar el compás según el .dtx.

Ahora se tiene que recorrer otra vez la lista de compases y ahora se puede coger todas las

notaciones de los lanes y compases, y separarlos en notas con un tiempo, quitando las notas 00 que son silencios en los .dtx.

Se crea un evento para salir del reproductor cuando se acabe el tiempo de la canción más el doble tiempo de sincronización, para tener un margen extra.

Se inicia el control del tiempo, ya que se tiene todos los datos ya tratados para comenzar la reproducción del .dtx.

Se anima el primer compás, ahora se explicara que se considera animar un compás, se quita de la lista de compases pasándolo a ser el compas que se esta comprobando actualmente, y se pasa al bucle principal de la reproducción, usando un timer de Qt5 para cada milisegundo volver al bucle principal.

Animar un compás

Para animar un compás en la reproducción de un .dtx, se necesita un compás, y el tiempo actual.

Para animar el compás actual lo que se tiene que hacer es repasar todos sus lanes, mirar la primera nota de cada uno de esos lanes y si su tiempo es menor que el tiempo actual, quitarla del compás a animar y pasar esta nota a su lane correspondiente modificando el tiempo de esta nota (sumando el tiempo de sincronización), pensar que se estará animando este compás mientras no se llege al tiempo del siguiente compás.

Hay que separar las notas en dos grandes grupos:

Las que están en lanes puros automáticos (01,61 y 62).

Y las notas para los lanes visuales, en este caso se tendrá que crear una nota visual con sus propiedades gráficas dependiendo del lane y añadirla a la scene asociada al Qt5.

Si fuera una nota no asociada a ningún lane implementado se pondrá esta información en el terminal.

Bucle principal de la reproducción

En el bucle principal de la reproducción se tiene que hacer varias cosas:

Coger el tiempo actual, continuar animando el compás actual, ya que puede tener todavía notas a animar.

Comprobar que no se esta en el tiempo del siguiente compás. Si fuera de esta manera se actualiza el compás actual y se quita de la lista de compases.

Se da la orden de reproducción en todos los lanes auto, tanto los puramente auto como los configurados por el usuario.

Se comprueba que no se tiene que eliminar ninguna nota por tiempo, en los lanes visuales.

Y se vuelve a usar un timer de Qt5 para hacer una parada en la CPU y que Qt5 pueda actualizar todos los elementos gráficos y después se volverá a llamar a este bucle principal.

Capítulo 8

Detalles de la implementación

8.1. Transformar .dtx a .dtx libre

La idea es bajarse canciones de internet y ejecutando alguno de los dos scripts de transformar a .dtx libre estos se encargaran de varias tareas:

- Codificar los .dtx en utf-8
- Transformar los .xa a wav
- Transformar los .mp3 a .ogg
- Cambiar internamente los .dtx apuntando a estos archivos en formatos libres

Se dan dos posibles scripts ya que hay .dtx que tienen los sonidos .xa o .wav dentro de una subcarpeta para mejorar la estructura de ficheros.

Las canciones tendrían que estar dentro de la carpeta que contiene estos scripts ya que el wine necesita tener los path relativos para encontrar tanto los ficheros a transformar como el xa.exe para poder transformar de .xa a .wav.

8.2. DRUM-HAT MIDI

Se ha conseguido un código independiente que si esta en cpu al mismo tiempo, que el pyDTX, el Drum-HAT emite señales MIDI que se pueden tratar en este o otros programas que reciban

información MIDI.

8.3. pyDTX

Se ha configurado un script de python con la opción de optimización, para que se mejore en lo posible la ejecución, viendo las ideas generales, el mayor problema de este proyecto es el buen uso de los timers y de la capacidad de la cpu para reproducir simultáneamente todos sonidos en los lanes, de esto se encargara las librerías Qt5, en segundo plano queda el tema gráfico que también se encarga el Qt5.

Son las librerías Qt5 que según su código y sistema operativo donde se esta ejecutando, según los recursos disponibles obtendrán acceso a la pantalla y al sonido.

Se ha implementado el control MIDI vía Cython por lo que esta bastante optimizado.

La programación se ha realizado de tal manera que siempre el sonido tiene prioridad, sobre los gráficos.

Debido al poco uso de dobles líneas en un .dtx para un mismo compás y lane en concreto, no se ha implementado, aunque este descrito en el formato, la descripción no indica que hacer si por error se crean dos notas al mismo tiempo en el mismo lane.

8.4. Instalación

8.4.1. Fedora 25 - x86 64bits

En una distribución fedora 25, las librerías necesarias y sus instrucciones para instalarlas son las siguientes:

```
sudo dnf install alsa-lib-devel
sudo dnf install python3-Cython
sudo dnf install rtmidi-2*
sudo dnf install rtmidi-devel-
sudo dnf install redhat-rpm-config
sudo dnf install gcc-c++
sudo pip3 install rtmidi2
```

En un linux mint se tuvo que añadir las siguientes librerías para poder compilar el Cython:

```
sudo apt-get install python3-setuptools
sudo apt-get install g++
sudo apt-get install python3-dev
```

8.4.2. Raspberry3 - armv7

En la raspberry3 en la distribución Raspbian y ubuntu, están son las librerías a instalar para este proyecto:

```
sudo apt-get install librtaudio-dev
sudo apt-get install librtmidi-dev
sudo apt-get install librtmidi2
sudo pip3 install Cython
sudo pip3 install rtmidi2
sudo apt-get install libqt5multimedia5
sudo apt-get install libqt5multimedia5-plugins
sudo apt-get install python3-pyqt5
sudo apt-get install python3-pyqt5.qtmultimedia
sudo apt-get install python3-pyqt5.qtopengl
```

Indicar que la instalación de las librerías Cython dura unos 20 minutos.

En la distribución Raspbian la instrucción:

```
sudo apt-get install librtmidi2
```

No es necesaria ya que en la instrucción anterior por dependencias ya se la instala pero se ha dejado para tener las instrucciones tanto en raspbian como en ubuntu en el mismo lugar.

8.4.3. Conseguir el código fuente

Para conseguir el código fuente del proyecto se puede usar las siguientes ordenes

```
git clone https://github.com/kapoirapyDTX.git
```

Para conseguir el código fuente del script para transformar los .dtx de internet se puede usar la siguiente orden:

```
git clone https://github.com/kapoirapyDTX-libre.git
```

Si se quiere conseguir la utilidad para que el DrumHat genere señales MIDI, se puede obtener con la siguiente orden:

```
git clone https://github.com/kapoirapyMIDI-out-DrumHat.git
```

8.4.4. Problemas encontrados con .dtx de internet

Se ha de comentar que además de no tener toda la documentación en inglés, ya que el autor ha ido ampliando el formato según quería adaptarlo a las nuevas versiones del dtzmania hd o del gitadora, el autor también ha ido añadiendo posibles comandos de los que no hay ninguna documentación.

Un claro ejemplo es el comando:

```
#DTXC_CHIPPALETTE:
```

Hay .dtx en internet donde no se cumple la documentación de los .dtx con errores del estilo:

No poner el carácter “:” cuando esta definido en la documentación ejemplo : #L1LABEL en vez de #L1LABEL:

Apuntar a canales de sonido no definidos en el mismo .dtx.

Todos estos errores no son tratados por el script que pasa de .dtx a .dtx libres, por lo que según que errores pueden hacer que este reproductor tenga fallos, se ha dejado la salida del terminal activa para poder tener controlados estos fallos y modificar el .dtx en consecuencia.

8.4.5. Comentarios finales en la Raspberry

Para que funcione en raspbian se tiene que llamar al demonio pulseaudio, y si además se quiere usar el drumHat al mismo tiempo, también se tendrá que llamar al programa MIDI-out-DrumHat.

En la raspberry3 este proyecto funciona pero el sonido se entrecorta, se puede usar como lector de partituras pero no se aconseja estar pendiente del sonido que saca la raspberry.

Como se puede observar el código esta funcionando por lo que solo es cuestión de tiempo o de alguna de las siguientes opciones:

- El interprete de python pypy3 tenga soporte para las librerías qt5.
- Raspbian tenga un kernel en arm64 (La raspberry3 tiene una cpu capaz pero por no tener que mantener doble código por ahora la comunidad oficial solo dan la opción de armv7).
- Se obtenga la aceleración OpenGL completa, de esta manera se liberara la cpu y se podrá mejorar los tiempos de latencia que afectan al sonido.
- Se mejore el pulseaudio en la distribución Raspbian (Raspbian por ahora soporta ALSA de base ya que consume menos recursos).
- Si sale una raspberry más potente.

También se podría probar en otros microordenadores como los ODROID, ya que estos tienen una cpu más potente que la Raspberry, aunque también se tendría que mirar su compatibilidad con OpenGL y las librerías Qt5.

8.4.6. Comentarios finales al proyecto

Ahora que se ha tenido que implementar un reproductor de .dtx en entornos libres, se han observado carencias en el formato o puntos flojos ya que se han ido añadiendo comandos a medida que pasaba el tiempo que no estaban pensados en un origen.

Aquí se expone una lista de posibles mejoras para un futuro formato.

- Un detalle que poco a poco se han dado cuenta es evitar los formatos con DRM, aunque eso muchas canciones continúan usando los .xa o .mp3.
- Referente a esto en los lanes visuales se suele usar sonidos del estilo rápido ya que es donde se colocan los sonidos de la batería, pero estaría bien tener un lane o varios que fueran

específicamente para la música de fondo, de esta manera se simplificaría la parte de control de sonidos por si son .wav o .ogg.

- Simplificar el cambio de tempo dejando solo cambiar con valores absolutos, ya que es mucho más simple para el ordenador de programar.
- Se podría mejorar el tratamiento de los .dtx generando un fichero intermedio, ya que cada vez que queremos reproducir un .dtx, tenemos que repasar todo el .dtx para generar todos los compases, todas sus notas y asignar tiempos a estas notas, como todo esto no depende del tiempo de sincronización se podría crear un fichero que mientras no se modifique el .dtx asociado, y quitarnos tiempo de computación.
- Definir bien los lanes asociados a los platillos izquierdo y derecho, como ya se hizo con los pies, el problema de esto es que hay que rehacer canciones para tenerlas en el formato correcto.

Capítulo 9

Posibles mejoras en el futuro

Cuando se actualice la versión de python en las distribuciones de la raspberry, se podría actualizar la manera de revisar los ficheros existentes en los directorios de las canciones, fue un gran cambio en el paso a python 3.5 pero no se ha usado en este proyecto para no dejar a las grandes distribuciones de raspberry colgadas.

El siguiente gran cambio en python 3.6 seria actualizar la manera de ver las notas ya que se actualizo la manera de entender los números ya que se contempla a partir de esa versión poner el carácter guión bajo para una mejor lectura de los números: 1_000_000 ahora se acepta como número y esta característica se podría emplear para limpiar la parte de coger objetos y notas de un .dtx.

Ampliar las ordenes del .dtx y los lanes implementados para tener un mayor porcentaje de compatibilidad con los .dtx que hay en internet.

Una opción que seria muy interesante y no esta implementada es conseguir que también en el menú de navegación de las canciones poder usar el instrumento-MIDI configurado como entrada para de esta manera poder arrancar el pyDTX y si esta la batería-MIDI bien configurada, poder usar todo el pyDTX sin dejar las baquetas.

Aunque se muestra la puntuación no se guarda en ningún sitio ni hay una pantalla de final canción para comprobar tranquilamente lo bien o mal, que se ha tocado una canción, aquí según como se quiera entender el aprendizaje de la batería se debería implementar de distintas maneras ya que la básica de solo guardar la puntuación máxima, y no poder ver que parte de la canción ha sido donde se ha fallado no es del todo correcta.

Actualmente no hay ninguna animación para cuando se toca un Pad, esto es bueno en el aspecto que minimiza todo el uso de cpu para carga gráfica pero estaría bien tener una animación para

cada lane para poder comprobar que nivel de sincronización tenemos en cada nota, ya que en las notas dobles es difícil saber que nota es la que no se ha tocado bien.

No se analizan los box.def que pueden existir en las carpetas superiores de las canciones para tener o una imagen (de todo el álbum) o más información que solo el nombre de la carpeta.

Para dejar todo más elegante estaría bien poner también unos fondos de pantalla en las pantallas de configuraciones, dependientes del tema.

Un detalle para los baterías sería indicar al reproductor, que haga marcas visuales de en que compás esta la canción y marcar visualmente el inicio de cada compás.

Sería interesante implementar este proyecto usando QtQuick2 ya que Qt5 se esta enfocando en su lenguaje QML, más que en los widgets, de esta manera logran todavía más una mayor independencia del sistema operativo y en teoría una mejor optimización de los recursos empleados.

