10

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE MÚSICA

GAB: Sistema de reinterpretación para pianos

Opción de tesis:

Tesina y examen de conocimientos generales

Para obtener el Título de

Licenciado en Piano

29999

Presenta:

Hugo Solís García

México D. F. Noviembre de 2001

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

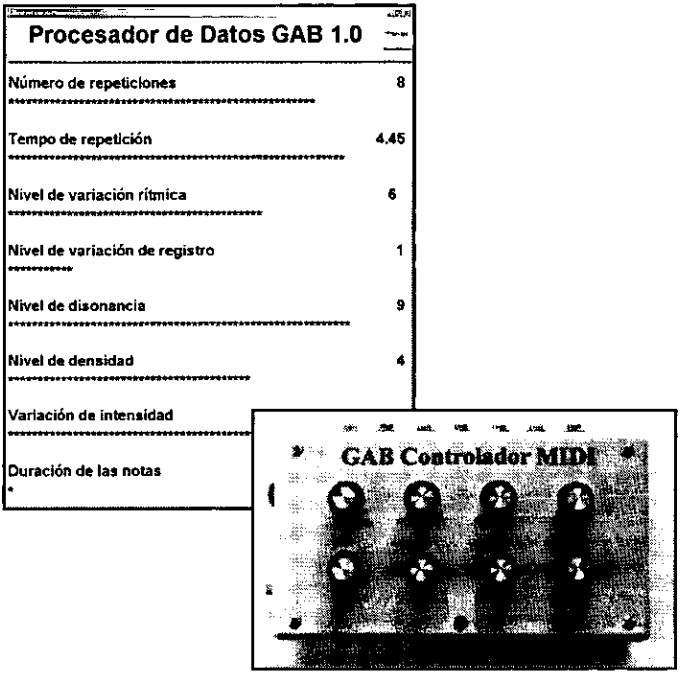
UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

GAB: Sistema de reinterpretación para pianos



299991

Hugo Solís García

Agradecimientos:

A mis amigos Isa, Jiu, Ixchel, Mauricio y Julieta por ser una familia ultra-cachete-piocha.

A Emilio, Omar, Mauricio, Rodrigo, Ulianov, Hector y Carlos por ser los que son.

A Suad, Ana, Vanessa y Mónica. Miles de millones de besos.

Al Laboratorio de Creación Musical por ser un lugar para imaginar.

A mis maestros más chidos: Alejandro Escuer, Julio Estrada y Salvador Rodríguez.

A mis maestros Nick Didkovsky y James Carpino de NYU.

A mis maestros de piano: Ario Garza, Eva del Carmen Medina, Andrés Acosta, Tere Frenk y Maurcio Náder.

A toda la comunidad de la Escuela Nacional de Música: estudiantes, trabajadores y académicos. Goya!

A mi tío Germán, a mi prima Pao y a mis abuelas.

A los que trabajan por un futuro justo y digno para todos.

A GABy por aquella época tan mágica.

ÍNDICE

1 Introducción

En 1997 el inventor de instrumentos musicales Trimpin realizó una serie de pláticas en la Escuela Nacional de Música dentro de las Jornadas Conlon Nancarrow. Estas pláticas estuvieron acompañadas de algunas demostraciones de un equipo electrónico capaz de "tocar" cualquier piano acústico por medio de pequeños selenoides que eran colocados sobre las teclas del piano y controlados por medio de una computadora. Este dispositivo había sido diseñado y construido en su totalidad por Trimpin quien es músico de formación pero posee sólidos conocimientos en el área de la ingeniería, la electrónica y la computación. Su particular visión de la música y sus conocimientos en otras áreas le han permitido crear un lenguaje musical propio y original.

Las pláticas de Trimpin fueron un evento significativo en mis estudios de licenciatura. Por primera vez me percataba de que mi inquietud por la electrónica y la ingeniería no estaba divorciada de mis estudios de piano y que, incluso, podía ser aprovechada para generar expresiones musicales personales. ¿Si la pianola automática de Trimpin era capaz de tocar con una precisión dinámica, con velocidades y con densidades imposibles para un ser humano, no sería posible llevar estas posibilidades sonoras a la ejecución en vivo? Fue en este momento cuando surgió la idea de diseñar un equipo electrónico que posibilitara a un pianista controlar diferentes procesos musicales desde su instrumento.

Mis estudios finales dentro del Laboratorio de Creación Musical; mi participación dentro del proyecto Innovación Tecnológica para la Composición Musical; una estancia en la Universidad de New York en donde tuve la oportunidad de tomar cursos de programación; y algunas pláticas con el matemático

Miller Puket -diseñador de los programas MAX y Pure Data-, fueron las actividades que dieron forma y solidez a la idea de crear un equipo electrónico que posibilitara a un pianista controlar diferentes variables acústicas durante la ejecución de improvisaciones musicales. Así surgió el sistema GAB, un sistema diseñado con la expresa intención de permitirle a un pianista manipular en tiempo real diferentes variables musicales como son la densidad, la velocidad en la repetición de las notas, el rango dinámico, etc.

En esta primera versión, GAB está conformado por los siguientes módulos:

- El *Procesador de datos* es un programa de cómputo encargado de modificar –de acuerdo a los valores dados con el *controlador MIDI GAB*- la información registrada en el piano para posteriormente mandar la información procesada a un sintetizador o a un piano MIDI. El programa fue escrito en el lenguaje de programación Java aprovechando los paquetes JMSL desarrollado por Nick Didkovsky y Phil Burk y MidiShare desarrollado en el Laboratorio de Investigación de Música por Computadora del Centro Nacional de Creación Musical GRAME en Lyon Francia.

- El *Controlador MIDI GAB* es un dispositivo que, conectado a una computadora, permite modificar las diferentes variables de cambio por medio de perillas y pedales. Los diferentes valores que adquieran las perillas y pedales tendrán un resultado acústico reconocible.

La versión final del sistema GAB incluirá otros dos módulos:

- El *lector MIDI para pianos acústicos* permitirá leer el material ejecutado en un piano acústico y traducirlo a valores MIDI para así poder realizar los procesos de transformación en el procesador de datos.

- El *reproductor automático para pianos acústicos* será un equipo electrónico capaz de activar -por medio de selenoides- a las teclas de un piano acústico de acuerdo a los valores MIDI que reciba en su puerto de entrada. Este equipo ofrecerá la posibilidad mecánica de alcanzar rangos dinámicos, velocidades y densidades extremas.

Es importante mencionar que el sistema GAB se encuentra en una primera fase de desarrollo. Si bien es cierto que la meta final contempla que todos los módulos y equipos que lo componen sean diseñados y construidos personalmente, dicho objetivo debe de ser visto como un proyecto a largo plazo que excede en tiempo y conocimientos a lo alcanzable en los estudios de licenciatura. En esta primera versión del sistema GAB, el *lector MIDI para pianos acústicos* es sustituido por un equipo comercial y el *reproductor automático para pianos acústicos* es sustituido por un sintetizador.

En el presente trabajo se explica el funcionamiento del sistema GAB. Con respecto al *procesador de datos* se explican sus posibilidades musicales y se expone y detalla su código fuente. Con el propósito de dar claridad al documento, se presenta una breve explicación del lenguaje Java y de los paquetes JMSL y MidiShare. Con relación al *Controlador MIDI* se explica su construcción y funcionamiento, se presenta y detalla el código

fuente del microcontrolador y se da una breve explicación sobre microcontroladores, en particular sobre el BasicStampII.

2 El sistema GAB

GAB es un sistema de reinterpretación musical. Su objetivo es proporcionarle al pianista una herramienta más en su trabajo creativo. Los resultados acústicos finales dependen en gran medida del material que el pianista le proporcione al sistema. GAB no pretende ser un generador de música aleatoria con procesos autónomos; por el contrario, GAB busca ser un equipo que permita la reinterpretación de material musical. GAB puede ser visto como un ensamble de "pianistas virtuales" que acompañan a un pianista principal. El material que ejecutan estos "pianistas" es derivado en su totalidad del material que interpreta el pianista principal en un piano con capacidades MIDI. El nivel de similitud o diferencia entre el nuevo material con respecto al material original estará determinado por el valor que tengan en ese momento las ocho variables que GAB utiliza para generar su material. Cada vez que el pianista toca una nota, GAB genera una o varias estructuras que llamaremos *objeto nota*. Las características de cada *objeto nota* estarán determinadas por los valores que en ese momento presenten las ocho variables que GAB maneja. Estas variables son:

- *Número de repeticiones*: Permite determinar la cantidad de veces que una misma nota es repetida periódicamente.

- *Tempo de repetición*: Permite determinar la velocidad con la que se realizan las repeticiones periódicas definidas con el *Número de repeticiones*.

fuente del microcontrolador y se da una breve explicación sobre microcontroladores, en particular sobre el BasicStampII.

2 El sistema GAB

GAB es un sistema de reinterpretación musical. Su objetivo es proporcionarle al pianista una herramienta más en su trabajo creativo. Los resultados acústicos finales dependen en gran medida del material que el pianista le proporcione al sistema. GAB no pretende ser un generador de música aleatoria con procesos autónomos; por el contrario, GAB busca ser un equipo que permita la reinterpretación de material musical. GAB puede ser visto como un ensamble de "pianistas virtuales" que acompañan a un pianista principal. El material que ejecutan estos "pianistas" es derivado en su totalidad del material que interpreta el pianista principal en un piano con capacidades MIDI. El nivel de similitud o diferencia entre el nuevo material con respecto al material original estará determinado por el valor que tengan en ese momento las ocho variables que GAB utiliza para generar su material. Cada vez que el pianista toca una nota, GAB genera una o varias estructuras que llamaremos *objeto nota.* Las características de cada *objeto nota* estarán determinadas por los valores que en ese momento presenten las ocho variables que GAB maneja. Estas variables son:

- *Número de repeticiones*: Permite determinar la cantidad de veces que una misma nota es repetida periódicamente.