Apéndice A

Código fuente

A.1. Transformar .dtx a .dtx libre

```
find . -type f -iname *.dtx -exec sh -c 'iconv -f $(file -bi "$1" | sed -e "s/.*[\r\n]
    charset=//") -t utf-8 -o converted "$1" && mv converted "$1"' -- {} \;
find . -iname '*.xa' -execdir wine ../xa122/xa.exe -d -u '{} ' \;
find . -iname '*.dtx' -execdir sed -i -e 's/\.xa/\.wav/g' '{} ' \;
find . -iname '*.dtx' -execdir sed -i -e 's/\.XA/\.wav/g' '{} ' \;
find . -iname '*.dtx' -execdir sed -i -e 's/\\\/\\\/g' '{} ' \;
find . -type f -iname "*.mp3" -exec bash -c 'FILE="$1"; ffmpeg -i "${FILE}" -acodec
    libvorbis "${FILE%.mp3}.ogg";' _ '{}' \;
find . -iname '*.dtx' -execdir sed -i -e 's/\.mp3/\.ogg/g' '{} ' \;
```

A.2. DRUMHAT en instrumento midi

```
#!/usr/bin/env python3
```

```
import caplxxx
import time
import rtmidi2
```

```
"""
```

```
4 3 2
5 7 1
6 0
```

```
"""
```

```
#modificado de los ejemplos de pimoroni para que use el rtmidi2 y se convierta
# en un instrumento MIDI
```

```

dh = cap1xxx.Cap1188(
    i2c_addr=0x2c,
    alert_pin=25)

ledmap = [
    5,
    4,
    3,
    2,
    1,
    0,
    6,
    7
]

state = [True] * 8

#Definimos el canal MIDI de salida
midi_out = rtmidi2.MidiOut()
midi_out.open_port(0)

def handle_press(event):
    """
    funcion para cuando se pulsa un pad del drum hat

    event: evento que lo dispara
    """
    # Siempre usamos la nota relacionada con el evento +1
    # por comodidad y la velocity 100
    midi_out.send_noteon(153, event.channel + 1, 100)
    dh.set_led_state(ledmap[event.channel], True)

def handle_release(event):
    """
    funcion para cuando se despulsa un pad del drum hat

    event: evento que lo dispara
    """
    dh.set_led_state(ledmap[event.channel], False)

for x in range(8):
    dh.on(x, event='press', handler=handle_press)
    dh.on(x, event='release', handler=handle_release)

dh._write_byte(cap1xxx.R_LED_LINKING, 0b000000)

while True:
    time.sleep(1)

```

A.3. pyDTX

No se indica el código fuente generado desde los .ui

A.3.1. ChipAuto.py

```
# -*- coding: utf-8 -*-

class ChipAuto():
    """
        Clase, sera un chip, (nota y tiempo)
    """

    def __init__(self, nota, tiempo):
        """
            Creamos los valores iniciales

            nota: banco asociado a la nota
            tiempo: tiempo de la nota
        """
        self.tiempo = tiempo
        self.nota = nota

    def leer_tiempo(self):
        """
            Devolvemos el valor del tiempo de la nota

            return: tiempo
        """
        return self.tiempo

    def leer_nota(self):
        """
            Devolvemos la nota (banco de sonido) asociado al chip

            return: nota
        """
        return self.nota
```

A.3.2. Chip.py

```
# -*- coding: utf-8 -*-

from PyQt5 import QtCore
from PyQt5 import QtWidgets

class Chip(QtWidgets.QGraphicsWidget):
```

```

"""
    Clase chip gráfico, sera un chip, (nota y tiempo) y debera tener la
    información gráfica para Qt, animación y como pintarse
    también guardara si esta nota ya se ha pulsado en la bateria
"""

def __init__(self, nota, tiempo):
    """
        Creamos los valores iniciales

        nota: banco asociado a la nota
        tiempo: tiempo de la nota
    """
    super(QtWidgets.QGraphicsWidget, self).__init__()
    self.tiempo = tiempo
    self.nota = nota
    self.animacion_nota = False
    self.color = False
    self.hit = False
    self.prop = False

def paint(self, painter, option, widget):
    """
        Definimos como se ha de pintar una Chip grafico en qt
        Es un rectangulo de color(este color dependera de la lane)

        painter: Que painter usaremos lo ponemos por qt
        option: Opciones del painter esto lo tenemos poner por qt
        widget: Sobre que widget se pinta lo ponemos por qt
    """
    painter.fillRect(self.rect(), self.color)

def leer_tiempo(self):
    """
        Devolvemos el valor del tiempo de la nota

        return: tiempo
    """
    return self.tiempo

def leer_nota(self):
    """
        Devolvemos la nota (banco de sonido) asociado al chip

        return: nota
    """
    return self.nota

def leer_hit(self):
    """
        Devolvemos el valor de si ha sido golpeado este chip

        return: hit
    """

```



```

    """
    return self.hit

def set_hit(self):
    """
    Indicamos que esta nota ya acaba de tocar
    """
    self.hit = True

def set_color(self, color):
    """
    Definimos el color del chip, dependera de la lane

    color : color a asociar al chip
    """
    self.color = color

def animar(self, posicion, tiempo):
    """
    Creamos la animación y la guardamos en el mismo chip
    La nota comienza arriba de la pantalla e ira hasta abajo

    posicion: posicion horizontal del chip para esta animacion
    tiempo: tiempo que tardara el chip en bajar
    return: devolvemos la animacion para que qt la pueda pintar
    """
    geometry = QtCore.QByteArray(b'geometry')
    self.prop = QtCore.QPropertyAnimation(self, geometry)
    self.prop.setDuration(tiempo)
    self.prop.setStartValue(QtCore.QRect(posicion, 0, 80, 10))
    self.prop.setEndValue(QtCore.QRect(posicion, 720, 80, 10))
    self.animacion_nota = QtCore.QParallelAnimationGroup()
    self.animacion_nota.addAnimation(self.prop)
    self.animacion_nota.start()
    return self.animacion_nota

```

A.3.3. LaneAuto.py

```
# -*- coding: utf-8 -*-
```

```

class LaneAuto():
    """
    Clase Lane Automatico, sera un lane, una lista de chips
    Tiene asociado un reproductor de sonidos rapidos .wav
    y otro de sonidos lentos .ogg
    También tenemos el banco de sonidos de la canción
    """

    def __init__(self):
        """
        Creamos los valores iniciales

```

```

    """
    self.lista_chips = []
    self.banco_sonidos = {}

def insertar_chip(self , chip):
    """
        Insertamos un chip al final de la lista

        chip: chip a insertar
    """
    self.lista_chips.append(chip)

def insertar_sonidos(self , banco_sonidos):
    """
        Definimos el banco de sonidos

        banco_sonidos: Banco de sonidos de la canción
    """
    self.banco_sonidos = banco_sonidos

def reproducir(self , tiempo_actual):
    """
        Se buscara el primer chip y si justo estamos cuando ha de sonar
        reproduciremos su sonido y lo quitamos

        tiempo_actual: tiempo actual
    """
    #Comprobar que existe el primer chip
    if self.lista_chips:
        nota = self.lista_chips[0]
        if nota.leer_tiempo() <= tiempo_actual:
            #Tenemos que reproducir su sonido asociado
            self.banco_sonidos[nota.leer_nota()].play()
            self.lista_chips.pop(0)

```

A.3.4. LaneVisual.py

```
# -*- coding: utf-8 -*-
```

```
from LaneAuto import LaneAuto
```

```

class LaneVisual(LaneAuto):
    """
        Clase Lane Visual deriva del Lane Automatico ,
        Tiene la información gráfica del Lane
        y también la la configuración de los tiempos del reproductor
    """

def __init__(self , color , posicion , tiempo_pantalla):
    """
        Creamos los valores iniciales
    """

```

```

        Borramos el reproductor de sonidos .ogg que si que tienen los Auto

        color: color que se pondra en todas los chips de este lane
        posicion: posicion en la pantalla de este lane
        tiempo_pantalla: tiempo que tardara un chip en recorrer la pantalla
    """
    super(LaneVisual, self).__init__()

    self.color = color
    self.posicion = posicion
    self.tiempo_pantalla = tiempo_pantalla
    self.tiempo_poor = self.tiempo_pantalla / 2

def leer_color(self):
    """
        Devolvemos el valor del color del lane

        return: color
    """
    return self.color

def leer_pos(self):
    """
        Devolvemos el valor de la posicion del lane

        return: posicion
    """
    return self.posicion

def comprobar_y_poner_hit(self, tiempo_actual):
    """
        Encontrar la nota más cercana al tiempo actual, ponerla a hit
        y devolver la diferencia de tiempo para poder puntuar
        si no existe nota más cercana devolvemos valor de 10000 ms

        tiempo_actual: tiempo actual para comparar con tiempo de notas

        return: diferencia de tiempo en la nota más cercana
    """
    tiempo_menor = 10000
    nota_mas_cercana = None

    for nota in self.lista_chips:
        if not(nota.leer_hit()):
            #Si hay nota sin pulsar comprobamos tiempo
            if abs(nota.leer_tiempo() - tiempo_actual) < tiempo_menor:
                tiempo_menor = abs(nota.leer_tiempo() - tiempo_actual)
                nota_mas_cercana = nota
    if nota_mas_cercana:
        nota_mas_cercana.set_hit()
        self.banco_sonidos[nota_mas_cercana.leer_nota()].play()
    return tiempo_menor

```

```

def comprobar_tiempo(self, tiempo_actual):
    """
        Encontrar la nota más cercana al tiempo actual
        y devolver la diferencia de tiempo para poder puntuar
        si no existe nota más cercana devolvemos valor de 10000 ms

        tiempo_actual: tiempo actual para comparar con tiempo de notas

        return: diferencia de tiempo en la nota más cercana
    """
    tiempo_menor = 10000
    for nota in self.lista_chips:
        if not(nota.leer_hit()):
            #Si hay nota sin pulsar comprobamos tiempo
            if abs(nota.leer_tiempo() - tiempo_actual) < tiempo_menor:
                tiempo_menor = abs(nota.leer_tiempo() - tiempo_actual)
    return tiempo_menor

def eliminar_nota_antigua(self, tiempo_actual):
    """
        Si la primera nota en la lista es igual o mayor al tiempo actual
        más el tiempo posible de una nota con la peor puntuacion
        se ha de eliminar esa nota, tenemos que devolver true si eliminamos
        nota que no se ha pulsado ya que sera un miss

        tiempo_actual: tiempo actual

        return true si ha eliminado nota sin hit, false de otra manera
    """
    if self.lista_chips:
        nota = self.lista_chips[0]
        tiempo_final = nota.leer_tiempo() + self.tiempo_poor
        if tiempo_actual >= tiempo_final:
            estado = nota.leer_hit()
            self.lista_chips.pop(0)
            return not(estado)
    return False

```

A.3.5. main.py

```

#!/usr/bin/env Python3
# -*- coding: utf-8 -*-

```

```
import sys
```

```
from PyQt5 import QtWidgets
```

```

from MyMainWindowQt5 import MyMainWindowQt5
from MyWidgetNavegacion import MyWidgetNavegacion
from MyWidgetConfigurarDrum import MyWidgetConfigurarDrum
from MyWidgetConfigurarDTX import MyWidgetConfigurarDTX
from ReproductorDTX import ReproductorDTX

```

```

version = "1.0.0"

def main():
    """
    Programa principal creamos ventana principal, stacked widgets
    """

    print ("pyDTX", version, "<kapoir@gmail.com>")

    #Crear la aplicacion y cargar la configuracion de los archivos
    app = QtWidgets.QApplication(sys.argv)

    #Creamos la ventana principal
    mainWindow = MyMainWindowQt5()

    #Creamos las ventanas en stack para ya tenerlas en memoria
    ui_widgetNavegacion = MyWidgetNavegacion()
    mainWindow.ui.stackedWidget.insertWidget(1, ui_widgetNavegacion)
    ui_widgetConfigurarDrum = MyWidgetConfigurarDrum()
    mainWindow.ui.stackedWidget.insertWidget(2, ui_widgetConfigurarDrum)
    ui_widgetConfigurarDTX = MyWidgetConfigurarDTX()
    mainWindow.ui.stackedWidget.insertWidget(3, ui_widgetConfigurarDTX)
    ui_widgetReproductor = ReproductorDTX()
    mainWindow.ui.stackedWidget.insertWidget(4, ui_widgetReproductor)

    #Bucle principal
    mainWindow.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    """
    La llamada desde terminal ejecuta el codigo main()
    """
    main()

```

A.3.6. UtilidadesDTX.py

```

#-*- coding: utf-8 -*-

```

```

import os

```

```

def separar_strings_grupos_2(string):
    """
    Crear una lista separada en grupos de 2 caracteres para obtener
    normalmente las notas de un lane

    string: string original
    """

```

```

        return: string separado en grupos de 2 caracteres
    """
    return [string[i:i + 2] for i in range(0, len(string), 2)]

def es_notacion(string):
    """
        Quitando al string que nos pasan
        el primer caracter y los tres ultimos sabremos
        si es una notacion de objetos para un compas sino sera una orden

        string: string original

        return: true si quitando primer y tres últimos caracteres es un integer
    """
    try:
        int(string[1:-3])
        return True
    except:
        return False

def buscar_subdirectorios(directorio):
    """
        Dado un directorio crearemos una lista con sus subdirectorios

        directorio: directorio donde comprobaremos sus subdirectorios

        return : lista de subdirectorios
    """
    lista_directorios = []
    for name in os.listdir(directorio):
        posible_file = os.path.join(directorio, name)
        if os.path.isdir(posible_file):
            lista_directorios.append(name)
    return lista_directorios

def buscar_info_directorio(directorio):
    """
        Buscara informacion de los posibles archivos .dtx/.def en un directorio
        Es la información que se empleara en los menus

        directorio: directorio donde buscaremos la información

        return: info del directorio, lista vacia si no hay info
    """
    posible_file = os.path.join(directorio, 'set.def')
    if os.path.exists(posible_file):
        return conseguir_info_set_def(posible_file)
    posible_file = os.path.join(directorio, 'SET.DEF')
    if os.path.exists(posible_file):

```

```

        return conseguir_info_set_def(possible_file)
    posible_file = os.path.join(directorio, 'box.def')
    if os.path.exists(possible_file):
        print("TODO_Conseguir_info_box.def")
        return []
    posible_file = os.path.join(directorio, 'BOX.DEF')
    if os.path.exists(possible_file):
        print("TODO_Conseguir_info_BOX.DEF")
        return []
#Tendremos que buscar algún dtx al no existir ningún .def
    for name in os.listdir(directorio):
        posible_file = os.path.join(directorio, name)
        if name.endswith('.dtx') or name.endswith('.DTX'):
            return conseguir_info_dtx(possible_file)
    return []

def conseguir_info_set_def(archivo):
    """
        Analizamos el set.def y rellenamos una lista con 5 posibles items,
        cada uno con tres valores: label, file.dtx, información de ese .dtx

        archivo: archivo que analizaremos

        return: lista con la información de ese set.def
    """
    #Creamos la lista vacia, para añadir la información que encontremos
    lista_info = [[None, None, None],
                  [None, None, None],
                  [None, None, None],
                  [None, None, None],
                  [None, None, None]]

    fp = open(archivo, "r")
    while True:
        linea = fp.readline()
        if not linea:
            break
        #Limpiamos caracteres especiales de la linea
        linea = linea.rstrip('\r\n\t')
        linea = linea.split(";")

        #La posible orden es desde el inicio hasta el primer espacio
        primer_espacio = linea[0].find(" ")
        orden = linea[0][:primer_espacio]

        """Comprobamos si hay información tipo
        #XLABEL:
        #LXFILE:
        """
        if orden:
            if orden == "#L1LABEL:":
                lista_info[0][0] = linea[0][primer_espacio + 1:]

```

```

elif orden == "#L1FILE:":
    archivo_local = linea[0][primer_espacio + 1:]
    path = os.path.dirname(archivo)
    archivo_dtx = os.path.join(path, archivo_local)
    info_dtx = conseguir_info_dtx(archivo_dtx)
    lista_info[0][2] = info_dtx
    lista_info[0][1] = archivo_dtx
elif orden == "#L2LABEL:":
    lista_info[1][0] = linea[0][primer_espacio + 1:]
elif orden == "#L2FILE:":
    archivo_local = linea[0][primer_espacio + 1:]
    path = os.path.dirname(archivo)
    archivo_dtx = os.path.join(path, archivo_local)
    info_dtx = conseguir_info_dtx(archivo_dtx)
    lista_info[1][2] = info_dtx
    lista_info[1][1] = archivo_dtx
elif orden == "#L3LABEL:":
    lista_info[2][0] = linea[0][primer_espacio + 1:]
elif orden == "#L3FILE:":
    archivo_local = linea[0][primer_espacio + 1:]
    path = os.path.dirname(archivo)
    archivo_dtx = os.path.join(path, archivo_local)
    info_dtx = conseguir_info_dtx(archivo_dtx)
    lista_info[2][2] = info_dtx
    lista_info[2][1] = archivo_dtx
elif orden == "#L4LABEL:":
    lista_info[3][0] = linea[0][primer_espacio + 1:]
elif orden == "#L4FILE:":
    archivo_local = linea[0][primer_espacio + 1:]
    path = os.path.dirname(archivo)
    archivo_dtx = os.path.join(path, archivo_local)
    info_dtx = conseguir_info_dtx(archivo_dtx)
    lista_info[3][2] = info_dtx
    lista_info[3][1] = archivo_dtx
elif orden == "#L5LABEL:":
    lista_info[4][0] = linea[0][primer_espacio + 1:]
elif orden == "#L5FILE:":
    archivo_local = linea[0][primer_espacio + 1:]
    path = os.path.dirname(archivo)
    archivo_dtx = os.path.join(path, archivo_local)
    info_dtx = conseguir_info_dtx(archivo_dtx)
    lista_info[4][2] = info_dtx
    lista_info[4][1] = archivo_dtx
fp.close()
return lista_info

def conseguir_info_dtx(archivo):
    """
    Conseguir información de un .dtx, se devolvera una lista de 4 items
    imagen preview, titulo del .dtx, dificultad del .dtx, panel del .dtx

    archivo: archivo .dtx que analizaremos

```



```

        return: lista con la información del dtx
    """
    #Creamos la lista vacia
    lista_info = [None, None, None, None]
    fp = open(archivo, "r")
    while True:
        linea = fp.readline()
        if not linea:
            break
        #Limpiamos la linea de caracteres especiales
        linea = linea.rstrip('\r\n')
        linea = linea.split(";")

        #La posible orden es desde el inicio hasta el primer espacio
        primer_espacio = linea[0].find(" ")
        orden = linea[0][:primer_espacio]
        if orden:
            if orden == "#TITLE:":
                lista_info[1] = linea[0][primer_espacio + 1:]
            elif orden == "#DLEVEL:":
                lista_info[2] = linea[0][primer_espacio + 1:]
            elif orden == "#PREIMAGE:":
                archivo_local = linea[0][primer_espacio + 1:]
                path = os.path.dirname(archivo)
                archivo_imagen = os.path.join(path, archivo_local)
                lista_info[0] = archivo_imagen
            elif orden == "#PANEL:":
                lista_info[3] = linea[0][primer_espacio + 1:]
    fp.close()
    return lista_info

```

A.3.7. MyMainWindowQt5.py

```

# -*- coding: utf-8 -*-

import configparser

from PyQt5 import QtWidgets

from MainWindowQt5 import Ui_MainWindow

class MyMainWindowQt5(QtWidgets.QMainWindow):
    """
        Clase Main Window tendra el stackwidget principal, y por defecto pondra
        la pantalla inicial
    """

    def __init__(self):
        """
            Creamos los valores iniciales

```

```

"""
super(MyMainWindowQt5, self).__init__()

self.ui = Ui_MainWindow()
self.ui.setupUi(self)

#Conectamos los botones de la pantalla principal
self.ui.button_StartGame.pressed.connect(self.cambiar_a_navegacion)
self.ui.button_ConfigureDrum.pressed.connect(self.cambiar_drum_keys)
self.ui.button_ConfigureDTX.clicked.connect(self.cambiar_config_dtx)
self.ui.button_Quit.clicked.connect(self.salir)

#Cargamos configuracion de dtx ya que el tema depende de la conf
config_dtx = configparser.ConfigParser()
config_dtx.read('config_dtx.ini')
theme = config_dtx['Graphics']['theme']

#Ponemos el background dependiendo de la configuración
style_sheet = "#Principal{border-image: url(" + theme + \
              "/background_menu.jpg) stretch stretch" + ";}"

self.ui.Principal.setStyleSheet(style_sheet)

def cambiar_a_navegacion(self):
    """
        Cambiamos el stackedwidget al widget de navegación
    """
    self.ui.stackedWidget.setCurrentIndex(1)

def cambiar_drum_keys(self):
    """
        Cambiamos el stackedwidget al widget de configurar las teclas
    """
    self.ui.stackedWidget.setCurrentIndex(2)

def cambiar_config_dtx(self):
    """
        Cambiamos el stackedwidget al widget de configurar de pydtx
    """
    self.ui.stackedWidget.setCurrentIndex(3)

def cambiar_menu_principal(self):
    """
        Cambiamos el stackedwidget al widget inicial
    """
    self.ui.stackedWidget.setCurrentIndex(0)

def cambiar_reproduccion(self):
    """
        Cambiaremos al widget de reproduccion
        esta funcion la llamaran antes de comenzar una cancion
    """
    self.ui.stackedWidget.setCurrentIndex(4)

```

```
def salir(self):
    """
        Salir del programa, para ello cerramos el stacked widget
    """
    self.close()
```

A.3.8. MyWidgetConfigurarDrum.py

```
# -*- coding: utf-8 -*-

import configparser
import rtmidi2

from PyQt5 import QtWidgets

from WidgetConfigurarDrum import Ui_WidgetConfigurarDrum

class MyWidgetConfigurarDrum(QtWidgets.QWidget):
    """
        Clase Widget tendra sera donde se configuren tocas las teclas tanto
        las de teclado como las de bateria-MIDI
    """

    def __init__(self):
        """
            Creamos los valores iniciales
        """
        super(MyWidgetConfigurarDrum, self).__init__()

        self.ui = Ui_WidgetConfigurarDrum()
        self.ui.setupUi(self)

        #Leemos la configuracion
        self.leer_config_drum()

        #Conectamos los botones a sus funciones
        self.ui.exitButton.pressed.connect(self.exit)
        self.ui.saveConfigurationButton.pressed.connect(self.save_config_drum)
        self.ui.portsConnectedButton.pressed.connect(self.view_ports)
        self.ui.viewNotesButton.pressed.connect(self.view_note)

        self.midi_in = None

    def view_ports(self):
        """
            Mostramos en el text asociado, que puertos (instrumentos) MIDI
            vemos conectados.
        """
        #Creamos el texto para que sea más legible que puertos estamos viendo
        i = 0
```

```

text = ""
for port in rtmidi2.get_in_ports():
    text = text + str(i) + "□:□" + port + "\n"
    i = i + 1
#Ponemos en el label asociado todos los puertos que estamos viendo
self.ui.portsConnectedTextEdit.setPlainText(text)

def view_note(self):
    """
        Conectamos la bateria-MIDI al midi Handler
    """

def midi_handler(message, time_stamp):
    """
        Si detectamos un mensaje MIDI de tipo note-on ponemos la nota
        en el line edit asociado

        message: message MIDI
        time_stamp: tiempo diferido desde el mensaje hasta ahora
    """
    if message[0] == 153:
        #Nota On
        self.ui.viewNotesLineEdit.setText(str(message[1]))

self.midi_in = rtmidi2.MidiIn()
midi_port = self.ui.portMidiSpinBox.value()
if midi_port < len(rtmidi2.get_in_ports()):
    #Puede estar desconectado al menos comprobar que cuadra números
    self.midi_in.open_port(midi_port)
    #Juntamos el midi a la funcion de entrada
self.midi_in.callback = midi_handler

def exit(self):
    """
        Volvemos al menu principal, para ello apuntamos en el stackedwidget
        al widget principal y desactivamos el MIDI
    """
    self.midi_in = None
    self.ui.viewNotesLineEdit.setText("")
    self.parentWidget().setCurrentIndex(0)

def leer_lane_drum(self, lane, auto, midi1, midi2, key):
    """
        Coger los valores del lane según configuración y ponerlos
        en las variables

        lane: lane de la que leeremos los datos
        auto: checkbox a poner el valor auto
        midi1: Spinbox a poner el valor del midi1 de este lane
        midi2: Spinbox a poner el valor del midi2 de este lane
        key: Textbox a poner el valor del teclado de este lane
    """
    if self.config_drum[lane]['auto'] == 'true':

```

```

        auto.setChecked(True)
    else:
        auto.setChecked(False)
    midi1.setValue(int(self.config_drum[lane]['midi_1']))
    midi2.setValue(int(self.config_drum[lane]['midi_2']))
    key.setText(self.config_drum[lane]['key'])

def leer_config_drum(self):
    """
        Leemos el fichero de configuración del drum y modificamos todos
        los elementos gráficos

    """
    #Leemos el fichero de configuración
    self.config_drum = configparser.ConfigParser()
    self.config_drum.read('config_drum.ini')
    midi_port = self.config_drum['MIDI']['midi_port']
    self.ui.portMidiSpinBox.setValue(int(midi_port))

    #Lane11
    self.leer_lane_drum('Lane_11',
                        self.ui.auto_lane11_CheckBox,
                        self.ui.midi1_lane11_SpinBox,
                        self.ui.midi2_lane11_SpinBox,
                        self.ui.key_lane_11_LineEdit)

    #Lane12
    self.leer_lane_drum('Lane_12',
                        self.ui.auto_lane12_CheckBox,
                        self.ui.midi1_lane12_SpinBox,
                        self.ui.midi2_lane12_SpinBox,
                        self.ui.key_lane_12_LineEdit)

    #Lane13
    self.leer_lane_drum('Lane_13',
                        self.ui.auto_lane13_CheckBox,
                        self.ui.midi1_lane13_SpinBox,
                        self.ui.midi2_lane13_SpinBox,
                        self.ui.key_lane_13_LineEdit)

    #Lane14
    self.leer_lane_drum('Lane_14',
                        self.ui.auto_lane14_CheckBox,
                        self.ui.midi1_lane14_SpinBox,
                        self.ui.midi2_lane14_SpinBox,
                        self.ui.key_lane_14_LineEdit)

    #Lane15
    self.leer_lane_drum('Lane_15',
                        self.ui.auto_lane15_CheckBox,
                        self.ui.midi1_lane15_SpinBox,
                        self.ui.midi2_lane15_SpinBox,
                        self.ui.key_lane_15_LineEdit)

    #Lane16
    self.leer_lane_drum('Lane_16',
                        self.ui.auto_lane16_CheckBox,
                        self.ui.midi1_lane16_SpinBox,

```

```

        self.ui.midi2_lane16_SpinBox ,
        self.ui.key_lane_16_LineEdit)

#Lane17
self.leer_lane_drum( 'Lane_17' ,
                    self.ui.auto_lane17_CheckBox ,
                    self.ui.midi1_lane17_SpinBox ,
                    self.ui.midi2_lane17_SpinBox ,
                    self.ui.key_lane_17_LineEdit)

#Lane18
self.leer_lane_drum( 'Lane_18' ,
                    self.ui.auto_lane18_CheckBox ,
                    self.ui.midi1_lane18_SpinBox ,
                    self.ui.midi2_lane18_SpinBox ,
                    self.ui.key_lane_18_LineEdit)

#Lane19
self.leer_lane_drum( 'Lane_19' ,
                    self.ui.auto_lane19_CheckBox ,
                    self.ui.midi1_lane19_SpinBox ,
                    self.ui.midi2_lane19_SpinBox ,
                    self.ui.key_lane_19_LineEdit)

#Lane1A
self.leer_lane_drum( 'Lane_1A' ,
                    self.ui.auto_lane1A_CheckBox ,
                    self.ui.midi1_lane1A_SpinBox ,
                    self.ui.midi2_lane1A_SpinBox ,
                    self.ui.key_lane_1A_LineEdit)

#Lane1B
self.leer_lane_drum( 'Lane_1B' ,
                    self.ui.auto_lane1B_CheckBox ,
                    self.ui.midi1_lane1B_SpinBox ,
                    self.ui.midi2_lane1B_SpinBox ,
                    self.ui.key_lane_1B_LineEdit)

#Lane1C
self.leer_lane_drum( 'Lane_1C' ,
                    self.ui.auto_lane1C_CheckBox ,
                    self.ui.midi1_lane1C_SpinBox ,
                    self.ui.midi2_lane1C_SpinBox ,
                    self.ui.key_lane_1C_LineEdit)

def guardar_lane_drum(self , lane , auto , midi1 , midi2 , key):
    """
        Guardamos en la configuración los valores del lane

        lane: lane de la que guardaremos los datos
        auto: valor auto de la lane
        midi1: valor del midi1 de este lane
        midi2: valor del midi2 de este lane
        key: valor del teclado de este lane
    """
    self.config_drum[lane] = {}
    if auto:
        self.config_drum[lane]['auto'] = 'true'
    else:

```

```

        self.config_drum[lane]['auto'] = 'false'
self.config_drum[lane]['midi_1'] = midi1
self.config_drum[lane]['midi_2'] = midi2
self.config_drum[lane]['key'] = key

def save_config_drum(self):
    """
        Salvamos los valores del widget en el fichero de configuración
    """
    self.config_drum['MIDI'] = {}
    midi_port = self.ui.portMidiSpinBox.value()
    self.config_drum['MIDI']['midi_port'] = str(midi_port)

#Lane 11
self.guardar_lane_drum('Lane_11',
                        self.ui.auto_lane11_CheckBox.checkState(),
                        str(self.ui.midi1_lane11_SpinBox.value()),
                        str(self.ui.midi2_lane11_SpinBox.value()),
                        self.ui.key_lane_11_LineEdit.text())

#Lane 12
self.guardar_lane_drum('Lane_12',
                        self.ui.auto_lane12_CheckBox.checkState(),
                        str(self.ui.midi1_lane12_SpinBox.value()),
                        str(self.ui.midi2_lane12_SpinBox.value()),
                        self.ui.key_lane_12_LineEdit.text())

#Lane 13
self.guardar_lane_drum('Lane_13',
                        self.ui.auto_lane13_CheckBox.checkState(),
                        str(self.ui.midi1_lane13_SpinBox.value()),
                        str(self.ui.midi2_lane13_SpinBox.value()),
                        self.ui.key_lane_13_LineEdit.text())

#Lane 14
self.guardar_lane_drum('Lane_14',
                        self.ui.auto_lane14_CheckBox.checkState(),
                        str(self.ui.midi1_lane14_SpinBox.value()),
                        str(self.ui.midi2_lane14_SpinBox.value()),
                        self.ui.key_lane_14_LineEdit.text())

#Lane 15
self.guardar_lane_drum('Lane_15',
                        self.ui.auto_lane15_CheckBox.checkState(),
                        str(self.ui.midi1_lane15_SpinBox.value()),
                        str(self.ui.midi2_lane15_SpinBox.value()),
                        self.ui.key_lane_15_LineEdit.text())

#Lane 16
self.guardar_lane_drum('Lane_16',
                        self.ui.auto_lane16_CheckBox.checkState(),
                        str(self.ui.midi1_lane16_SpinBox.value()),
                        str(self.ui.midi2_lane16_SpinBox.value()),
                        self.ui.key_lane_16_LineEdit.text())

#Lane 17
self.guardar_lane_drum('Lane_17',
                        self.ui.auto_lane17_CheckBox.checkState(),
                        str(self.ui.midi1_lane17_SpinBox.value()),

```

```

        str(self.ui.midi2_lane17_SpinBox.value()),
        self.ui.key_lane_17_LineEdit.text())

#Lane 18
self.guardar_lane_drum('Lane_18',
                        self.ui.auto_lane18_CheckBox.checkState(),
                        str(self.ui.midi1_lane18_SpinBox.value()),
                        str(self.ui.midi2_lane18_SpinBox.value()),
                        self.ui.key_lane_18_LineEdit.text())

#Lane 19
self.guardar_lane_drum('Lane_19',
                        self.ui.auto_lane19_CheckBox.checkState(),
                        str(self.ui.midi1_lane19_SpinBox.value()),
                        str(self.ui.midi2_lane19_SpinBox.value()),
                        self.ui.key_lane_19_LineEdit.text())

#Lane 1A
self.guardar_lane_drum('Lane_1A',
                        self.ui.auto_lane1A_CheckBox.checkState(),
                        str(self.ui.midi1_lane1A_SpinBox.value()),
                        str(self.ui.midi2_lane1A_SpinBox.value()),
                        self.ui.key_lane_1A_LineEdit.text())

#Lane 1B
self.guardar_lane_drum('Lane_1B',
                        self.ui.auto_lane1B_CheckBox.checkState(),
                        str(self.ui.midi1_lane1B_SpinBox.value()),
                        str(self.ui.midi2_lane1B_SpinBox.value()),
                        self.ui.key_lane_1B_LineEdit.text())

#Lane 1C
self.guardar_lane_drum('Lane_1C',
                        self.ui.auto_lane1C_CheckBox.checkState(),
                        str(self.ui.midi1_lane1C_SpinBox.value()),
                        str(self.ui.midi2_lane1C_SpinBox.value()),
                        self.ui.key_lane_1C_LineEdit.text())

#Guardamos en el fichero
with open('config_drum.ini', 'w') as configfile:
    self.config_drum.write(configfile)

```

A.3.9. MyWidgetConfigurarDTX.py

```
# -*- coding: utf-8 -*-
```

```
import configparser
```

```
from PyQt5 import QtWidgets
```

```
from WidgetConfigurarDTX import Ui_WidgetConfigurarDTX
```

```
class MyWidgetConfigurarDTX(QtWidgets.QWidget):
```

```
    """
```

```
    Clase Widget tendra sera donde se configuren todas las teclas tanto
    las de teclado como las de bateria-MIDI
```

```
    """
```



```

def __init__(self):
    """
        Creamos los valores iniciales
    """
    super(MyWidgetConfigurarDTX, self).__init__()

    self.ui = Ui_WidgetConfigurarDTX()
    self.ui.setupUi(self)

    #Conectamos los botones
    self.ui.exitButton.pressed.connect(self.exit)
    self.ui.saveButton.pressed.connect(self.save_config_dtx)
    self.ui.speed_SpinBox.valueChanged.connect(self.visualizar_tiempos)

    #Leemos la configuracion y ponemos los valores en la información gráfica
    self.leer_config_dtx()

def exit(self):
    """
        Volvemos al menu principal, para ello apuntamos en el stackedwidget
        al principal
    """
    self.parentWidget().setCurrentIndex(0)

def leer_config_dtx(self):
    """
        Leemos el fichero de configuración del dtx, y según sus valores
        ponemos los elementos gráficos
    """
    #Leemos el fichero de configuración
    self.config_dtx = configparser.ConfigParser()
    self.config_dtx.read('config_dtx.ini')
    #Ponemos los elementos gráficos según el fichero
    if self.config_dtx['Graphics']['OpenGL'] == 'true':
        self.ui.checkBox_Gl.setChecked(True)
    else:
        self.ui.checkBox_Gl.setChecked(False)
    self.ui.theme_LineEdit.setText(self.config_dtx['Graphics']['theme'])
    self.ui.songs_LineEdit.setText(self.config_dtx['Songs']['path'])
    tiempo = int(self.config_dtx['DTX']['speed'])
    self.ui.speed_SpinBox.setValue(tiempo)
    self.visualizar_tiempos(tiempo)

def save_config_dtx(self):
    """
        Analizamos los elementos gráficos y según ellos escribimos en el
        fichero de configuración
    """
    self.config_dtx['DTX'] = {}
    self.config_dtx['DTX']['speed'] = str(self.ui.speed_SpinBox.value())

    theme = self.ui.theme_LineEdit.text()

```

```

self.config_dtx['Graphics']['theme'] = theme
songs = self.ui.songs_LineEdit.text()
self.config_dtx['Songs']['path'] = songs
if self.ui.checkBox_Gl.checkState():
    self.config_dtx['Graphics']['opengl'] = 'true'
else:
    self.config_dtx['Graphics']['opengl'] = 'false'
#Grabamos el fichero
with open('config_dtx.ini', 'w') as configfile:
    self.config_dtx.write(configfile)

def visualizar_tiempos(self, tiempo):
    """
        Calculamos los tiempos de sincronización y los ponemos en sus
        elementos gráficos para que de esta manera se pueda conocer el nivel
        de sincronización

        tiempo: tiempo base que se empleara para calcular la sincronización
    """
    #Calculamos los valores según tiempo
    #tiempo base es el tiempo que tarda un chip en recorrer toda la pantalla
    tiempo_limpio = str(round(tiempo / 15, 2))
    perfect = "-" + tiempo_limpio + ".." + tiempo_limpio + "ms"
    tiempo_limpio = str(round(tiempo / 10, 2))
    great = "-" + tiempo_limpio + ".." + tiempo_limpio + "ms"
    tiempo_limpio = str(round(tiempo / 5, 2))
    good = "-" + tiempo_limpio + ".." + tiempo_limpio + "ms"
    tiempo_limpio = str(round(tiempo / 2, 2))
    poor = "-" + tiempo_limpio + ".." + tiempo_limpio + "ms"

    #Actualizmos los Labels
    self.ui.perfect_time_Label.setText(perfect)
    self.ui.great_time_Label.setText(great)
    self.ui.good_time_Label.setText(good)
    self.ui.poor_time_Label.setText(poor)

```

A.3.10. MyWidgetNavegacion.py

```
# -*- coding: utf-8 -*-
```

```

import os
import configparser

import UtilidadesDTX

from PyQt5 import QtWidgets
from PyQt5 import QtGui

from WidgetNavegacion import Ui_WidgetNavegacion

class MyWidgetNavegacion(QtWidgets.QWidget):

```

```

"""
    Classe Widget donde se podra navegar entre los directorios de canciones
    viendo la información de los .dtx y .def además de poder configurar el
    nivel de sincronización
"""

def __init__(self):
    """
        Creamos los valores iniciales
    """
    super(MyWidgetNavegacion, self).__init__()

    self.ui = Ui_WidgetNavegacion()
    self.ui.setupUi(self)
    self.ui.listWidget.itemSelectionChanged.connect(self.cambio_seleccion)

    #Conectamos los botones a sus funciones
    self.ui.downButton.clicked.connect(self.seleccion_abajo)
    self.ui.upButton.clicked.connect(self.seleccion_arriba)
    self.ui.backButton.clicked.connect(self.seleccion_back)
    self.ui.selectButton.clicked.connect(self.seleccion_select)
    self.ui.changedirButton.clicked.connect(self.seleccion_changedir)
    self.ui.l1_Button.clicked.connect(self.seleccion_select_l1)
    self.ui.l2_Button.clicked.connect(self.seleccion_select_l2)
    self.ui.l3_Button.clicked.connect(self.seleccion_select_l3)
    self.ui.l4_Button.clicked.connect(self.seleccion_select_l4)
    self.ui.l5_Button.clicked.connect(self.seleccion_select_l5)
    self.ui.speed_inc_Button.clicked.connect(self.speed_inc)
    self.ui.speed_dec_Button.clicked.connect(self.speed_dec)

    #Instalamos los filtros en los botones de l1..l5 para poder actualizar
    self.ui.l1_Button.installEventFilter(self)
    self.ui.l2_Button.installEventFilter(self)
    self.ui.l3_Button.installEventFilter(self)
    self.ui.l4_Button.installEventFilter(self)
    self.ui.l5_Button.installEventFilter(self)

    #Leemos la configuracion de pydtx
    config_dtx = configparser.ConfigParser()
    config_dtx.read('config_dtx.ini')

    #Leemos la configuracion del dtx para tener el tiempo de pantalla
    self.config_dtx = configparser.ConfigParser()
    self.config_dtx.read('config_dtx.ini')
    self.tiempo_pantalla = int(self.config_dtx['DIX']['speed'])
    self.ui.speed_label.setText(str(self.tiempo_pantalla))

    self.path_songs = config_dtx['Songs']['path']
    self.path_songs_inicial = config_dtx['Songs']['path']

    #Rehacemos toda la lista de directorios
    self.rehacer_lista()

```

```

def eventFilter(self, source, event):
    """
        Si se activa el focus en un button de l1..l5 tenemos que actualizar
        la información en pantalla
    """
    if source == self.ui.l1_Button:
        if event.type() == QtGui.QFocusEvent.FocusIn:
            #Han seleccionado el L1
            datos = self.lista_info_items[self.seleccionado][4][2]
            self.actualizar_datos(datos)
    elif source == self.ui.l2_Button:
        if event.type() == QtGui.QFocusEvent.FocusIn:
            #Han seleccionado el L2
            datos = self.lista_info_items[self.seleccionado][3][2]
            self.actualizar_datos(datos)
    elif source == self.ui.l3_Button:
        if event.type() == QtGui.QFocusEvent.FocusIn:
            #Han seleccionado el L3
            datos = self.lista_info_items[self.seleccionado][2][2]
            self.actualizar_datos(datos)
    elif source == self.ui.l4_Button:
        if event.type() == QtGui.QFocusEvent.FocusIn:
            #Han seleccionado el L4
            datos = self.lista_info_items[self.seleccionado][1][2]
            self.actualizar_datos(datos)
    elif source == self.ui.l5_Button:
        if event.type() == QtGui.QFocusEvent.FocusIn:
            #Han seleccionado el L5
            datos = self.lista_info_items[self.seleccionado][0][2]
            self.actualizar_datos(datos)
    #Devolvemos False para que el handler de Qt5 pueda continuar su trabajo
    return False

def speed_inc(self):
    """
        Tenemos que disminuir el tiempo de pantalla
        comprobar limite de 100ms
    """
    if self.tiempo_pantalla > 100:
        self.tiempo_pantalla = self.tiempo_pantalla - 100
        self.ui.speed_label.setText(str(self.tiempo_pantalla))

def speed_dec(self):
    """
        Tenemos que aumentar el tiempo de pantalla
        comprobar limite de 5000ms
    """
    if self.tiempo_pantalla < 5000:
        self.tiempo_pantalla = self.tiempo_pantalla + 100
        self.ui.speed_label.setText(str(self.tiempo_pantalla))

def seleccion_abajo(self):
    """

```

```

        Tenemos que seleccionar siguiente objeto de la lista de items y
        actualizar
    """
    if ((self.seleccionado + 1) < len(self.lista_items)):
        self.seleccionado = self.seleccionado + 1
        self.ui.listWidget.setCurrentRow(self.seleccionado)

def seleccion_arriba(self):
    """
        Tenemos que seleccionar objeto anterior de la lista de items y
        actualizar
    """
    if self.seleccionado > 0:
        self.seleccionado = self.seleccionado - 1
        self.ui.listWidget.setCurrentRow(self.seleccionado)

def seleccion_back(self):
    """
        Intentamos subir un nivel en el árbol de directorio si hemos llegado
        al nivel inicial volvemos al menú principal
    """
    if self.path_songs == self.path_songs_inicial:
        self.parentWidget().setCurrentIndex(0)
    else:
        self.path_songs = os.path.dirname(self.path_songs)
        self.rehacer_lista()

def cambio_seleccion(self):
    """
        Se ha cambiado la seleccion de la lista de items, por lo que se
        tiene que actualizar toda la información de la pantalla
    """
    self.seleccionado = self.ui.listWidget.currentRow()
    #Borramos toda la información de las canciones de la pantalla
    self.imagenCancion = None
    self.ui.imagenCancion.clear()
    self.ui.label_name.setText("Author:")
    self.ui.label_dificulty.setText("Dificulty:")
    self.ui.label_others.setText("")
    #Desactivamos los botones de l1..l5
    self.ui.l1_Button.hide()
    self.ui.l2_Button.hide()
    self.ui.l3_Button.hide()
    self.ui.l4_Button.hide()
    self.ui.l5_Button.hide()
    #Desactivamos el botón de un .dtx
    self.ui.selectButton.hide()
    #Desactivamos el botón de cambiar de directorio
    self.ui.changedirButton.hide()
    #Pasamos a analizar el elemento seleccionado
    if len(self.lista_info_items[self.seleccionado]) == 4:
        #Es un .dtx
        self.ui.selectButton.show()