- *Tempo de repetición*: Permite determinar la velocidad con la que se realizan las repeticiones periódicas definidas con el *Número de repeticiones*.

- *Nivel de variación rítmica*: Permite determinar la exactitud en el tempo de las repeticiones de cada nota. Niveles bajos producirán pulsos regulares y niveles altos producirán pulsos con irregularidades rítmicas.

- *Nivel de variación de registro*: Permite definir la distancia máxima posible (en número de octavas) entre la nota que toca el pianista y el *objeto nota* generado por GAB.

- *Nivel de disonancia*: Permite determinar el tipo de intervalos posibles entre la nota que toca el pianista y el *objeto nota* generado por el sistema.

- *Nivel de densidad*: Permite definir la cantidad de *objetos nota* que serán creados por cada nota que sea tocada por el pianista.

- *Variación de intensidad*: Permite determinar la diferencia de intensidades posibles entre la nota que toca el pianista y el *objeto nota* generado por GAB.

- *Duración de las notas*: Permite determinar la duración de las notas que conforman un *objeto nota.*

Algunas de las características de los *objetos nota* están previamente definidas y no son modificables; otras, como la locación de un *objeto nota* son escogidas con procesos aleatorios realizados en tiempo real.

En esta primera versión, el sistema GAB está conformado por dos módulos. El *procesador de datos GAB* es un programa de computadora en donde es posible observar el valor de las ocho variables y es el encargado de recibir la información del pianista

(vía MIDI) y los valores del *controlador MIDI GAB* (vía MIDI). Con dicha información, el programa realiza los diferentes cálculos matemáticos necesarios para generar el nuevo material musical el cual es mandado por el puerto de salida MIDI OUT de la computadora.

El *controlador MIDI GAB* es un equipo electrónico encargado de convertir el valor de hasta quince potenciómetros (para esta versión sólo se usan ocho) en información MIDI. El *controlador MIDI GAB* es un equipo posibilita al pianista modificar de una forma cómoda las variables del sistema y puede ser colocado cerca del teclado del piano.

La versión actual del sistema GAB está diseñada para generar el material reinterpretado en seis canales MIDI. Para aprovechar los efectos de espacialización sonora se sugiera que estos seis "pianistas virtuales" estén dispuestos en una sala de conciertos de acuerdo al esquema de la figura 1.

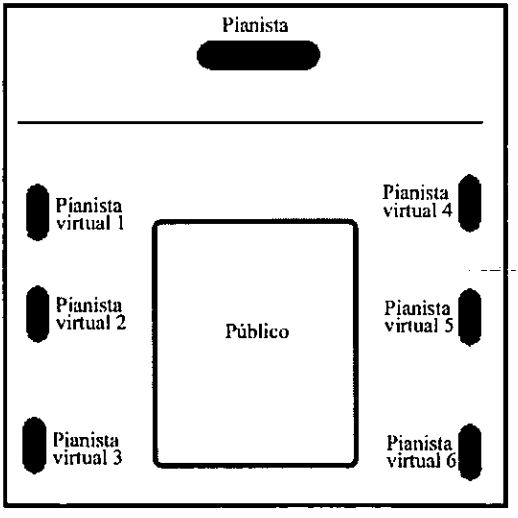


Fig 1

Para poder utilizar el sistema GAB se requiere de:

- Una computadora personal PC o Mac con dos puertos MIDI de entrada y uno de salida con el sistema GAB correctamente instalado.
- Un piano electrónico MIDI o un piano acústico con implementación MIDI.
- Seis pianos electrónicos o un sintetizador multicanal con sonido de piano y seis salidas de audio independientes.
- En caso de usar un sintetizador es necesario un sistema de amplificación de seis canales.

2.1 El procesador de datos GAB 1.0

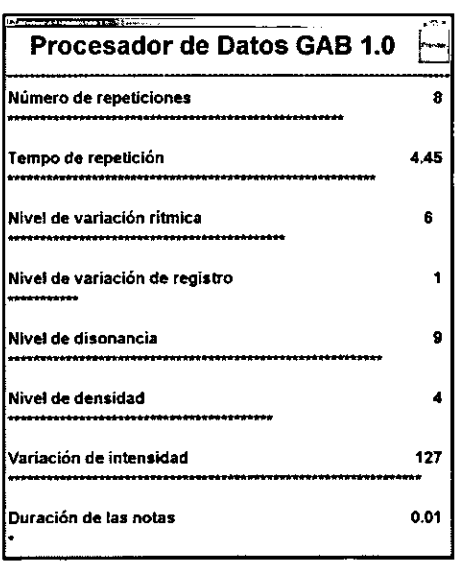


Figura 2

2.1.1 Funciones del procesador de datos

Una vez que el *procesador de datos* aparece en pantalla (Figura 2) es necesario presionar el botón *Prender* para activar los puertos MIDI. No es necesario apagar el sistema pues esto se realiza automáticamente al cerrar el programa. Cuando el programa es iniciado los valores de las ocho variables se establecen con su valor inicial. En la Tabla 1 se presentan: el valor inicial y los valores mínimo y máximo permitidos.

Controlador	Valor inicial	Valor mínimo	Valor máximo
Número de repeticiones	0 Rep.	0 Rep.	10 Rep.
Tempo de repetición	.05 Seg.	.05 Seg.	5 Seg.
Nivel de variación rítmica	0	0	10
Nivel de variación de registro	0 Octavas	0 Octavas	7 Octavas
Nivel de disonancia	0	0	11
Nivel de densidad	1 Nota	1 Nota	6 Notas
Variación de intensidad	0	0	127
Duración de las notas	.01 Seg.	.01 Seg.	5 Seg.

Tabla 1

Al mover cada una de las perillas del *Controlador MIDI GAB* se podrá ver el cambio en su respectivo valor numérico en pantalla y una línea de asteriscos debajo del nombre cuya dimensión irá en incremento o decremento según la posición de la perilla. Dicha línea representa gráficamente el valor actual de la perilla correspondiente.

Las variables *Número de repeticiones*, *Tempo de repetición* y *Nivel de variación rítmica*:

Llamaremos *objeto nota* a un conjunto de notas con alturas, intensidades, duraciones y locaciones iguales generadas a partir de una sola nota del piano principal. Por tanto, cuando hablemos de la altura, la duración, la intensidad o la locación de un *objeto nota* nos estaremos refiriendo a la altura, la duración, la intensidad o la locación de cada una de las notas que conforman dicho objeto. Cada que el pianista toca una nota en el piano, el procesador de datos genera uno o varios *objeto nota* (dependiendo del *Nivel de densidad*). La cantidad de notas que conformen un *objeto nota* está dada por la variable *Número de repeticiones*. Si esta variable presenta un valor de cero, entonces el *objeto nota* estará integrado por una sola nota que sonará en el mismo momento que la nota del piano principal (cero repeticiones).

Si la variable *Número de repeticiones* presenta un valor de cero, el cambio en las variables *Tempo de repetición* y *Nivel de variación rítmica* no tiene efecto. Por el contrario, si la variable *Número de repeticiones* es mayor que cero, entonces la variable *Tempo de repetición* indica la velocidad a la que se realizarán las repeticiones y la variable *Nivel de variación rítmica* nos permitirá definir la precisión rítmica de dichas repeticiones. Niveles bajos de *variación rítmica* generarán pulsos regulares; a medida que el nivel de *variación rítmica* se incrementa, el ritmo se vuelve cada vez más irregular.

El índice máximo de *variación rítmica* definido en el programa se fijó en cuatro unidades por lo que el nivel máximo de *variación rítmica* (127) modifica al azar el valor establecido en la variable *Tempo de repetición* en un rango que puede ir de una cuarta parte de su valor, hasta cuatro veces su valor (Tabla 2).

Índice de variación rítmica	Valor mínimo posible	Valor máximo posible
0	1/1 del valor (sin modificación)	1 vez el valor (sin modificación)
64	½ del valor	2 veces el valor
127	¼ del valor	4 veces el valor

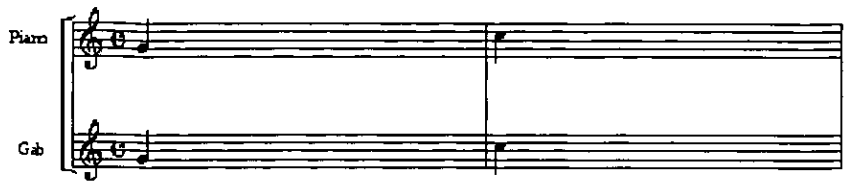
Tabla 2

Un *objeto nota* creado con los valores de la Tabla 3 tendría las características del Ejemplo 1.[1]

Controlador	Valor	Controlador	Valor
Número de repeticiones	**0 Rep.**	Nivel de disonancia	0
Tempo de repetición	**1.0 Seg.**	Nivel de densidad	1 Nota
Nivel de variación rítmica	**0**	Variación de intensidad	0
Nivel de variación de registro	0 Octavas	Duración de las notas	1.0 Seg.

Tabla 3

1 Recordemos que la locación (canal MIDI de salida) es escogida al azar de entre los primeros 6 canales MIDI. Una vez definida la locación, el objeto nota *se toca en dicho lugar. Por tal motivo y para fines de simplicidad sólo se utilizarán dos sistemas para exponer cada ejemplo. Uno para el pianista y otro para el "pianista virtual" que podrá ser uno cualquiera de entre los seis "pianistas virtuales".*

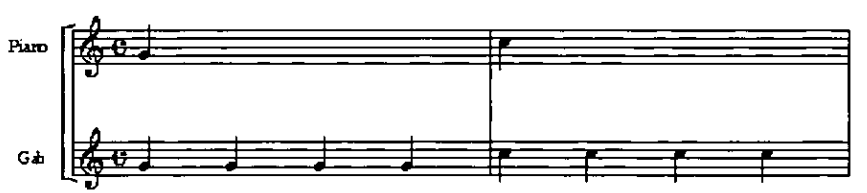


Ejemplo 1

Si modificamos el valor de la variable *Número de repeticiones* de acuerdo con la Tabla 4 el resultado sonoro sería el del Ejemplo 2:

Controlador	**Valor**	**Controlador**	**Valor**
Número de repeticiones	**3 Rep.**	Nivel de disonancia	0
Tempo de repetición	**1.0 Seg.**	Nivel de densidad	1 Nota
Nivel de variación rítmica	**0**	Variación de intensidad	0
Nivel de variación de registro	0 Octavas	Duración de las notas	1.0 Seg.

Tabla 4

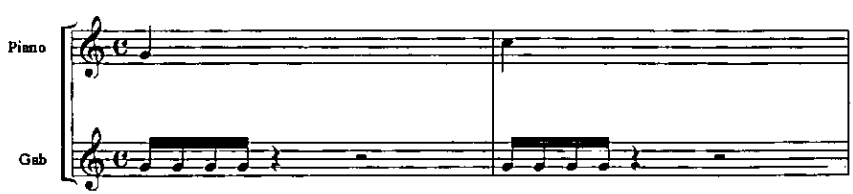


Ejemplo 2

Si mantenemos el *Número de repeticiones* en tres y el *Nivel de variación rítmica* en cero pero modificamos el *Tempo de repetición* de acuerdo a los valores de la Tabla 5, la velocidad en las repeticiones cambiará como se puede apreciar en el Ejemplo 3.

Controlador	**Valor**	**Controlador**	**Valor**
Número de repeticiones	**3 Rep.**	Nivel de disonancia	0
Tempo de repetición	**0.25 Seg.**	Nivel de densidad	1 Nota
Nivel de variación rítmica	**0**	Variación de intensidad	0
Nivel de variación de registro	0 Octavas	Duración de las notas	.25 Seg.

Tabla 5



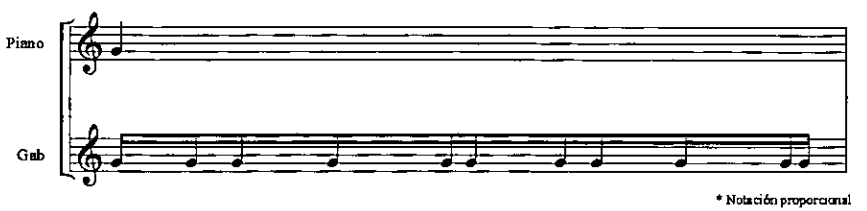
Ejemplo 3

Como se ha explicado previamente, el *Nivel de variación rítmica* permite modificar la precisión rítmica de las repeticiones, el Ejemplo 4[2] muestra los resultados de utilizar los valores de la Tabla 6:

2 Este ejemplo está escrito en notación proporcional.

Controlador	Valor	Controlador	Valor
Número de repeticiones	**10 Rep.**	Nivel de disonancia	0
Tempo de repetición	**0.25 Seg.**	Nivel de densidad	1 Nota
Nivel de variación rítmica	**4**	Variación de intensidad	0
Nivel de variación de registro	0 Octavas	Duración de las notas	.01 Seg.

Tabla 6



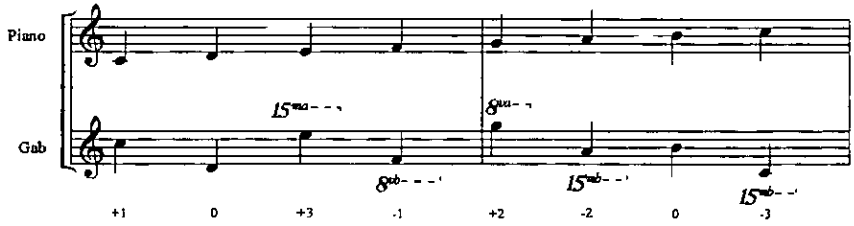
Ejemplo 4

La variable *Nivel de variación de registro*:

En los ejemplos anteriores hemos visto que la altura de todos los *objetos nota* ha sido la misma que la tocada en el piano principal. Esto se debe a que las variables *Nivel de Variación de registro* y *Nivel de disonancia* se han mantenido en cero. La variable *Nivel de variación de registro* permite que la altura de un *objeto nota* se encuentre en una octava diferente a la nota original. La variable *Nivel de variación de registro* define la cantidad máxima posible de octavas entre la nota original y el *objeto nota.* Un proceso aleatorio define, dentro del rango establecido, la cantidad de octavas ascendentes o descendentes que habrá entre la nota original y el *objeto nota.* El Ejemplo 5 muestra los resultados de utilizar los valores de la Tabla 7:

Controlador	Valor	Controlador	Valor
Número de repeticiones	0 Rep.	Nivel de disonancia	0
Tempo de repetición	1.0 Seg.	Nivel de densidad	1 Nota
Nivel de variación rítmica	0	Variación de intensidad	0
Nivel de variación de registro	**3 Octavas**	Duración de las notas	1.0 Seg.

Tabla 7



Ejemplo 5

La variable *Nivel de disonancia*:

Así como la variable *Nivel de variación de registro* se encarga de modificar la distancia en octavas entre la nota del piano y el *objeto nota*, la variable *Nivel de disonancia* se encarga de modificar el intervalo entre ambas notas. Nuevamente, un proceso aleatorio se encarga de definir el intervalo entre las notas. El valor del *Nivel de disonancia* permite establecer el tipo de intervalos posibles. En la Tabla 8 se observa cómo a medida que el *valor de disonancia* aumenta se van agregando posibles

intervalos al proceso de selección. El intervalo entre la nota del piano y el objeto nota siempre será ascendente.

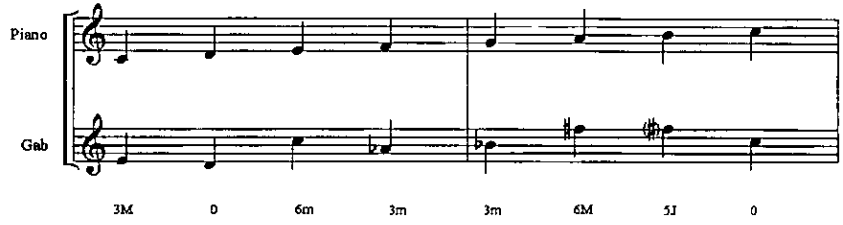
Nivel de disonancia	Intervalos posibles
0	Unísono
1	Unísono, 3M
2	Unísono, 3M, 3m
3	Unísono, 3M, 3m, 6M
4	Unísono, 3M, 3m, 6M, 6m
5	Unísono, 3M, 3m, 6M, 6m, 5J
6	Unísono, 3M, 3m, 6M, 6m, 5J, 4J
7	Unísono, 3M, 3m, 6M, 6m, 5J, 4J, 2M
8	Unísono, 3M, 3m, 6M, 6m, 5J, 4J, 2M, 2m
9	Unísono, 3M, 3m, 6M, 6m, 5J, 4J, 2M, 2m, 7m
10	Unísono, 3M, 3m, 6M, 6m, 5J, 4J, 2M, 2m, 7m, 7M
11	Unísono, 3M, 3m, 6M, 6m, 5J, 4J, 2M, 2m, 7m, 7M, 4A

Tabla 8

Veamos cómo los valores de la Tabla 9 producen intervalos de unísono, tercera menor, tercera mayor, sexta menor, sexta mayor y quinta justa (Ejemplo 6):

Controlador	Valor	Controlador	Valor
Número de repeticiones	0 Rep.	**Nivel de disonancia**	**5**
Tempo de repetición	1.0 Seg.	Nivel de densidad	1 Nota
Nivel de variación rítmica	0	Variación de intensidad	0
Nivel de variación de registro	0 Octavas	Duración de las notas	1.0 Seg.

Tabla 9



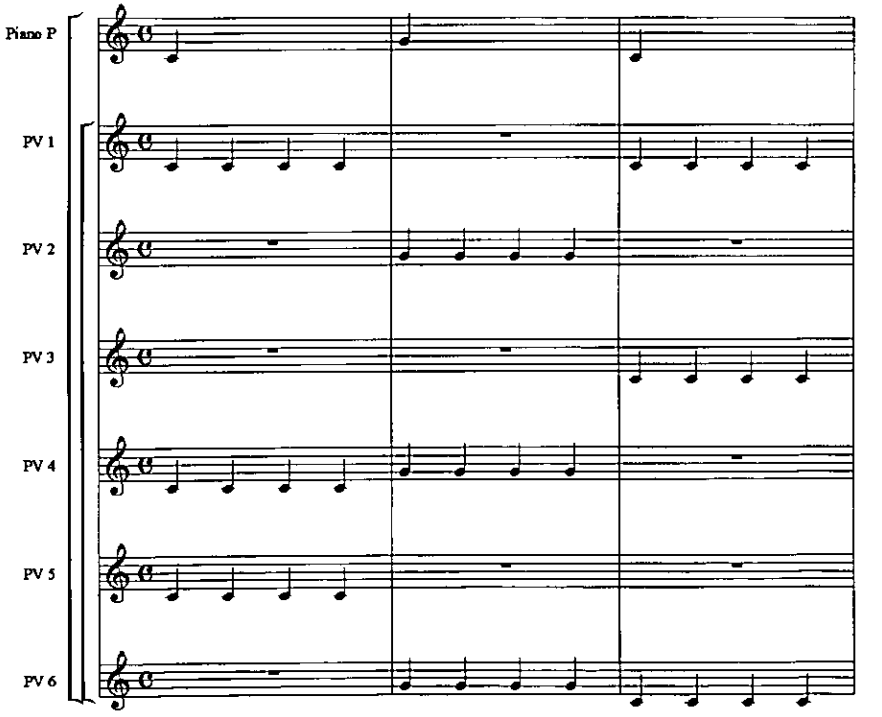
Ejemplo 6

La variable *Nivel de densidad*:

Hasta este momento, todos los ejemplos que se han visto han contemplado un *Nivel de Densidad* de uno. Con este valor, únicamente se crea un *objeto nota* por cada nota del piano principal. Sin embargo, es posible generar más de un *objeto nota* por cada nota del piano principal. La variable *Nivel de densidad* establece la cantidad de *objetos nota* que deberán crearse por cada nota. Recordemos que la locación de cada *objeto nota* se establece de forma aleatoria. Veamos el Ejemplo 7 resultado de utilizar los valores de la Tabla 10.