```

```

        self.actualizar_datos(self.lista_info_items[self.seleccionado])
    elif len(self.lista_info_items[self.seleccionado]) == 5:
        #Estamos en un set.def, actualizamos los l1..l5
        self.actualizar_button_label(self.ui.l1_Button, 4)
        self.actualizar_button_label(self.ui.l2_Button, 3)
        self.actualizar_button_label(self.ui.l3_Button, 2)
        self.actualizar_button_label(self.ui.l4_Button, 1)
        self.actualizar_button_label(self.ui.l5_Button, 0)
    else:
        #Es un directorio
        self.ui.changedirButton.show()

def actualizar_button_label(self, button, posicion):
    """
        Tenemos que actualizar un botón de estilo lX con los valores
        que nos indican

        button: boton que tenemos que modificar
        posicion: apunta a los datos que tenemos que usar
    """
    info_label = self.lista_info_items[self.seleccionado][posicion]
    if info_label[1]:
        button.setText(info_label[0])
        button.show()
        self.actualizar_datos(info_label[2])

def actualizar_datos(self, datos):
    """
        Actualizar los datos de la canción con la informacion que nos pasan

        datos: datos de la cancion
    """
    if datos[0]:
        self.imagenCancion = QtGui.QPixmap(datos[0])
        self.ui.imagenCancion.setPixmap(self.imagenCancion)
    if datos[1]:
        self.ui.label_name.setText(datos[1])
    if datos[2]:
        self.ui.label_difficulty.setText(datos[2])
    if datos[3]:
        self.ui.label_others.setText(datos[3])

def seleccion_changedir(self):
    """
        Tenemos que cambiar el directorio actual al seleccionado
    """
    self.path_songs = self.lista_items[self.seleccionado]
    self.rehacer_lista()

def seleccion_select(self):
    """
        Selecciona el .dtx y comenzamos la cancion con el tiempo actual
    """

```

```

self.parentWidget().setCurrentIndex(4)
path = self.lista_items[self.seleccionado]
encontrado = False
#Buscamos el dtx de ese directorio
for name in os.listdir(path):
    dtx_file = os.path.join(path, name)
    if name.endswith('.dtx') or name.endswith('.DTX'):
        if not(encontrado):
            #Comenzamos el reproductor
            widgetDTX = self.parentWidget().currentWidget()
            widgetDTX.reproducir_cancion(dtx_file, self.tiempo_pantalla)
            encontrado = True

def seleccion_label(self, selec):
    """
        Estamos en un boton de L1..L5 tenemos que comenzar la canción
        con el tiempo actual

        selec: apuntara al .dtx de este label
    """
    self.parentWidget().setCurrentIndex(4)
    path = self.lista_info_items[self.seleccionado][selec][1]
    widgetDTX = self.parentWidget().currentWidget()
    widgetDTX.reproducir_cancion(path, self.tiempo_pantalla)

def seleccion_select_l1(self):
    """
        Se ha pulsado el boton l1 comenzamos la canción de este botón
    """
    self.seleccion_label(4)

def seleccion_select_l2(self):
    """
        Se ha pulsado el boton l2 comenzamos la canción de este botón
    """
    self.seleccion_label(3)

def seleccion_select_l3(self):
    """
        Se ha pulsado el boton l3 comenzamos la canción de este botón
    """
    self.seleccion_label(2)

def seleccion_select_l4(self):
    """
        Se ha pulsado el boton l4 comenzamos la canción de este botón
    """
    self.seleccion_label(1)

def seleccion_select_l5(self):
    """
        Se ha pulsado el boton l5 comenzamos la canción de este botón
    """

```

```

        self.seleccion_label(0)

    def rehacer_lista(self):
        """
        Tenemos que buscar toda la información dentro del directorio actual
        """
        #Borramos toda la información gráfica
        self.ui.listWidget.clear()
        self.lista_items = []
        self.lista_info_items = []
        #Buscamos todos los subdirectorios
        lista_subdirec = UtilidadesDTX.buscar_subdirectorios(self.path_songs)
        for name in lista_subdirec:
            #Actualizamos por cada subdirectorio
            self.ui.listWidget.addItem(name)
            posible_file = os.path.join(self.path_songs, name)
            self.lista_items.append(posible_file)
            info_directorio = UtilidadesDTX.buscar_info_directorio(posible_file)
            self.lista_info_items.append(info_directorio)
        #Si hay subdirectorios ya seleccionamos el primero
        self.seleccionado = 0
        if len(self.lista_items) > 0:
            self.ui.listWidget.setCurrentRow(self.seleccionado)

```

A.3.11. MyWidgetPuntuacion.py

```
# -*- coding: utf-8 -*-
```

```
from PyQt5 import QtWidgets
```

```
from WidgetPuntuacion import Ui_WidgetPuntuacion
```

```
class MyWidgetPuntuacion(QtWidgets.QWidget):
```

```
    """
    Widget que se empleara para poner la puntuación mientras se reproduce
    la canción
    """
```

```
    def __init__(self, tiempo_pantalla, title):
```

```
        """
        Creamos los valores iniciales

        tiempo_pantalla: para calcular nivel de sincronización con las notas
        title: información del título que se pondra encima de la puntuación
        """
```

```
        super(MyWidgetPuntuacion, self).__init__()
```

```
        self.ui = Ui_WidgetPuntuacion()
        self.ui.setupUi(self)
```

```
        #Ponemos todos los valores a 0
```



```

self.combo = 0
self.num_notas = 0
self.max_combo = 0
self.perfect = 0
self.great = 0
self.good = 0
self.poor = 0
self.miss = 0

#Ponemos el titulo
self.ui.title_text_label.setText(title)

#Calculamos los valores de sincronización de las notas
self.tiempo_perfect = tiempo_pantalla / 15
self.tiempo_great = tiempo_pantalla / 10
self.tiempo_good = tiempo_pantalla / 5
self.tiempo_poor = tiempo_pantalla / 2

#Ponemos en los textLabels los valores iniciales
self.ui.perfect_text_Label.setText(str(self.perfect))
self.ui.perfect_per_Label.setText("%")
self.ui.great_text_Label.setText(str(self.great))
self.ui.great_per_Label.setText("%")
self.ui.good_text_Label.setText(str(self.good))
self.ui.good_per_Label.setText("%")
self.ui.poor_text_Label.setText(str(self.poor))
self.ui.poor_per_Label.setText("%")
self.ui.miss_text_Label.setText(str(self.miss))
self.ui.miss_per_Label.setText("%")
self.ui.max_combo_text_Label.setText(str(self.max_combo))
self.ui.max_combo_per_Label.setText("%")

def comprobar_puntuacion(self, tiempo_puntuacion):
    """
        Actualizamos el num_de_notas y comparamos el tiempo_puntuación con
        la sincronización de la canción para puntuar y actualizar
        la puntuación visualmente en concordancia

        Mientras sea great o good se va aumentando el combo actual.

        tiempo_puntuacion: tiempo de diferencia con una nota
    """
    self.num_notas = self.num_notas + 1

    if tiempo_puntuacion <= self.tiempo_perfect:
        #Tiempo esta en perfect
        self.perfect = self.perfect + 1
        self.combo = self.combo + 1
        if self.combo > self.max_combo:
            self.max_combo = self.combo
    elif tiempo_puntuacion <= self.tiempo_great:
        #Tiempo esta en great
        self.great = self.great + 1

```

```

        self.combo = self.combo + 1
        if self.combo > self.max_combo:
            self.max_combo = self.combo
    elif tiempo_puntuacion <= self.tiempo_good:
        #Tiempo esta en great, pondremos combo a 0
        self.good = self.good + 1
        self.combo = 0
    elif tiempo_puntuacion <= self.tiempo_poor:
        #Tiempo esta en poor, pondremos combo a 0
        self.poor = self.poor + 1
        self.combo = 0
    else:
        #Es un miss, pondremos combo a 0
        self.miss = self.miss + 1
        self.combo = 0
    self.actualizar_puntuacion()

def actualizar_puntuacion(self):
    """
        Actualizamos todos los textos del widget de puntuación
    """
    #Ponemos cuantas notas de cada llevamos actualmente
    self.ui.perfect_text_Label.setText(str(self.perfect))
    self.ui.great_text_Label.setText(str(self.great))
    self.ui.good_text_Label.setText(str(self.good))
    self.ui.poor_text_Label.setText(str(self.poor))
    self.ui.miss_text_Label.setText(str(self.miss))
    self.ui.max_combo_text_Label.setText(str(self.max_combo))

    #Preparamos los textos de los porcentajes
    perfect_float = round((self.perfect / self.num_notas) * 100, 2)
    perfect_per = str(perfect_float) + "%"
    great_float = round((self.great / self.num_notas) * 100, 2)
    great_per = str(great_float) + "%"
    good_float = round((self.good / self.num_notas) * 100, 2)
    good_per = str(good_float) + "%"
    poor_float = round((self.poor / self.num_notas) * 100, 2)
    poor_per = str(poor_float) + "%"
    miss_float = round((self.miss / self.num_notas) * 100, 2)
    miss_per = str(miss_float) + "%"
    max_combo_float = round((self.max_combo / self.num_notas) * 100, 2)
    max_combo_per = str(max_combo_float) + "%"

    #Ponemos los textos de los porcentajes
    self.ui.perfect_per_Label.setText(perfect_per)
    self.ui.great_per_Label.setText(great_per)
    self.ui.good_per_Label.setText(good_per)
    self.ui.poor_per_Label.setText(poor_per)
    self.ui.miss_per_Label.setText(miss_per)
    self.ui.max_combo_per_Label.setText(max_combo_per)

def aumentar_miss(self):
    """

```

```

        Esta función normalmente se llamara ya que una nota no ha sido
        tocada, por lo que actualizamos puntuación y puntuación visual en
        concordancia
    """
    self.num_notas = self.num_notas + 1
    self.miss = self.miss + 1
    self.combo = 0
    self.actualizar_puntuacion()

```

A.3.12. ReproductorDTX.py

```

#!/usr/bin/env Python
#-*- coding: utf-8 -*-
import configparser
import os
import rtmidi2

import UtilidadesDTX

from PyQt5 import QtCore
from PyQt5 import QtWidgets
from PyQt5.QtMultimedia import QMediaPlayer
from PyQt5.QtMultimedia import QMediaContent
from PyQt5.QtMultimedia import QSoundEffect

from Chip import Chip
from ChipAuto import ChipAuto
from LaneAuto import LaneAuto
from LaneVisual import LaneVisual
from MyWidgetPuntuacion import MyWidgetPuntuacion

class ReproductorDTX(QtWidgets.QGraphicsView):
    """
        Clase QGraphicsView ya que pintaremos la canción en sí, además de hacer
        todo el control de la canción
    """
    def __init__(self):
        """
            Creamos los valores iniciales
        """
        QtWidgets.QGraphicsView.__init__(self)
        self.borrar_datos_cancion()
        #Leemos configuracion para configurar dependiendo de valores
        self.leer_config_dtx()

        if self.config_dtx['Graphics']['OpenGL'] == 'true':
            self.setViewport(QtWidgets.QOpenGLWidget())

        #Desactivamos que salgan las barras de scroll
        self.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)

```

```

self.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)

def resizeEvent(self, event):
    """
        Esta función es para cuando se haga un resize de la ventana
        siempre estamos viendo la canción redimensionada en consecuencia
        para ello volvemos a definir el la resolución interna del visor

        event: el evento que ha disparado este resize
    """
    self.fitInView(QtCore.QRectF(0, 0, 1264, 714))

def keyPressEvent(self, event):
    """
        Han pulsado una tecla, tenemos que comprobar si es de un lane
        la puntuación

        event: el evento de Key que ha disparado esta función
    """
    if event.key() == QtCore.Qt.Key_Escape:
        #Tecla especial para salir
        self.salir()
    elif event.text() in self.dict_teclado:
        #Tenemos esta tecla definida en la configuración de teclado
        tiempo_actual = float(self.timer.elapsed())
        lane_selec = self.dict_teclado[event.text()]
        lane = self.dict_lanes_visual[lane_selec]
        tiempo_puntuacion = lane.comprobar_y_poner_hit(tiempo_actual)
        self.puntuacion.comprobar_puntuacion(tiempo_puntuacion)

def configurar_teclado_lane(self, lane_en_conf, lane):
    """
        Si el lane no esta en auto cogemos la key y la ponemos en el dicc

        lane_en_conf: Nombre de lane según configuración
        lane: Nombre que se pondra en el diccionario de teclas
    """
    if self.config_drum[lane_en_conf]['auto'] == 'false':
        tecla = self.config_drum[lane_en_conf]['key']
        self.dict_teclado[tecla] = lane

def configurar_teclado(self):
    """
        Según la configuracion guardada se crea el teclado que usaremos
        para comprobar si se pulsa una tecla a que lane le corresponde
    """
    self.dict_teclado = {}
    self.configurar_teclado_lane('Lane_11', '11')
    self.configurar_teclado_lane('Lane_12', '12')
    self.configurar_teclado_lane('Lane_13', '13')
    self.configurar_teclado_lane('Lane_14', '14')
    self.configurar_teclado_lane('Lane_15', '15')
    self.configurar_teclado_lane('Lane_16', '16')

```

```

self.configurar_teclado_lane('Lane_17', '17')
self.configurar_teclado_lane('Lane_18', '18')
self.configurar_teclado_lane('Lane_19', '19')
self.configurar_teclado_lane('Lane_1A', '1A')
self.configurar_teclado_lane('Lane_1B', '1B')
self.configurar_teclado_lane('Lane_1C', '1C')

def configurar_midi_lane(self, lane_en_conf, lane):
    """
        Dado un lane en concreto tenemos que mirar la configuracion para
        poner sus posibles notas-midi en el dicc, cuidado que puede estar
        duplicadas

        lane_en_conf: Nombre de lane según configuración
        lane: Nombre que se pondra en el diccionario de midi
    """
    if self.config_drum[lane_en_conf]['auto'] == 'false':
        midi1 = int(self.config_drum[lane_en_conf]['midi_1'])
        midi2 = int(self.config_drum[lane_en_conf]['midi_2'])
        if midi1 > 0:
            if midi1 in self.dict_midi:
                #Esta nota midi ya tiene un lane asociado
                lista = self.dict_midi[midi1]
                lista.append(lane)
                self.dict_midi[midi1] = lista
            else:
                #Ponemos la nota midi e indicamos el lane asociado
                lista = []
                lista.append(lane)
                self.dict_midi[midi1] = lista
        #Ahora lo mismo para la segunda nota posible de este lane
        if midi2 > 0:
            if midi2 in self.dict_midi:
                #Esta nota midi ya tiene un lane asociado
                lista = self.dict_midi[midi2]
                lista.append(lane)
                self.dict_midi[midi2] = lista
            else:
                #Ponemos la nota midi e indicamos el lane asociado
                lista = []
                lista.append(lane)
                self.dict_midi[midi2] = lista

def configurar_midi(self):
    """
        Según la configuracion guardada se crea el dicc MIDI que usaremos
        para comprobar si se pulsa un MIDI a que lane le corresponde
    """
    self.dict_midi = {}
    self.configurar_midi_lane('Lane_11', '11')
    self.configurar_midi_lane('Lane_12', '12')
    self.configurar_midi_lane('Lane_13', '13')
    self.configurar_midi_lane('Lane_14', '14')