Controlador	Valor	Controlador	Valor
Número de repeticiones	3 Rep.	Nivel de disonancia	0
Tempo de repetición	1.0 Seg.	**Nivel de densidad**	**3 Notas**
Nivel de variación rítmica	0	Variación de intensidad	0
Nivel de variación de registro	0 Octavas	Duración de las notas	1.0 Seg.

Tabla 10



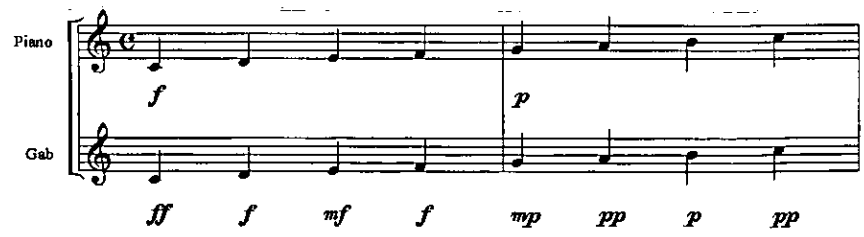
Ejemplo 7

La variable *Variación de intensidad*:

El procesador de datos permite modificar la intensidad de los *objetos nota*. Si la variable presenta un valor de cero, los *objetos nota* se crean con la misma intensidad que la nota del piano principal. A medida que el valor de la variable *Variación de intensidad* aumenta, la diferencia de intensidad entre la nota original y el objeto nota es mayor. Un proceso aleatorio aumenta o disminuye la intensidad del objeto nota con respecto a la nota original según el valor de esta variable. En el Ejemplo 8 se puede ver cómo la intensidad de las notas generadas por el sistema es medianamente diferente a las notas del piano principal. Los valores de las variables para este ejemplo se muestran en la Tabla 11.

Controlador	**Valor**	**Controlador**	**Valor**
Número de repeticiones	1 Rep.	Nivel de disonancia	0
Tempo de repetición	1.0 Seg.	Nivel de densidad	1 Nota
Nivel de variación rítmica	0	**Variación de intensidad**	**64**
Nivel de variación de registro	0 Octavas	Duración de las notas	1.0 Seg.

Tabla 11



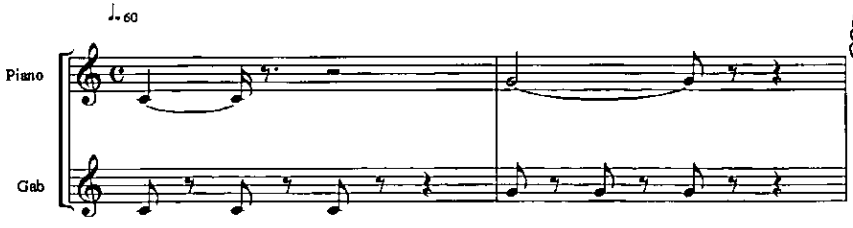
Ejemplo 8

La variable *Duración de las notas*:

Hasta este momento y por fines de notación en los ejemplos, la duración de las notas (tanto en el piano principal, como en los *objetos nota*) se ha mantenido de forma esquemática. El *procesador de datos GAB*, no reconoce la duración de una nota tocada por el pianista, únicamente recibe la información del momento en que es presionada. Por tal motivo, la duración de las notas de un *objeto nota* es siempre fija y está definida por la variable *Duración de las notas*. El rango para esta variable va de 0.01 Seg. a 5.0 segundos. En el Ejemplo 9 se puede ver que, independientemente de la duración de la nota principal, las notas generadas por el sistema mantienen una duración constante. La Tabla 12 muestra los valores utilizados para el Ejemplo 9.

Controlador	**Valor**	**Controlador**	**Valor**
Número de repeticiones	2 Rep.	Nivel de disonancia	0
Tempo de repetición	1.0 Seg.	Nivel de densidad	1 Nota
Nivel de variación rítmica	0	Variación de intensidad	0
Nivel de variación de registro	0 Octavas	**Duración de las notas**	**.5 Seg.**

Tabla 12



Ejemplo 9

ESTA TESIS NO DEBE SALIR DE LA

2.1.2 El lenguaje de programación Java

Java es un lenguaje de programación orientado a objetos diseñado para que los programas desarrollados con él puedan ser ejecutados en diferentes plataformas y diferentes sistemas operativos. Java fue desarrollado por Sun Microsystems e incluye características especiales que lo hacen ideal para la programación en Internet, entre ellas la simplicidad para incluir gráficos y efectos en páginas web.

Java es un lenguaje orientado a objetos, esto quiere decir que todo (o casi todo en él) son objetos. Una *clase* es una colección de *datos* y *métodos* que definen a un particular tipo de *objeto*. Las *clases* pueden ser extendidas a partir de otras *clases* y de esta manera pueden ser reutilizadas en diferentes circunstancias. Tomemos como ejemplo una *clase* que llamaremos *avión*. Dicha *clase* es resultado de agrupar las *clases*: *ala*, *motor*, *cabina* y *hélice*. Un *método* puede ser visto como la actividad que puede realizar una *clase*. De esta manera los *métodos* de la clase *avión* podrían ser: *despegar*, *aterrizar* y *planear*. Para hacer una clase *avión de pasajeros* solo será necesario crear el objeto *asiento*, tomar nuestra clase *avión* y extenderla a *avión de pasajeros* agregándole la clase *asiento* y el método *poner película*. Del mismo modo las clases *asiento* y *motor* nos podrían servir para diseñar nuestra clase *automóvil*. JMSL y MidiShare son un conjunto de clases escritas en Java con aplicaciones específicamente musicales.

2.1.3 JMSL y MidiShare

JMSL (Java Music Specification Language) es una herramienta diseñada en Java enfocada al diseño de composiciones con algoritmos e instrumentos inteligentes con aplicaciones en la ejecución musical en vivo. JMSL es el sucesor

de HMSL (Hierarchical Music Specification Language) lenguaje, este último, basado en FORTH y diseñado en el Mills College Center for Contemporary Music por Phil Burk, Larry Polansky y David Rosenboom. En 1997 al percatarse de que la computadora Amiga quedaría fuera del mercado y con el surgimiento de un lenguaje que prometía funcionar independientemente de la plataforma, Nick Didkovsky inició la exportación de HMSL a Java. Finalmente, al ver la gran ventaja que ofrecía JMSL, el miso Didkovsky y el coautor de HMSL, Phil Burk, no sólo exportaron HMSL a Java sino que aprovecharon las posibilidades de este último para mejorar y simplificar HMSL. JMSL amplía a Java mediante una gran cantidad de clases con aplicaciones musicales: temporalización jerárquica de objetos musicales, generación de secuencias, funciones de distribución, etc. Estas clases ofrecen un territorio muy amplio para la exploración musical.

La solidez de JMSL radica en su derivación de Java lo que lo hace un sistema abierto al propio desarrollo de Java incluyendo su conectividad a bases de datos, a las herramientas propias de las redes, a los paquetes gráficos, y a numerosos paquetes en constante implementación. Actualmente JMSL soporta los paquetes MidiPort de Robert Marsanyi, MidiShare de GRAME y JSyn de Phil Burk.

La noción de jerarquía es el punto central de JMSL. Una jerarquía es una simple colección de relaciones padre-hijo. Un padre podría ser, por ejemplo, una sinfonía. El primero, el segundo y el tercer movimientos serían los hijos. Como el segundo movimiento no inicia sino hasta terminar el primer movimiento, y el tercer movimiento no inicia sino hasta terminar el segundo, este es un caso de una *colección secuencial* de eventos. Por otro lado, un ejemplo de colección paralela sería el objeto *cuarteto de cuerdas*, en este caso los hijos: violín primero,

violín segundo, viola y chelo deben de iniciar al mismo tiempo y mantenerse en constante sincronía.

JMSL permite agendar eventos arbitrarios en el tiempo (musicales y no musicales), todos estos "trabajos musicales" se pueden agrupar en colecciones y a su vez las colecciones pueden ser agrupadas en supercolecciones. Finalmente las supercolecciones, las colecciones y cualquier "trabajo musical" puede ser agendado en el tiempo ya sea paralela o secuencialmente con respecto a otras colecciones.

JMSL ofrece también clases para generar, modificar y acceder a listados musicales (MusicShapes) que son arreglos numéricos de dos dimensiones equivalentes a tablas con un número cualquiera de renglones y columnas. Dichos arreglos pueden ser utilizados e interpretados por un *trabajo musical* de muchas formas diversas. Por ejemplo para definir duración de un evento, intensidad, número de repeticiones, índices de variación tímbrica, procesos estocásticos, etc.

Como se pude apreciar, JMSL es una herramienta muy útil en los campos de la composición y la ejecución musical. Permite diseñar y solucionar ideas musicales complejas y al ser un sistema abierto permite relacionar eventos musicales con otro tipo de estructuras.

Por su parte, MidiShare 1.86 fue desarrollado en el Laboratorio de Investigación de Música por Computadora del Centro Nacional de Creación Musical GRAME en Lyon Francia. Al igual que JMSL, MidiShare amplía a Java mediante un grupo de clases con aplicaciones en el uso del protocolo MIDI. Con las clases de MidiShare es posible acceder a los puertos MIDI de una computadora y realizar una gran cantidad de operaciones con la información obtenida.

2.1.4 Explicación y transcripción del código fuente

El programa Gab está compuesto de tres clases: A) la clase *Gab*, que es la clase principal, extendida de *Applet* y que implementa dos interfases: *MidiListener* -para poder recibir MIDI- y *ActionListener* -para poder responder al uso del botón *Prender*-; B) la clase *NotaCompuesta*, extendida de *MusicJob* encargada de generar la estructura de los *objetos nota* y C) la clase *NotaBase* encargada de generar las notas que conforman a los *objetos nota*. Esta última implementa la interfase *Playable* con el fin de poder ser utilizada en los métodos de agendación de la clase *NotaCompuesta*.

Al principio de la clase *Gab* se definen e inicializan todas las variables necesarias para el funcionamiento del programa: variables numéricas, arreglos, etiquetas, botones y tipos de letra. En el método *Main* se definen los valores necesarios para poder ejecutar el programa como una aplicación fuera de páginas web. En el método *start* se crean y agregan el título, el botón y todas las etiquetas a la interfase para el usuario del programa. El método *inicio* es invocado cada que el botón *Prender* es presionado y se encarga de prender los puertos MIDI. En el método *cambiaGráficos* se encuentran todas las rutinas necesarias para poder actualizar los valores y líneas de asteriscos en la interfase al usuario cada que es recibido el mensaje MIDI correspondiente. El método *actionPerformed* se encarga de definirle al botón *Prender* su capacidad de prender las rutinas MIDI.

Los métodos *handleNoteOn*, *handleNoteOff*, *handlePolyphonicAftertouch*, *handleControlChange*, *handleProgramChange*, *handleChannelAftertouch* y *handlePitchBend* son los métodos encargados de recibir la información MIDI proveniente del Puerto MIDI IN de la

computadora. Todos estos métodos, a excepción de *handleControlChange* y *handleNoteOn*, se encargan de mandar sin modificación la información que reciben en su entrada al puerto de salida MIDI Out de la computadora.

Los métodos *handleNoteOn* y *handleControlChange* son los métodos MIDI (implementados por *MIDIListener*) importantes en el programa. El primero se encarga de invocar al método *alTocarNota* que se describirá más adelante. El segundo se encarga de actualizar las variables que generan los valores de los ocho parámetros que GAB puede modificar. Los números de *ControlChange* que el programa reconoce para realizar los procesos son[3]:

Parámetros en GAB	**Número de Control Change**
Número de repeticiones	16
Tempo de repetición	17
Nivel de variación rítmica	18
Nivel de variación de registro	19
Nivel de disonancia	20
Nivel de densidad	21
Variación de intensidad	22
Duración de las notas	23

Tabla 13

3 Los valores 16 a 23 fueron escogidos debido a que en las especificaciones MIDI son valores para propósitos generales y no interfieren con las funciones de los equipos comerciales.

Cada vez que una nota es recibida desde el piano principal se crea y activa un objeto *NotaCompuesta* utilizando el valor de altura e intensidad de dicha nota. El método *alTocarNota* es el encargado directo de generar todos los *objetos nota*. La clase *NotaCompuesta* es el molde o marco de las estructuras *objeto nota*. Esta clase utiliza a la *notaBase* y a los valores de *Número de repeticiones* y *Tempo de repeticiones* para generar las características particulares de cada *objeto nota*. Además, esta clase utiliza el valor de *Nivel de densidad* para decidir la cantidad de *objetos nota* que deberán ser creados. La clase *NotaBase* es el molde de las notas que conforman a un *objeto nota*, en dicha clase están la mayoría de los métodos utilizados para generar la altura, la intensidad, la duración y la locación de los *objetos nota* según el valor que en ese momento presenten los parámetros de *Nivel de variación de registro*, *Nivel de disonancia*, *Variación de intensidad* y *Duración de las notas*.