```

```

self.configurar_midi_lane('Lane_15', '15')
self.configurar_midi_lane('Lane_16', '16')
self.configurar_midi_lane('Lane_17', '17')
self.configurar_midi_lane('Lane_18', '18')
self.configurar_midi_lane('Lane_19', '19')
self.configurar_midi_lane('Lane_1A', '1A')
self.configurar_midi_lane('Lane_1B', '1B')
self.configurar_midi_lane('Lane_1C', '1C')

def borrar_datos_cancion(self):
    """
        Borramos todos los datos del reproductor
    """
    self.analizando_compas = None
    self.lista_compas = []
    self.dict_bpm = {}
    self.dict_wav = {}
    self.dict_lanes_auto = {}
    self.dict_lanes_visual = {}
    self.bpm = 0
    self.base_bmp = 0
    self.title = ""

def leer_config_dtx(self):
    """
        Leemos el fichero de configuración del dtx
    """
    self.config_dtx = configparser.ConfigParser()
    self.config_dtx.read('config_dtx.ini')

def leer_config_drum(self):
    """
        Leemos el fichero de configuración de las teclas y bateria-MIDI
    """
    self.config_drum = configparser.ConfigParser()
    self.config_drum.read('config_drum.ini')

def crear_lane_visual(self, nombre_lane, lane, color, posicion):
    """
        Creamos un lane visual según datos que nos pasan
        Cuidado si esta en auto según conf también lo tenemos que insertar
        en el diccionario de lanes auto

        nombre_lane: Nombre en la configuración
        lane: Nombre del lane
        color: color que se asociara a lane y por tanto a sus chips
        posicion: posicion en pantalla
    """
    auto = self.config_drum[nombre_lane]['auto']
    lane_creada = LaneVisual(color, posicion, self.tiempo_pantalla)
    self.dict_lanes_visual[lane] = lane_creada
    if auto == 'true':
        self.dict_lanes_auto[lane] = lane_creada

```

```

def reproducir_cancion(self, fichero, tiempo_pantalla):
    """
        Funcion que se llamara para iniciar el reproductor

        fichero: fichero .dtx a reproducir
        tiempo_pantalla: tiempo que tardara un chip en recorrer la pantalla
    """
    #Leer la configuración ya que puede cambiar entre canciones
    self.leer_config_dtx()
    self.leer_config_drum()

    self.scene = QtWidgets.QGraphicsScene(self)
    self.scene.setBackgroundBrush(QtCore.Qt.black)
    #Quitamos 6 píxeles de margen de Qt
    self.scene.setSceneRect(QtCore.QRectF(0, 0, 1264, 714))
    self.setScene(self.scene)
    #En el widget actual se indica la resolución interna, por si se hubiera
    #reescalado en otros widgets
    self.fitInView(QtCore.QRectF(0, 0, 1264, 714))

    self.tiempo_pantalla = tiempo_pantalla

    # CANAL MIDI se ha de seleccionar dependiendo de config
    self.midi_in = rtmidi2.MidiIn()
    midi_port = int(self.config_drum['MIDI']['midi_port'])
    if midi_port < len(rtmidi2.get_in_ports()):
        #Puede estar desconectado al menos comprobar que cuadra números
        self.midi_in.open_port(midi_port)

    #Preparamos el teclado y el instrumento midi para
    self.configurar_teclado()
    self.configurar_midi()

    #Crear los lanes automaticos que siempre estaran
    lane_auto = LaneAuto()
    self.dict_lanes_auto['01'] = lane_auto
    lane_auto = LaneAuto()
    self.dict_lanes_auto['61'] = lane_auto
    lane_auto = LaneAuto()
    self.dict_lanes_auto['62'] = lane_auto

    #Analizar el fichero .dtx para tener todos los datos
    print ("———Analizing .dtx to play———")

    #Crear lanes según configuración
    #pensar en el Hihat dos lanes misma columna y dos platillos derecha
    # 1A 11 1C 12 14 13 15 17 16
    # 18 1B 19

    self.crear_lane_visual('Lane_1A', '1A', QtCore.Qt.darkRed, 10)
    self.crear_lane_visual('Lane_11', '11', QtCore.Qt.blue, 110)
    self.crear_lane_visual('Lane_18', '18', QtCore.Qt.white, 110)

```

```

self.crear_lane_visual('Lane_1C', '1C', QtCore.Qt.gray, 210)
self.crear_lane_visual('Lane_1B', '1B', QtCore.Qt.lightGray, 210)
self.crear_lane_visual('Lane_12', '12', QtCore.Qt.yellow, 310)
self.crear_lane_visual('Lane_14', '14', QtCore.Qt.green, 410)
self.crear_lane_visual('Lane_13', '13', QtCore.Qt.gray, 510)
self.crear_lane_visual('Lane_15', '15', QtCore.Qt.red, 610)
self.crear_lane_visual('Lane_17', '17', QtCore.Qt.darkYellow, 710)
self.crear_lane_visual('Lane_16', '16', QtCore.Qt.blue, 810)
self.crear_lane_visual('Lane_19', '19', QtCore.Qt.darkBlue, 810)

compas_actual = 0
path = os.path.dirname(fichero)

"""
Analizamos el .dtx para obtener la información inicial
Información básica de la canción, títulos
Información de bancos de sonidos
Información de BPM
Lista de compases y sus notas asociadas si tiene
"""
fp = open(fichero, "r")
while True:
    linea = fp.readline()
    if not linea:
        break
    #Limpiamos caracteres raros en la línea
    linea = linea.replace("\t", "")
    linea = linea.rstrip('\r\n')
    linea = linea.split(";")
    campos = linea[0].split()
    primer_espacio = linea[0].find(" ")
    orden = linea[0][:primer_espacio]
    if orden:
        if not (UtilidadesDTX.es_notacion(orden)):
            if orden == "#TITLE:":
                #TITLE
                self.title = linea[0][primer_espacio + 1:]
            elif orden == "#PANEL:":
                #PANEL:
                self.title = linea[0][primer_espacio + 1:]
            elif orden[0:4] == "#BPM":
                if orden[4] == ":":
                    #BPM:
                    self.base_bpm = float(campos[1])
                    self.bpm = float(campos[1])
                else:
                    #BPMzz:
                    canal = str(orden[4:6])
                    self.dict_bpm[canal] = float(campos[1])
            elif orden[0:4] == "#WAV":
                #WAVzz
                canal = str(orden[4:6])
                sonido = linea[0][primer_espacio + 1:]

```



```

        posible_file = os.path.join(path, sonido)
        posible_file = os.path.abspath(posible_file)
        url = QtCore.QUrl.fromLocalFile(posible_file)
        url = url.adjusted(QtCore.QUrl.EncodeSpaces)
        string_url = url.toString()
        if string_url.endswith('.ogg'):
            #ogg
            banco = QMediaPlayer()
            media = QMediaContent(url)
            banco.setMedia(media)
        else:
            #wav
            banco = QSoundEffect()
            banco.setSource(url)
            self.dict_wav[canal] = banco
    elif orden == "#DLEVEL: ":
        orden = ""
        #Orden no tenemos que hacer nada en cuando reproducimos
    else:
        print ("ORDEN_ .dtx_ no implementada_ ", orden)
else:
    #Es una notación de compas, lane, notas
    compas = int(orden[1:-3])
    lane = str(orden[4:-1])
    while compas_actual <= compas:
        if len(self.lista_compas) <= compas:
            #Creamos las lanes para este compas
            compas_vacio = {}
            compas_vacio['lanes'] = {}
            self.lista_compas.append(compas_vacio)
            compas_actual = compas_actual + 1
        if not(lane in self.lista_compas[compas]):
            #Ponemos las notas en su compas y lane
            campo = campos[1].replace('_', '')
            campo = UtilidadesDTX.separar_strings_grupos_2(campo)
            self.lista_compas[compas]['lanes'][lane] = campo
        else:
            print ("TODO_ cuidado_ doble_ linea_ para_ la_ misma_ lane ")
fp.close()

#Creamos la puntuacion
self.puntuacion = MyWidgetPuntuacion(self.tiempo_pantalla, self.title)
self.puntuacion_scene = self.scene.addWidget(self.puntuacion)
self.puntuacion_scene.setPos(900, 80)
style = "QLabel_{background-color:_black;_color:_white;};"
style = style + "Qwidget{background-color:_black;_color:_white;} "
self.puntuacion.setStyleSheet(style)

#Ponemos los bancos de sonidos en los lanes
for key in self.dict_lanes_auto:
    self.dict_lanes_auto[key].insertar_sonidos(self.dict_wav)
for key in self.dict_lanes_visual:
    self.dict_lanes_visual[key].insertar_sonidos(self.dict_wav)

```

```

#calculamos tiempo de compases en ms
#Para ello recorreremos toda la lista de compases y tendremos que ir
#calculando el tiempo de ese compas, cuidado con los cambios de tiempo
tiempo = 0.0
for compas in self.lista_compas:
    compas['start'] = tiempo
    lanes = compas['lanes']
    if lanes.get('03'):
        #LANE 03 es un cambio de bpm pero usando sumas y restas
        #Cambio de bpm para el compas actual
        if len(lanes['03']) == 1:
            self.bpm = self.base_bpm + int(lanes['03'][0], 16)
            compas['bpm'] = self.bpm
            tiempo_compas = (60.0 * 4.0 / self.bpm) * 1000
            tiempo_final = tiempo + tiempo_compas
        else:
            #Supongo que es un cambio para el compas siguiente
            compas['bpm'] = self.bpm
            tiempo_compas = (60.0 * 4.0 / self.bpm) * 1000
            tiempo_final = tiempo + tiempo_compas
            self.bpm = self.base_bpm + int(lanes['03'][0], 16)
    elif lanes.get('08'):
        #LANE 08 es un cambio de bpm pero usando bpm totales
        if len(lanes['08']) == 1:
            #Cambio para el compas actual
            self.bpm = self.dict_bpm[lanes['08'][0]]
            compas['bpm'] = self.bpm
            tiempo_compas = (60.0 * 4.0 / self.bpm) * 1000
            tiempo_final = tiempo + tiempo_compas
        else:
            #Supongo que es un cambio para el compas siguiente
            compas['bpm'] = self.bpm
            tiempo_compas = (60.0 * 4.0 / self.bpm) * 1000
            tiempo_final = tiempo + tiempo_compas
            self.bpm = self.dict_bpm[lanes['08'][-1]]
    else:
        #No hay cambio de bpm
        compas['bpm'] = self.bpm
        tiempo_compas = (60.0 * 4.0 / self.bpm) * 1000
        tiempo_final = tiempo + tiempo_compas
    tiempo = tiempo_final
    compas['tiempo'] = tiempo_compas

#Ahora que tenemos el tiempo y cambios de bpm podemos poner el
#tiempo en las notas y quitando las notas no existentes '00'
for compas in self.lista_compas:
    tiempo = compas['tiempo']
    lanes = compas['lanes']
    if lanes:
        for lane in lanes:
            notas_tiempo = []
            num_notas = len(lanes[lane])

```

```

        inc_tiempo = tiempo / num_notas
        for i in range(0, num_notas):
            #Creamos las notas con su tiempo asociado
            nota = lanes[lane][i]
            if nota != '00':
                nota_tiempo = [None, None]
                nota_tiempo[0] = nota
                #Tiempo Absoluto
                tiempo_nota = compas['start'] + (inc_tiempo * i)
                nota_tiempo[1] = tiempo_nota
                notas_tiempo.append(nota_tiempo)
            lanes[lane] = notas_tiempo

#Ya tenemos todos los compases con su tiempo
self.tiempo_final = tiempo_final + self.tiempo_pantalla

#Tiempo final preparamos para volver al menu
self.timer_salida = QtCore.QTimer()
self.timer_salida.setSingleShot(True)
self.timer_salida.timeout.connect(self.salir)

#Timer para bucle principal
self.timer_bucle_cancion = QtCore.QTimer()
self.timer_bucle_cancion.setSingleShot(True)
self.timer_bucle_cancion.timeout.connect(self.animar_cancion)

#Definimos la funcion del MIDI
def midi_handler(message, time_stamp):
    """
        Se dispara cuando hay un evento MIDI

        message: el evento MIDI en si
        time_stamp: tiempo diferido hasta que se ha llamado esta función
    """
    if message[0] == 153:
        #Nota On
        if message[1] in self.dict_midi:
            tiempo_actual = float(self.timer.elapsed())
            if len(self.dict_midi[message[1]]) == 1:
                #Solo hay que mirar un lane
                #Mas optimizado miramos 1 vez y lo ponemos en hit
                lane_selec = self.dict_midi[message[1]][0]
                lane = self.dict_lanes_visual[lane_selec]
                tiempo = lane.comprobar_y_poner_hit(tiempo_actual)
                self.puntuacion.comprobar_puntuacion(tiempo)
            else:
                #Hay que mirar varios lanes
                tiempo_menor = 10000
                lista_lanes_menores = []
                for lane_mirar in self.dict_midi[message[1]]:
                    #Miramos todos los lanes posibles sin poner a hit
                    lane = self.dict_lanes_visual[lane_mirar]
                    tiempo = lane.comprobar_tiempo(tiempo_actual)