A continuación se transcribe íntegro y sin modificaciones el código fuente del programa:

```
/* Procesador de Datos GAB
** versión 1.0
** autor Hugo Solís
** Fecha 30 de octubre del 2001
** Al recibir Notas MIDI en el puerto de entrada 0, el programa genera notas
MIDI en el puerto de salida 0
**con diferentes características de acuerdo a los valores de los mensajes
Control Change 16 a 23 que reciba.
**El programa genera un applet con un botón para activar MIDI y permite
visualizar los valores de las ocho
**variables que utiliza para generar las Notas. Las variables son: Número de
repeticiones, Tempo de repetición,
```

```
**Nivel de variación rítmica, Nivel de variación de registro, Nivel de disonancia, Nivel de densidad

**Variación de intensidad y Duración de las notas.

** Este procesador está diseñado para ser usado con el Controlador MIDI GAB

**Para detalles de funcionamiento ver la tesis "Gab 1.0, sistema de reinterpretación electrónica para pianos acústicos" de Hugo Solís

*/


package gabver1;


import java.awt.*;

import java.awt.event.*;

import com.softsynth.jmsl.*;

import com.softsynth.jmsl.midi.*;


public class Gab extends java.applet.Applet implements MidiListener, ActionListener

{

 //variables de inicialización

  int numDeCanales = 6;

  double durMaxDeNota = 5;

  double durMinDeNota = .01;

  int maxDeDensidad = 6;

  int numMaxDeRepeticiones = 10;

  double tiempoMinimoDeRepeticion = .05;

  double tiempoMaximoDeRepeticion = 5.0;

  double unidadDeVariacionRitmica = 4.0;
```

```
//variables generales
int varRep = 0;
int varTem = 0;
int varRit = 0;
int varReg = 0;
int varDis = 0;
int varDen = 0;
int varInt = 0;
int varDur = 0;
int dis[] = {0,4,3,9,8,7,5,2,1,10,11,6};
int altura;
int intensidad;
MidiParser midiFrases;

//Etiquetas, botones y tiposdeletra
Label etiTitulo;
Label etiLinea;
Label etiNumDeRep;
Label etiTmpDeRep;
Label etiVarDeRit;
Label etiVarDeReg;
Label etiNivDeDis;
Label etiNivDeDen;
Label etiVarDeInt;
Label etiNivDeDur;
Label etiNumDeRepB;
Label etiTmpDeRepB;
```

```
Label etiVarDeRitB;
Label etiVarDeRegB;
Label etiNivDeDisB;
Label etiNivDeDenB;
Label etiVarDeIntB;
Label etiNivDeDurB;
Label etiNumDeRepC;
Label etiTmpDeRepC;
Label etiVarDeRitC;
Label etiVarDeRegC;
Label etiNivDeDisC;
Label etiNivDeDenC;
Label etiVarDeIntC;
Label etiNivDeDurC;
Button botonPaPrender;
Font titFont;
Font etiFont;

        // Definición de Main
        public static void main(String args[])
        {
                Gab gaby = new Gab();
                gaby.start();
                gaby.setSize(1100,1100);
                gaby.setVisible(true);
                Frame f = new Frame("Interfase del Sistema GAB 1.0");
                f.addWindowListener(new java.awt.event.WindowAdapter()
```

```
                {
                        public void
windowClosing(java.awt.event.WindowEvent e)
                        {
                        JMSL.midi.closeDevices();
                        System.exit(0);
                        }
                });
        f.add(gaby, BorderLayout.CENTER);
        f.setSize(1100,1025);
        f.setVisible(true);
        }

        //Definición del método principal Gab
        public void start()
        {
                setLayout (new BorderLayout());
                //Definición de tipos de letra, etiquetas y paneles
                Font titFont = new Font("TimesRoman", Font.BOLD, 48);
                Font etiFont = new Font("TimesRoman", Font.BOLD, 30);

                etiTitulo = new Label("Procesador de Datos GAB 1.0",
Label.CENTER);
                etiTitulo.setFont(titFont);
                botonPaPrender = new Button("Prender");
                etiLinea = new Label("----------------------------------------
----------------------------------------------------------", Label.CENTER);
                etiLinea.setFont(etiFont);
```

```
                etiNumDeRep = new Label("Número de repeticiones",
Label.LEFT);
                etiNumDeRep.setFont(etiFont);
                etiTmpDeRep = new Label("Tempo de repetición",
Label.LEFT);
                etiTmpDeRep.setFont(etiFont);
                etiVarDeRit = new Label("Nivel de variación rítmica",
Label.LEFT);
                etiVarDeRit.setFont(etiFont);
                etiVarDeReg = new Label("Nivel de variación de registro",
Label.LEFT);
                etiVarDeReg.setFont(etiFont);
                etiNivDeDis = new Label("Nivel de disonancia",
Label.LEFT);
                etiNivDeDis.setFont(etiFont);
                etiNivDeDen = new Label("Nivel de densidad",
Label.LEFT);
                etiNivDeDen.setFont(etiFont);
                etiVarDeInt = new Label("Variación de intensidad",
Label.LEFT);
                etiVarDeInt.setFont(etiFont);
                etiNivDeDur = new Label("Duración de las notas",
Label.LEFT);
                etiNivDeDur.setFont(etiFont);
                etiNumDeRepB = new Label("0     ", Label.LEFT);
                etiNumDeRepB.setFont(etiFont);
                etiTmpDeRepB = new
Label(Double.toString(tiempoMinimoDeRepeticion), Label.LEFT);
                etiTmpDeRepB.setFont(etiFont);
                etiVarDeRitB = new Label("0     ", Label.LEFT);
```

```
        etiVarDeRitB.setFont(etiFont);
        etiVarDeRegB = new Label("0    ", Label.LEFT);
        etiVarDeRegB.setFont(etiFont);
        etiNivDeDisB = new Label("0    ", Label.LEFT);
        etiNivDeDisB.setFont(etiFont);
        etiNivDeDenB = new Label("1    ", Label.LEFT);
        etiNivDeDenB.setFont(etiFont);
        etiVarDeIntB = new Label("0    ", Label.LEFT);
        etiVarDeIntB.setFont(etiFont);
        etiNivDeDurB = new Label(Double.toString(durMinDeNota),
Label.LEFT);
        etiNivDeDurB.setFont(etiFont);
        etiNumDeRepC = new Label("*", Label.LEFT);
        etiNumDeRepC.setFont(etiFont);
        etiTmpDeRepC = new Label("*", Label.LEFT);
        etiTmpDeRepC.setFont(etiFont);
        etiVarDeRitC = new Label("*", Label.LEFT);
        etiVarDeRitC.setFont(etiFont);
        etiVarDeRegC = new Label("*", Label.LEFT);
        etiVarDeRegC.setFont(etiFont);
        etiNivDeDisC = new Label("*", Label.LEFT);
        etiNivDeDisC.setFont(etiFont);
        etiNivDeDenC = new Label("*", Label.LEFT);
        etiNivDeDenC.setFont(etiFont);
        etiVarDeIntC = new Label("*", Label.LEFT);
        etiVarDeIntC.setFont(etiFont);
        etiNivDeDurC = new Label("*", Label.LEFT);
```

```
etiNivDeDurC.setFont(etiFont);

Panel n = new Panel();
n.setLayout(new BorderLayout());
n.add("Center", etiTitulo);
n.add("East", botonPaPrender);
n.add("South", etiLinea);

Panel c = new Panel();
c.setLayout(new GridLayout(8,1,0,25));

Panel c1 = new Panel();
c1.setLayout(new BorderLayout());
c1.add("Center", etiNumDeRep);
c1.add("East", etiNumDeRepB);
c1.add("South", etiNumDeRepC);
Panel c2 = new Panel();
c2.setLayout(new BorderLayout());
c2.add("Center", etiTmpDeRep);
c2.add("East", etiTmpDeRepB);
c2.add("South", etiTmpDeRepC);
Panel c3 = new Panel();
c3.setLayout(new BorderLayout());
c3.add("Center", etiVarDeRit);
c3.add("East", etiVarDeRitB);
c3.add("South", etiVarDeRitC);
Panel c4 = new Panel();
```

```
c4.setLayout(new BorderLayout());
c4.add("Center", etiVarDeReg);
c4.add("East", etiVarDeRegB);
c4.add("South", etiVarDeRegC);
Panel c5 = new Panel();
c5.setLayout(new BorderLayout());
c5.add("Center", etiNivDeDis);
c5.add("East", etiNivDeDisB);
c5.add("South", etiNivDeDisC);
Panel c6 = new Panel();
c6.setLayout(new BorderLayout());
c6.add("Center", etiNivDeDen);
c6.add("East", etiNivDeDenB);
c6.add("South", etiNivDeDenC);
Panel c7 = new Panel();
c7.setLayout(new BorderLayout());
c7.add("Center", etiVarDeInt);
c7.add("East", etiVarDeIntB);
c7.add("South", etiVarDeIntC);
Panel c8 = new Panel();
c8.setLayout(new BorderLayout());
c8.add("Center", etiNivDeDur);
c8.add("East", etiNivDeDurB);
c8.add("South", etiNivDeDurC);

c.add(c1);
c.add(c2);
```

```
            c.add(c3);
            c.add(c4);
            c.add(c5);
            c.add(c6);
            c.add(c7);
            c.add(c8);

            add("North", n);
            add("Center", c);

            // Definición de receptores de acción
            botonPaPrender.addActionListener(this);

            WindowAdapter myWindowAdapter = new
WindowAdapter()
            {
                  public void windowClosing(WindowEvent e)
                  {
                        JMSL.midi.closeDevices();
                        System.exit(0);
                  }
            };

            //Generador de valores aleatorios
            JMSLRandom.randomize();
      }
```

```
        // Definición del método inicio, se ejecuta el precionar el boton
"prender"
        public void inicio()
        {
                JMSL.midi.openDevices(0,0);
                MidiParser midiFrases = new MidiParser();
                JMSL.midi.addMidiParser( midiFrases );
                midiFrases.addMidiListener(this);
                System.out.println("Se ha iniciado el programa");
        }

        //Definición del método cambiaGráficos, encargado de actualizar los
valores en pantalla, es invocado cada que se recibe un ControlChange 16-23
        public void cambiaGraficos(int id, int valor)
        {
                StringBuffer cadena = new StringBuffer("*");
                for (int i = 0; i < valor/2; i++)
                {
                        cadena.append("*");
                }

                switch(id)
                {
                        case 16:

        etiNumDeRepB.setText(Integer.toString((varRep *
numMaxDeRepeticiones / 127)));
                                etiNumDeRepC.setText(cadena.toString());
```

```
                    break;
                    case 17:
etiTmpDeRepB.setText(Double.toString( new Double(new
Double(100*(Math.min((varTem * tiempoMaximoDeRepeticion / 127) +
tiempoMinimoDeRepeticion,
tiempoMaximoDeRepeticion))).intValue()).doubleValue() /100));
                        etiTmpDeRepC.setText(cadena.toString());
                    break;
                    case 18:

        etiVarDeRitB.setText(Integer.toString((varRit * 10 / 127)));
                        etiVarDeRitC.setText(cadena.toString());
                    break;
                    case 19:

        etiVarDeRegB.setText(Integer.toString(valor * 8 / 127));
                        etiVarDeRegC.setText(cadena.toString());
                    break;
                    case 20:

        etiNivDeDisB.setText(Integer.toString(valor * 11 / 127));
                        etiNivDeDisC.setText(cadena.toString());
                    break;
                    case 21:

        etiNivDeDenB.setText(Integer.toString((valor * maxDeDensidad /
127) + 1));
                        etiNivDeDenC.setText(cadena.toString());
                    break;
```

```
                    case 22:

        etiVarDeIntB.setText(Integer.toString(valor));
                                etiVarDeIntC.setText(cadena.toString());
                    break;
                    case 23:
                                etiNivDeDurB.setText(Double.toString(
new Double(new Double(100*(Math.min((varDur * durMaxDeNota / 127) +
durMinDeNota, durMaxDeNota))).intValue()).doubleValue() /100));
                                etiNivDeDurC.setText(cadena.toString());
                    break;
                    }
            }

        //Métodos de ActionListener, encargado de definir que el botón
"prende" realice el método inicio al ser presionado
        public void actionPerformed(ActionEvent e)
        {
            Object source = e.getSource();
            if (source == botonPaPrender) inicio();

        }

        // Metodos de recepción MIDI, necesario definir todos al implementar
MidiListener
        public void handleNoteOn(double timeStamp, int channel, int pitch,
int velocity)
        {
            altura = pitch;
```

```
            intensidad = velocity;

            alTocarNota();
        }

        public void handleNoteOff(double timeStamp, int channel, int pitch,
int velocity)
        {
        }

        public void handlePolyphonicAftertouch(double timeStamp, int
channel, int pitch, int pressure)
        {
            JMSL.midi.sendMessage(160 + (channel - 1), pitch,
pressure);
        }

        public void handleControlChange(double timeStamp, int channel, int
id, int valor)
        {
            switch(id)
            {
                case 16:
                    varRep = valor;
                    cambiaGraficos(id, valor);
                break;
                case 17:
                    varTem = valor;
```

```
            cambiaGraficos(id, valor);
        break;
        case 18:
            varRit = valor;
            cambiaGraficos(id, valor);
        break;
        case 19:
            varReg = valor;
            cambiaGraficos(id, valor);
        break;
        case 20:
            varDis = valor;
            cambiaGraficos(id, valor);
        break;
        case 21:
            varDen = valor;
            cambiaGraficos(id, valor);
        break;
        case 22:
            varInt = valor;
            cambiaGraficos(id, valor);
        break;
        case 23:
            varDur = valor;
            cambiaGraficos(id, valor);
        break;
        default:
```

```
                    JMSL.midi.control(channel, id, valor);
                }
        }

        public void handleProgramChange(double timeStamp, int channel, int
program)
        {
                JMSL.midi.programChange(channel, program);
        }

        public void handleChannelAftertouch(double timeStamp, int channel,
int pressure)
        {
                JMSL.midi.sendMessage(208 + (channel - 1), pressure, 0);
        }

        public void handlePitchBend(double timeStamp, int channel, int lsb,
int msb)
        {
                JMSL.midi.sendMessage(224 + (channel - 1), lsb, msb);
        }

        // El método al TocarNota se invoca cada que es recibido un mensaje
de NoteOn. Este método genera una o varias NotasCompuetas
        //según el valor de densidad.
        public void alTocarNota()
        {
                NotaCompuesta miMJ = new NotaCompuesta();
```

```
            int valorDeDensidad = (varDen * maxDeDensidad / 127) + 1;
            for (int i = 0; i < valorDeDensidad; i++)
            {
                NotaBase miNota = new NotaBase();
                miMJ.addRepeatPlayable(miNota);
            }
            miMJ.launch(JMSL.now());
        }

        // La classe NotaCompuesta genera según los valores de
NumeroDeRepetición, Tempo de Repetición y Variación rítmica una
estructrura musical que será
        // utilizada por alTocarNota
        public class NotaCompuesta extends MusicJob
        {
            double tmpDeRep;
            double nivVarRi;

            NotaCompuesta()
            {
                tmpDeRep = Math.min((varTem *
tiempoMaximoDeRepeticion / 127) + tiempoMinimoDeRepeticion,
tiempoMaximoDeRepeticion);
                nivVarRi = ((varRit * (unidadDeVariacionRitmica -
1) / 127) + 1);
                setRepeatPause(tmpDeRep);
                setRepeats(numeroDeRepeticiones(0));
```

```
        }

        public double repeat(double playTime) throws
InterruptedException
        {
            setRepeatPause(variacionRitmica(0));
            return (playTime);
        }

        public int numeroDeRepeticiones(int nr)
        {
            nr = (varRep * numMaxDeRepeticiones / 127) + 1;
            return (nr);
        }

        public double variacionRitmica(double barRit)
        {
            int volado;
            volado = JMSLRandom.choose(1);
            if (volado == 0) barRit = tmpDeRep *
JMSLRandom.choose(1.0, nivVarRi);
            if (volado == 1) barRit = tmpDeRep /
JMSLRandom.choose(1.0, nivVarRi);
            return (barRit);
        }
    }
```

```
//La clase NotaBase se encarga de generar una nota con canal, altura,
intensidad y duración específica según los valores que en ese momento tengan
//los controladores.
class NotaBase implements Playable
{
        int miAltura;
        int miIntensidad;
        int miCanal;
        double tiempoApaga;

        NotaBase()
        {
                miAltura = creadorDeAltura(altura);
                miIntensidad = creadorDeIntensidad(intensidad);
                miCanal = creadorDeCanal(0);
                tiempoApaga = creadorDeTiempoApaga(0);
        }

        public double play(double playTime, Composable thing)
        {
                JMSL.midi.noteOn(miCanal, miAltura,
miIntensidad);
                JMSL.midi.noteOff(JMSL.now() + tiempoApaga,
miCanal, miAltura);
                return (playTime);
        }

        int nivelDeDisonancia(int disonancia)
```

```
        {
                int valorDeDisonancia = varDis * 11 / 127;
                disonancia =
dis[JMSLRandom.choose(valorDeDisonancia + 1)];
                return(disonancia);
        }

        int nivelDeRegistro(int registro)
        {
                registro = registro + ((
JMSLRandom.choosePlusMinus(varReg / 21)*12));
                while (registro < 0 || registro > 127) registro =
registro + (( JMSLRandom.choosePlusMinus(varReg / 21)*12));
                return(registro);
        }

        int creadorDeAltura(int altura)
        {
                int miAltura = nivelDeDisonancia(0) +
nivelDeRegistro(altura);
                while (miAltura < 0 || miAltura > 127) miAltura =
nivelDeDisonancia(0) + nivelDeRegistro(altura);
                altura = miAltura;
                return(altura);
        }

        int creadorDeIntensidad(int intensidad)
        {
```

```
                    intensidad = intensidad +
JMSLRandom.choosePlusMinus(varInt);
                    while (intensidad < 1 || intensidad > 127) intensidad
= intensidad + JMSLRandom.choosePlusMinus(varInt);
                    return(intensidad);
                }

                int creadorDeCanal(int canal)
                {
                    canal = JMSLRandom.choose(1, numDeCanales +
1);
                    return(canal);
                }

                double creadorDeTiempoApaga(double tiempoApaga)
                {
                    tiempoApaga = Math.min( (varDur * durMaxDeNota
/ 127) + durMinDeNota, durMaxDeNota);
                    return(tiempoApaga);
                }
        }
```