```

```

        if tiempo < tiempo_menor:
            #Nuevo tiempo menor
            tiempo_menor = tiempo
            lista_lanes_menores.append(lane)
        elif tiempo == tiempo_menor:
            #Cuidado que podemos tener varios lanes
            lista_lanes_menores.append(lane)
        #Ponemos a hit
        for lane in lista_lanes_menores:
            tiempo = lane.comprobar_y_poner_hit(tiempo_actual)
            self.puntuacion.comprobar_puntuacion(tiempo)

#Juntamos el midi a la funcion de entrada
self.midi_in.callback = midi_handler

print ("Starting the song")

#Iniciamos el timer inicial para tener control del tiempo que llevamos
self.timer = QtCore.QElapsedTimer()
self.timer.start()
#Iniciamos timer de bucle principal y el de la salida
self.timer_salida.start(self.tiempo_final + 2* self.tiempo_pantalla)
self.timer_bucle_cancion.start(1)

#Se anima el primer compas para despues ir al bucle principal
self.analizando_compas = self.lista_compas[0]
self.lista_compas.pop(0)

def animar_cancion(self):
    """
        Hay que mirar, siempre depediendo del tiempo:
        insertar chips en su correspondiente lane
        si hay que pasar al siguiente compas
        reproducir sonidos en las lanes auto
        quitar chips de lane por mucho tiempo pasado

    """
    tiempo_actual = float(self.timer.elapsed())
    if self.analizando_compas:
        #Miramos si hay chips para animar
        self.animar_compas(self.analizando_compas, tiempo_actual)

        tiempo_siguiente_compas = self.analizando_compas['tiempo'] + \
            self.analizando_compas['start']
        if tiempo_actual >= tiempo_siguiente_compas:
            #pasamos a analizar el siguiente compas
            if self.lista_compas:
                self.analizando_compas = self.lista_compas[0]
                self.lista_compas.pop(0)

        #Mirar canales auto que suenen si tienen que sonar y quitamos notas
        for key in self.dict_lanes_auto:
            self.dict_lanes_auto[key].reproducir(tiempo_actual)

```

```

        #Borrar las notas antiguas de los lanes visuales
        for key in self.dict_lanes_visual:
            lane = self.dict_lanes_visual[key]
            estado = lane.eliminar_nota_antigua(tiempo_actual)
            if estado:
                self.puntuacion.aumentar_miss()
self.timer_bucle_cancion.start(0.1)

def borrar_nota_analizada_lane(self, lane):
    """
        Quitamos del compas que se esta analizando la primera nota del lane

        lane: lane donde se quitara la primera nota
    """
    lista_compas = self.analizando_compas['lanes'][lane]
    lista_compas.pop(0)
    self.analizando_compas['lanes'][lane] = lista_compas

def animar_chip(self, chip):
    """
        Tenemos que insertar el chip a la scene para que se pueda pintar

        chip: chip que se insertara en la scene
    """
    self.scene.addItem(chip)

def animar_chip_lane(self, nota, lane):
    """
        Insertamos la nota en un lane visual y
        le sumamos el tiempo de pantalla al tiempo de la nota
        creamos el chip dependiendo del lane donde tiene que ir

        nota: nota que se tiene que insertar al lane
        lane: lane
    """
    tiempo_final_nota = nota[1] + self.tiempo_pantalla
    chip = Chip(nota[0], tiempo_final_nota)
    color = self.dict_lanes_visual[lane].leer_color()
    posicion = self.dict_lanes_visual[lane].leer_pos()
    chip.set_color(color)
    chip.animar(posicion, self.tiempo_pantalla)
    self.dict_lanes_visual[lane].insertar_chip(chip)
    self.animar_chip(chip)

def animar_compas(self, compas_actual, tiempo_actual):
    """
        Comprobamos el compas_actual, y dependiendo del tiempo se ira
        animando los chips a sus lanes correspondientes

        compas_actual: compas actual
        tiempo_actual: tiempo actual
    """

```

```

#Analizamos el compas y pasamos los chips a sus lanes correspondientes
for lane in compas_actual['lanes']:
    if compas_actual['lanes'][lane]:
        #Existe lista de notas
        if compas_actual['lanes'][lane][0]:
            #Hay una primera nota
            nota = compas_actual['lanes'][lane][0]
            tiempo_final_nota = nota[1] + self.tiempo_pantalla
            #Lanes Auto
            if lane == '01' or lane == '61' or lane == '62':
                if tiempo_actual >= nota[1]:
                    chipAuto = ChipAuto(nota[0], tiempo_final_nota)
                    self.dict_lanes_auto[lane].insertar_chip(chipAuto)
                    self.borrar_nota_analizada_lane(lane)
            #Lanes Visuales
            elif lane == '11' or lane == '12' or lane == '13' or \
                 lane == '14' or lane == '15' or lane == '16' or \
                 lane == '17' or lane == '18' or lane == '19' or \
                 lane == '1A' or lane == '1B' or lane == '1C':
                if tiempo_actual >= nota[1]:
                    self.animar_chip_lane(nota, lane)
                    self.borrar_nota_analizada_lane(lane)
            else:
                #Comprobamos que no sea lan de control de tiempo
                #antes de warning
                if lane != '03' and lane != '08':
                    print("LANE_NO_IMPLMENTADO:", lane)

def salir(self):
    """
        Volvemos al menu principal, para ello apuntamos en el stackedwidget
        al widget de navegación y desactivamos el MIDI y la canción
    """
    self.borrar_datos_cancion()
    self.parentWidget().setCurrentIndex(1)
    self.parentWidget().update()
    self.parentWidget().repaint()
    self.midi_in = None

```

Apéndice B

Guía de estilo python

Aquí esta la guía de estilo que se utilizara durante el proyecto para escribir código, ya que de esta manera tanto la lectura, como el mantenimiento del código sera mucho más sencillo. Uno de los grandes problemas de python es que no se esta obligado a seguir un estilo de programación, a la hora de escribir código se puede generar verdaderos desastres y si encima se junta que se puede crear variables en tiempo de ejecución y en mitad del código, mejor no pensar en las barbaridades con las que uno se puede encontrar.

Para ello se va a exponer las normas que se usaran para generar código python.

Una de las ideas es que el código se suele leer mucho más de lo que se escribe. Las guías de estilo que proporciona este documento están dirigidas a mejorar la legibilidad del código y hacerlo consistente a lo largo del amplio espectro del código python. Como dice el Zen de python, “La legibilidad cuenta”. Una guía de estilo nos ayuda a lograr consistencia. Ser consistente con esta guía de estilo es importante. La consistencia dentro de un proyecto es aún más importante. La consistencia dentro de un módulo o función es la más importante.

Dos buenas razones para romper una regla en particular son:

1. Que al aplicar la regla el código se haga menos legible, incluso para alguien que esté acostumbrado a leer código que sigue las normas.
2. Para ser consistente con código relacionado que también la rompe (quizás por razones históricas) – aunque esto también es una oportunidad de arreglar el desaguisado de otra persona (al más puro estilo XP).

B.1. Formateo del código

B.1.1. Indentación

Usa 4 espacios por cada nivel de indentación

B.1.2. Tamaño máximo de línea

Limita todas las líneas a un máximo de 79 caracteres.

B.1.3. Líneas en blanco

Separa las funciones no anidadas y las definiciones de clases con dos líneas en blanco.

Las definiciones de métodos dentro de una misma clase se separan con una línea en blanco.

Se pueden usar líneas en blanco extra (de forma reservada) para separar grupos de funciones relacionadas. Las líneas en blanco se pueden omitir entre un grupo de funciones con una sola línea (por ejemplo, con un conjunto de funciones sin implementación).

Usa líneas en blanco en las funciones, de forma limitada, para indicar secciones lógicas.

B.1.4. Codificación de caracteres

Usaremos UTF-8

B.1.5. Imports

- Normalmente los imports deberían colocarse en distintas líneas, por ejemplo: Sí:

```
import os
import sys
```

No:

```
import sys, os2
```

aunque también es correcto hacer esto:


```
from subprocess import Popen, PIPE
```

- Los imports se colocan siempre en la parte superior del archivo, justo después de cualquier comentario o cadena de documentación del módulo, y antes de las variables globales y las constantes del módulo.

Los imports deberían agruparse siguiendo el siguiente orden:

- 1. imports de la librería estándar
- 2. imports de proyectos de terceras partes relacionados
- 3. imports de aplicaciones locales/imports específicos de la librería

Deberías añadir una línea en blanco después de cada grupo de imports.

Si es necesario especificar los nombres públicos definidos por el módulo con `__all__` esto debería hacerse después de los imports.

- Es muy desaconsejable el uso de imports relativos para importar código de un paquete. Utiliza siempre la ruta absoluta del paquete para todos los imports
- Cuando importes una clase de un módulo, normalmente es correcto hacer esto

```
from myclass import MyClass
from foo.bar.yourclass import YourClass
```

Si esto causa colisiones de nombres con objetos locales, utiliza en su lugar

```
import myclass
import foo.bar.yourclass
```

y usa “myclass.MyClass” y “foo.bar.yourclass.YourClass” en el código

B.1.6. Espacios en blanco en expresiones y sentencias

Evita espacios en blanco extra en las siguientes situaciones:

- Inmediatamente después de entrar en un paréntesis o antes de salir de un paréntesis, corchete o llave.

Sí:

```
spam(ham[1], {eggs: 2})
```

No:

```
spam( ham[ 1 ], { eggs: 2 } )
```

- Inmediatamente antes de una coma, punto y coma, o dos puntos:

Sí:

```
if x == 4: print x, y; x, y = y, x
```

No:

```
if x == 4 : print x , y ; x , y = y , x
```

- Inmediatamente antes de abrir un paréntesis para una lista de argumentos de una llamada a una función:

Sí:

```
spam(1)
```

No:

```
spam (1)
```

- Inmediatamente antes de abrir un paréntesis usado como índice o para particionar (*slicing*):

Sí:

```
dict['key'] = list[index]
```

No:

```
dict [ 'key' ] = list [index]
```

- Más de un espacio alrededor de un operador de asignación (u otro operador) para alinearlos con otro.

Sí:

```
x = 1
y = 2
long_variable = 3
```

No:

```
x           = 1
y           = 2
long_variable = 3
```

B.1.7. Otras Recomendaciones

- Rodea siempre los siguientes operadores binarios con un espacio en cada lado: asignación (=), asignación aumentada (+=, -= etc.), comparación (==, <, >, !=, <>, <=, >=, in, not in, is, is not), booleanos (and, or, not).

- Usa espacios alrededor de los operadores aritméticos:

Sí:

```
i = i + 1
submitted += 1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)
```

No:

```
i=i+1
submitted +=1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

- No uses espacios alrededor del signo '=' cuando se use para indicar el nombre de un argumento o el valor de un parámetro por defecto.

Sí:

```
def complex(real, imag=0.0):
    return magic(r=real, i=imag)
```

No:

```
def complex(real, imag = 0.0):
    return magic(r = real, i = imag)
```

- Generalmente se desaconsejan las sentencias compuestas (varias sentencias en la misma línea).

Sí:

```
if foo == 'blah':
    do_blah_thing()
do_one()
do_two()
do_three()
```

Preferiblemente no:

```
if foo == 'blah': do_blah_thing()
do_one(); do_two(); do_three()
```

- Aunque a veces es adecuado colocar un if/for/while con un cuerpo pequeño en la misma línea, nunca lo hagas para sentencias multi-clausula.

Preferiblemente no:

```
if foo == 'blah': do_blah_thing()
for x in lst: total += x
while t < 10: t = delay()
```

Definitivamente no:

```
if foo == 'blah': do_blah_thing()
else: do_non_blah_thing()

try: something()
finally: cleanup()

do_one(); do_two(); do_three(long, argument,
                             list, like, this)

if foo == 'blah': one(); two(); three()
```

B.2. Comentarios

Los comentarios que contradicen el código son peores que no tener ningún comentario. ¡Manten siempre como prioridad el mantener los comentarios actualizados cuando cambies el código!

Los comentarios deberían ser frases completas. Si un comentario es una frase o sentencia, la primera palabra debería estar en mayúsculas, a menos que sea un identificador que comience con una letra en minúsculas (¡nunca alteres el identificador sustituyendo mayúsculas o minúsculas!).

Si un comentario es corto, se puede omitir el punto al final. Los comentarios de bloque generalmente consisten en uno o más párrafos contruidos con frases completas, y cada frase debería terminar con un punto.

B.2.1. Comentarios de bloque

Los comentarios de bloque generalmente se aplican al código que se encuentra a continuación, y se indentan al mismo nivel que dicho código. Cada línea de un comentario de bloque comienza con un `#` y un único espacio (a menos que haya texto indentado dentro del comentario).

Los párrafos dentro de un comentario de bloque se separan por una línea conteniendo un único `#`.

B.2.2. Comentarios en línea

Utiliza comentarios en línea de forma moderada.

Un comentario en línea es un comentario que se encuentra en la misma línea que una sentencia. Los comentarios en línea deberían separarse por al menos dos espacios de la sentencia que comentan.

Deberían comenzar con un `#` y un espacio.

Los comentarios en línea son innecesarios y de hecho sólo sirven para distraer si indican cosas obvias. No hagas cosas como esta:

```
x = x + 1                # Incrementa x
```

Aunque a veces, algo como esto puede ser de utilidad:

```
x = x + 1                # Compensa por el borde
```

B.3. Cadenas de Documentación

- Escribe docstrings para todos los módulos, funciones, clases, y métodos públicos. Los docstrings no son necesarios para métodos no públicos, pero deberías tener un comentario que describa lo que hace el método. Este comentario debería aparecer antes de la línea `"def"`.
- Es importante notar, que el `"""` que finaliza un docstring de varias líneas debería situarse en una línea separada, y preferiblemente precederse por una línea en blanco, por ejemplo:

```
"""Return a foobang

Optional plotz says to frobnicate the bizbaz first.
"""
```

- Para cadenas de documentación de una línea, es adecuado mantener el `"""` de cierre en la misma línea.

B.4. Convenciones de nombres

Las convenciones de nombres de la librería de python son un pequeño desastre, así que nunca conseguiremos que sea completamente consistente – aún así, aquí tenéis los estándares de nombrado recomendados en la actualidad. Los nuevos módulos y paquetes (incluyendo los frameworks de terceras personas) deberían acomodarse a estos estándares, pero donde exista una librería con un estilo distinto, se prefiere la consistencia interna.

B.4.1. Descriptivo: Estilos de Nombrado

Hay un montón de estilos de nombrado distintos. Ayuda el poder reconocer qué estilo de nombrado se utiliza, independientemente de para lo que se utilice.

Podemos encontrarnos con los siguientes estilos de nombrado más comunes:

- b (una sola letra en minúsculas)
- B (una sola letra en mayúsculas)
- minúsculas
- minúsculas_con_guiones_bajos
- MAYÚSCULAS
- MAYÚSCULAS_CON_GUIONES_BAJOS
- PalabrasEnMayusculas (CapWords o CamelCase). Algunas veces también se llaman Studly-Caps. Nota: Cuando se usan abreviaturas en CapWords, mantén en mayúsculas todas las letras de la abreviatura. Por lo tanto `HTTPServerError` es mejor que `HttpServerError`.
- Mayusculas_Con_Guiones_Bajos

También existe un estilo en el cuál se utiliza un prefijo único corto para agrupar nombres relacionados de forma conjunta. Esto no se suele utilizar mucho en Python, pero se menciona con el objeto de que el texto sea lo más completo posible. Por ejemplo, la función `os.stat()` devuelve una tupla cuyos elementos tradicionalmente han tenido nombres como `st_mode`, `st_size`, `st_mtime` y similares. (Esto se hace para enfatizar la correspondencia con los campos de la estructura de la llamada a sistema POSIX, lo cual es de utilidad para los programadores que están familiarizados con ella.)

La librería X11 utiliza una X inicial para todas sus funciones públicas. En Python, este estilo generalmente se considera innecesario porque los atributos y métodos se preceden por el objeto al que pertenecen, y los nombres de funciones se preceden del nombre del módulo.

Además se reconocen las siguientes formas especiales usando guiones bajos al inicio o final del nombre (generalmente estos se pueden combinar con cualquier convención de capitalización):

- `_guion_bajo_al_inicio` : indicador débil de “uso interno”. Por ejemplo “`from M import *`” no importa objetos cuyos nombres comiencen con un guión bajo.
- `guion_bajo_al_final_` : usado por convención para evitar conflictos con palabras clave de Python, por ejemplo


```
Tkinter.Toplevel(master, class_='ClassName')
```
- `__doble_guion_bajo_al_inicio` : cuando se use para nombrar un atributo de clase, invoca la característica de “planchado de nombres”(dentro de la clase `FooBar`, `__boo` se convierte en `__FooBar__boo`; ver abajo).

- `__doble_guion_bajo_al_inicio_y_fin__` : objetos "mágicos." atributos que viven en espacios de nombres controlados por el usuario. Por ejemplo `__init__`, `__import__` o `__file__`. Nunca inventes nombres como estos; utilízalos sólo como están documentados.

B.4.2. Prescriptivo: Convenciones de Nombrado

Nombres a Evitar

Nunca utilices los caracteres “l” (letra ele minúscula), “O” (letra o mayúscula), o “I” (letra i mayúscula) como nombres de variables de un solo carácter.

En algunas fuentes, estos caracteres son indistinguibles de los numerales uno y cero. Cuando estés tentado de usar “l”, utiliza una “L” en su lugar.

Nombres de Paquetes y Módulos

Los módulos deberían tener nombres cortos formados en su totalidad por letras minúsculas. Se puede utilizar guiones bajos en el nombre del módulo si mejora la legibilidad. Los paquetes Python también deberían tener nombres cortos formados de letras minúsculas, aunque se desaconseja el uso de guiones bajos.

Dado que los nombres de los módulos se mapean a nombres de archivos, y algunos sistemas de ficheros no diferencian entre mayúsculas y minúsculas y truncan los nombres largos, es importante que los nombres de los módulos que se elijan sean suficientemente cortos – esto no es un problema en Unix, pero puede ser un problema cuando el código se porta a versiones antiguas de Mac o Windows, o a DOS.

Cuando un módulo de extensión escrito en C o C++ tenga un módulo Python asociado que proporcione una interfaz de más alto nivel (por ejemplo, más orientado a objetos), el módulo C/C++ debería comenzar con un guión bajo (por ejemplo `_socket`).

Nombres de Clases

Casi sin excepciones, los nombres de clases usan la convención CapWords. Las clases de uso interno tienen además un guión bajo al principio del nombre.

Nombres de Excepciones

Dado que las excepciones deberían ser clases, se aplica la convención relativa al nombrado de clases. De todas formas, deberías usar el sufijo “Error” en los nombres de las excepciones (si la excepción es realmente un error).

Nombres de Variables Globales

(Esperamos que estas variables estén destinadas a ser usadas sólo dentro de un módulo.) Las convenciones son prácticamente las mismas que para las funciones.

Los módulos diseñados para ser utilizados usando “from M import *” deberían usar el mecanismo `__all__` para prevenir que se exporten variables globales, o bien usar la convención antigua de añadir un guión bajo como prefijo a dichas variables globales (puede que quieras hacer esto para indicar que estas variables son “variables no públicas del módulo”).

Nombres de Funciones

Los nombres de funciones deberían estar en letras minúsculas, con palabras separadas mediante guiones bajos según sea necesario para mejorar la legibilidad.

Sólo se acepta capitalizaciónMezclada en contextos en los que ya se trate del estilo principal (por ejemplo `threading.py`), para mantener la compatibilidad hacia atrás.

Argumentos de funciones y métodos

Utiliza siempre `'self'` como primer argumento de los métodos de instancia.

Utiliza siempre `'cls'` como primer argumento de métodos de clase.

Si el nombre de un argumento de una función colisiona con una palabra reservada, generalmente es mejor añadir un guión bajo al final del nombre en lugar de usar una abreviatura o modificar la grafía. Por lo tanto “`print_`” se considera mejor que “`prnt`”. (Quizás utilizar sinónimos sea una forma incluso mejor.)

Nombres de métodos y variables de instancia

Utiliza las reglas de los nombres de funciones: minúsculas con palabras separadas por guiones bajos cuando sea necesario para mejorar la legibilidad.

Utiliza guiones bajos al inicio sólo para métodos no públicos y variables de instancia.

Para evitar colisiones de nombres con subclases, utiliza dos guiones bajos al principio del nombre para invocar las reglas de planchado de nombres de Python.

Python une estos nombres con el nombre de la clase: si la clase `Foo` tiene un atributo llamado `__a`, no se puede acceder utilizando `Foo.__a`. (Un usuario insistente podría acceder utilizando `Foo._Foo__a`.) Generalmente, sólo se debería usar un doble guión al inicio para evitar conflictos de nombres con atributos en clases diseñadas para que se herede de ellas.

Nota: existe controversia sobre el uso de `__` nombres (ver debajo)

Diseñar para la herencia

Decide siempre si los métodos y variables de instancia de una clase (llamados de forma colectiva “atributos”) deberían ser públicos o no públicos. Si dudas, elige que sea no público; es más sencillo convertirla en público más tarde que hacer no público un atributo que antes era público.

Los atributos públicos son aquellos que esperas que sean utilizados por los clientes de tu clase, y para los cuales te comprometes a evitar cambios que provoquen incompatibilidades hacia atrás. Los atributos no públicos son aquellos que no están destinados a que sean utilizados por terceras personas; no ofreces ninguna garantía de que los atributos no públicos no vayan a cambiar o a eliminarse.

No utilizamos el término “privado” aquí, dado que ningún atributo es realmente privado en Python (sin un trabajo añadido generalmente innecesario).

Otra categoría de atributos son aquellos que son parte de la “API de las subclases” (a menudo llamado “protected” en otros lenguajes). Algunas clases se diseñan para que se herede de ellas, bien para extender o bien para modificar aspectos del comportamiento de la clase. Cuando se diseña una clase de esta forma, es necesario tomar algunas decisiones explícitas sobre qué atributos van a ser públicos, cuáles son parte de la API de la subclase, y cuáles están pensados para ser utilizados exclusivamente dentro de la clase actual.

Teniendo esto en cuenta, aquí están las líneas a seguir:

- Los atributos públicos no deberían tener guiones bajos al inicio del nombre.
- Si el nombre de tu atributo público colisiona con el de una palabra reservada, añade un único guión bajo al final del nombre del atributo. Esto es preferible a abreviar o cambiar la grafía de la palabra.

Nota 1: Ver la recomendación anterior para métodos de clase.

- Para campos públicos simples, es mejor exponer sólo el nombre del atributo, sin complicaciones de métodos para accederlos y modificarlos (accessors y mutators). Ten en cuenta que Python proporciona una forma sencilla de modificarlo, en caso de que dentro de un tiempo lo necesites. En ese caso, utiliza propiedades para ocultar la implementación funcional con una sintaxis simple de acceso a atributos de datos.

Nota 1: Las propiedades sólo funcionan en las nuevas clases.

Nota 2: Intenta mantener el comportamiento funcional libre de efectos colaterales, aunque funcionalidad colateral como el cacheo generalmente es aceptable.

Nota 3: Evita usar propiedades para realizar operaciones que requieran de grandes cálculos; el utilizar la notación de atributos hace que la persona que accede al atributo piense que el acceso es (relativamente) barato en tiempo computacional.

B.5. Recomendaciones para la programación

- El código debería escribirse de forma que no pusiera en desventaja otras implementaciones de Python (PyPy, Jython, IronPython, Pyrex, Psyco, y similares).

Por ejemplo, no te apoyes en la eficiencia de la implementación de la concatenación de cadenas con `a+=b` o `a=a+b` en CPython. Estas órdenes se ejecutan más lentamente en Jython. En partes de la librería en las que el rendimiento sea importante, se debería utilizar en su lugar la forma `".join()`. Esto asegura que el tiempo necesario para la concatenación es del mismo orden en diferentes implementaciones.

- Las comparaciones con *singletons* como `None` siempre deberían llevarse a cabo con `'is'` o `'is not'`, nunca con operadores de igualdad.

También es importante tener cuidado con escribir “if x” cuando lo que realmente queríamos decir es “if x is not None” – por ejemplo, cuando comprobemos si a una variable o argumento que tiene como valor por defecto `None` se le ha dado otro valor. ¡El otro valor podría tener un tipo (como un contenedor) que podría ser falso en un contexto booleano!

- Utiliza excepciones basadas en clases.

Los módulos o paquetes deberían definir sus propias clases excepción específicas para el dominio del problema, las cuales deberían heredar de la clase `Exception`. Incluye siempre una cadena de documentación para la clase. Por ejemplo:

```
class MessageError(Exception):
    """Base class for errors in the email package."""
```

En este caso se aplican las convenciones de nombrado de las clases, aunque deberías añadir el sufijo “Error” a las clases excepción, si la excepción se trata de un error. Las excepciones que no sean errores, no necesitan ningún sufijo especial.

- Cuando lances una excepción, utiliza “raise ValueError(‘mensaje’)” en lugar de la forma antigua “raise ValueError, ‘mensaje’”.

Se prefiere la forma con paréntesis porque cuando los argumentos de la excepción son largos o incluyen formateo de cadenas, no necesitas usar caracteres para indicar que continúa en la siguiente línea gracias a los paréntesis. La forma antigua se eliminará en Python 3000.

- Cuando captures excepciones, menciona excepciones específicas cuando sea posible en lugar de utilizar una cláusula ‘except:’ genérica.

Por ejemplo, utiliza:

```
try:
    import platform_specific_module
except ImportError:
    platform_specific_module = None
```

Una cláusula ‘except:’ genérica capturaría las excepciones SystemExit y KeyboardInterrupt, complicando el poder interrumpir el programa usando Control-C, y pudiendo ocultar otros problemas. Si quieres capturar todas las excepciones relativas a errores en el programa, utiliza ‘except Exception:’.

Una buena regla a seguir es limitar el uso de las cláusulas ‘except’ genéricas a dos casos:

- Si el manejador de la excepción imprimirá o registrará el traceback; de esta forma al menos el usuario sabrá que ha ocurrido un error.
 - Si el código tiene que realizar algún trabajo de limpieza, pero en ese caso tenemos que hacer que la excepción se propague hacia arriba usando ‘raise’. La construcción ‘try...finally’ es una mejor forma de tratar con este caso.
- Adicionalmente, para todas las cláusulas try/except, es necesario limitar la cláusula ‘try’ a la mínima expresión de código necesaria. De nuevo, esto se hace para evitar ocultar bugs.

Sí:

```
try:
    value = collection[key]
except KeyError:
    return key_not_found(key)
else:
    return handle_value(value)
```

No:

```

try:
    # ¡Demasiado amplio!
    return handle_value(collection[key])
except KeyError:
    # También capturará la excepción KeyError lanzada por handle_value()
    return key_not_found(key)

```

- Utiliza métodos de cadenas en lugar del módulo string.

Los métodos de las cadenas son siempre mucho más rápidos y comparten la misma API con las cadenas unicode. Ignora esta regla si es necesaria compatibilidad hacia atrás con versiones de Python anteriores a la 2.0.

- Utiliza ".startswith()" y ".endswith()" en lugar del particionado de cadenas para comprobar prefijos y sufijos.

startswith() y endswith() son más limpios y menos propensos a errores. Por ejemplo:

Sí:

```
if foo.startswith('bar'):
```

No:

```
if foo[:3] == 'bar':
```

- Las comparaciones del tipo de objeto siempre deberían utilizar isinstance() en lugar de comparar tipos directamente.

Sí:

```
if isinstance(obj, int):
```

No:

```
if type(obj) is type(1):
```

- Para secuencias, (cadenas, listas, tuplas), aprovecha el que las secuencias vacías son equivalentes al falso booleano.

Sí:

```
if not seq:
if seq:
```

No:

```
if len(seq)
if not len(seq)
```

- No compares valores booleanos con True o False usando ==

Sí:

```
if greeting:
```

No:

```
if greeting == True:
```

Peor:

```
if greeting is True:
```

Apéndice C

GNU GENERAL PUBLIC LICENSE

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

<http://www.gnu.org/licenses/gpl-3.0.html>

Glosario

Chips Nota visual que va bajando por los lanes para indicarnos cuando se ha de tocar.

Guitar Hero Juego lanzado en 2005, por Harmonix que se basa con el controlador en forma de guitarra simula a una guitarra verdadera.

Lanes Columnas donde van los chips visuales del juego musical.

Pads Parte de una batería-MIDI donde se ha de golpear para que imite el sonido de ese instrumento, normalmente cada pad solo emite una nota, variando su potencia dependiendo del golpe, la excepción suele ser el charles que tiene posición abierto y cerrado, en las baterías profesionales hay pads que tienen varias zonas de golpeo y emiten un sonido dependiendo de la zona y la potencia.

Rock Band Juego lanzado en 2007, que se basa en simular una banda de rock, con sus diferentes partes vocales, guitarra y bajo, batería y piano.

Apéndice D

Bibliografía

Bibliografía

- [1] The Not So Short Introduction to LATEX: tobi.oetiker.ch/lshort/lshort.pdf
- [2] Wikipedia: www.wikipedia.es
- [3] Guia estilo python : <http://mundogeek.net/traducciones/guia-estilo-python.htm>
- [4] Proyecto DTXMania: <https://osdn.jp/projects/dtxmania/>
- [5] Pineado Drum HAT : https://pinout.xyz/pinout/drum_hat