2.2 El Controlador MIDI GAB

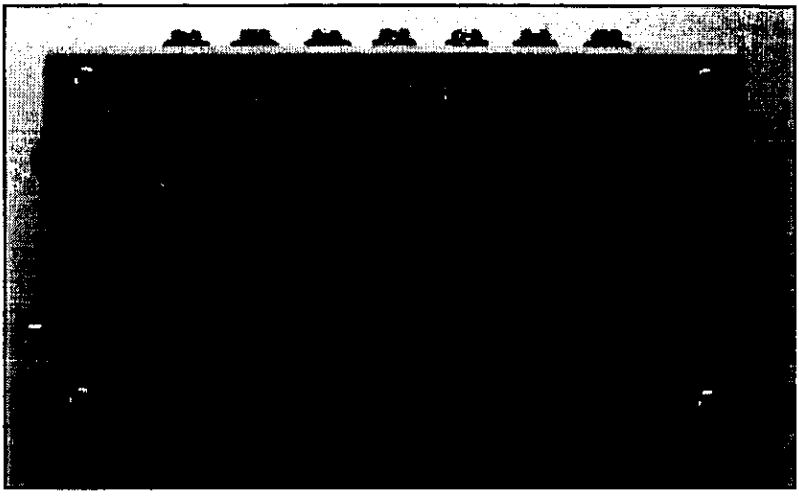


Figura 3

El *controlador MIDI GAB* (Figura 3) es un dispositivo electrónico que permite modificar los parámetros del *procesador de datos* de una manera sencilla mientras se está tocando el piano. Este controlador consta de ocho perillas y siete conectores para pedales MIDI comerciales[4]. La información MIDI que el controlador manda en su puerto de salida MIDI OUT es la siguiente:

4 La versión actual del controlador tiene deshabilitados los conectores de pedal pues no son utilizados por el procesador.

Control	Número de MIDI Control Change	Control	Número de MIDI Control Change
Perilla 1	16 (0-127)	Pedal 1 (9)	24 (0-127)
Perilla 2	17 (0-127)	Pedal 2 (10)	25 (0-127)
Perilla 3	18 (0-127)	Pedal 3 (11)	26 (0-127)
Perilla 4	19 (0-127)	Pedal 4 (12)	27 (0-127)
Perilla 5	20 (0-127)	Pedal 5 (13)	28 (0-127)
Perilla 6	21 (0-127)	Pedal 6 (14)	29 (0-127)
Perilla 7	22 (0-127)	Pedal 7 (15)	30 (0-127)
Perilla 8	23 (0-127)	-----	-----

Tabla 14

El empleo de este equipo es totalmente intuitivo y no requiere de ninguna explicación. Este equipo es alimentado por una pila de 9 V, tiene un interruptor para prenderlo y apagarlo, un led para ver su estado de actividad y un puerto DB9 para conectarlo a una computadora personal en caso de que se quiera modificar el programa residente en el microcontrolador.

La parte más importante para el funcionamiento de este equipo es el microcontrolador BasicStamp. Este circuito integrado es el encargado de convertir los valores analógicos de los potenciómetros en información digital codificada de acuerdo a los lineamientos del protocolo MIDI.

2.2.1 Construcción del Controlador MIDI

No se requiere de un gran conocimiento en electrónica ni de un equipo muy sofisticado para poder hacer un *controlador MIDI GAB*. Las ventajas de diseñar uno mismo este tipo de aparatos radica principalmente en lo económico que pueden resultar.

Los materiales requeridos para la construcción de un Controlador MIDI igual la *controlador MIDI GAB* se enlistan en la Tabla 15. La Figura 4 muestra todos los materiales requeridos y la Figura 5 presenta el diagrama del circuito.

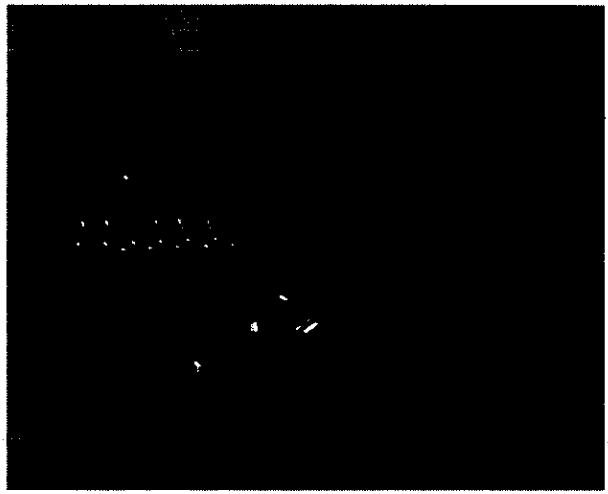
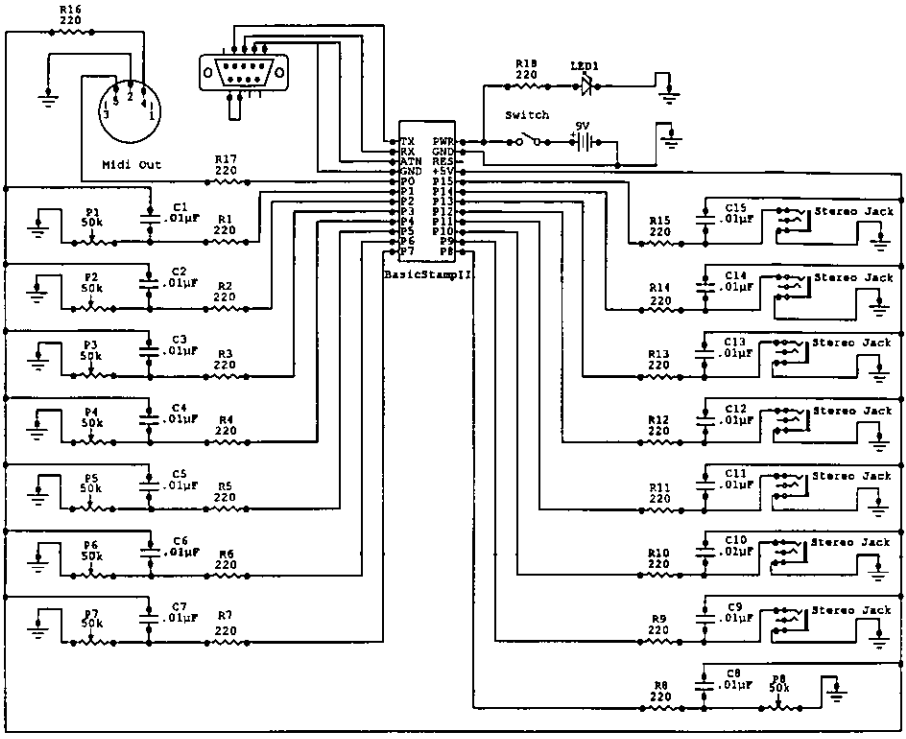


Figura 4

Lista de materiales	
1	Gabinete de tamaño adecuado
1	Placa de cobre lisa o previamente perforada de 10 x 10 Cm
1	Microprocesador BasicStampII de Páralax
1	Base para circuito integrado de 24 pines
1	pila de 9 V.
1	Soporte para pila de 9 V
1	conector para pila de 9 v
1	Led con resistencia integrada
1	interruptor de paso
1	conector DB9 hembra para chasis
1	conector DIN hembra de 5 pines para chasis
8	potenciómetros lineales de 50 KΩ
8	perillas para potenciómetro
7	conectores para plug estéreo
17	resistencias de 220Ω
15	capacitores de .01μF
***	Tornillos y tuercas de diferentes tamaños

Tabla 15




```
Digrama del Controlador MIDI GAB
Versión: 1.0
Autor: Hugo Solís García
Fecha: 25 de octubre del 2001
El circuito permite a un microcontrolador BasicStampII convertir el valor de hasta 15
potenciometros en información MIDI y perimte modificar el programa del microcontrolador
por medio del puerto DB9. El circuito puede ser alimentado con una pila de 9 V e
incluye un interruptor para prender y apagar y un led para ver si está prendido o
apagado. El circuito incluye 8 potenciómetros y 7 jacks para pedales u otros tipos de
controladores.
```

Figura 5

Para evitar falsos contactos y posibles cortos se recomienda diseñar un pequeño circuito en la placa de cobre. Existen varios procesos caseros que permiten el diseño de esta clase de circuitos.

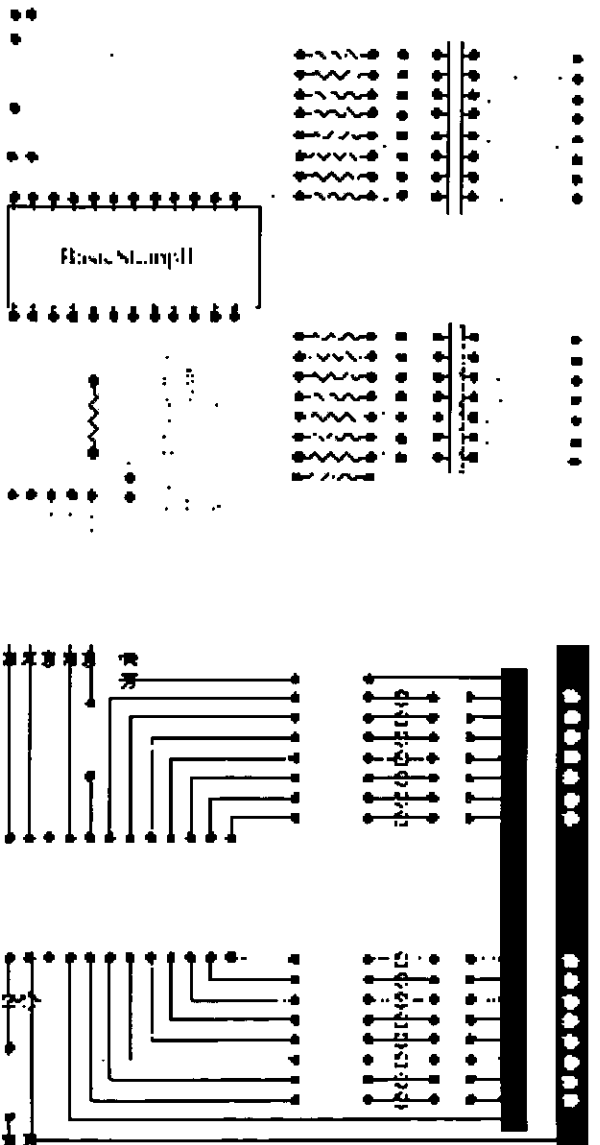


Figura 6

La figura 6 muestra en acetato (para facilitar el copiado) los diagramas en tamaño real para las caras superior e inferior para la placa de cobre de 10 * 10 cm. Las Figuras 7, 8 y 9 muestran la cara superior e inferior de la placa de cobre así como el controlador terminado.

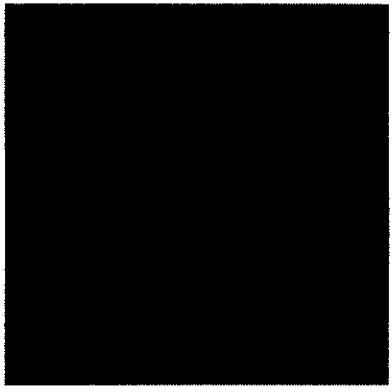


Figura 7

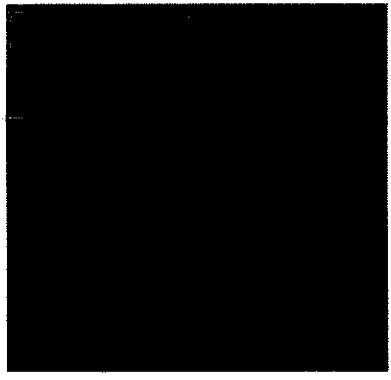


Figura 8

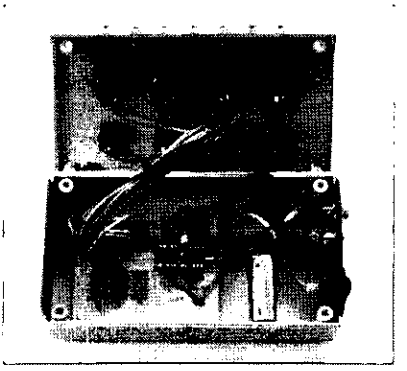


Figura 9

2.2.2 Que son los microcontroladores

Un microcontrolador es un circuito integrado (chip) capaz de almacenar y ejecutar rutinas o programas. Al igual que las computadoras, los microcontroladores tienen una Unidad Central de Procesamiento (CPU), memoria de acceso aleatorio (RAM), memoria de lectura únicamente (ROM), puertos de entrada y salida (I/O), puertos seriales y paralelos, relojes internos y en ocasiones otro tipo de periféricos como convertidores analógicos-digitales (A/D) o digitales-analógicos (D/A). La diferencia entre las computadoras y los microcontroladores radica en que las primeras están pensada para interactuar con los seres humanos (por medio de la pantalla, el teclado, la impresora, etc.) mientras que los segundos están diseñados para interactuar con otros equipos electrónicos. De hecho, los microcontroladores no son sino pequeñas computadoras que son utilizadas en todo tipo de equipos electrónicos: relojes, juguetes, semáforos, elevadores, etc.

La habilidad de los microcontroladores para ejecutar programas con fines específicos los hacen extremadamente versátiles. Uno puede programar un microcontrolador para tomar

decisiones lógicas de acuerdo a situaciones y selecciones predeterminadas que involucran normalmente a sus puertos de entrada y salida. La habilidad de estos equipos para ejecutar funciones matemáticas y lógicas les permite simular sofisticados circuitos electrónicos o lógicos, todo en un sólo componente electrónico de bajo costo.

2.2.3 El microcontrolador BasicStamp

Existen en el mercado una gran cantidad de microcontroladores con características, usos y capacidades diversas. La mayoría de los microcontroladores son programados en lenguaje ensamblador y por tal motivo se pueden volver complicados de programar. Sin embargo, existe un microcontrolador diseñado por la compañía Parallax que no es programado en ensamblador. El BasicStamp es un microcontrolador diseñado para ser utilizado por aficionados a la electrónica y es programado en un lenguaje muy sencillo llamado Pbasic. El BasicStamp presenta ciertas desventajas con respecto a otros microcontroladores como el PIC o el HC11 (poca velocidad y capacidad de memoria limitada); sin embargo el BasicStamp es una excelente opción para diseñar equipos en donde se requieran sencillas funciones MIDI.

2.2.4 Explicación y transcripción del código fuente

En la primera parte del programa se definen las constantes requeridas. Es importante recalcar que el valor 32780 de la constante midibaudmode corresponde al valor que el BasicStamp necesita para realizar la transmisión a la velocidad y con las características que el protocolo MIDI determina: 31.25 (+/- 1%)

Kbaud, asincrónico, con un bit de inicio, ocho bits de datos y un bit final.

El valor binario de la constante *controlador* %10110000 corresponde al status byte asignado al dato MIDI ControlChange canal 1.

Después de definir las constantes, se definen las variables necesarias y se declaran como salida todos los puertos del BasicStamp para evitar daños en el circuito.

La rutina *principal* se encarga de monitorear constantemente el valor de los potenciómetros por medio de la instrucción RCTime. Si el valor de alguno de ellos cambia, la rutina *mandaDato* se encarga de construir el dato MIDI correspondiente y transmitirlo a la velocidad requerida.

A continuación se transcribe íntegro y sin modificaciones el código fuente del programa:

```
' GAB controlador MIDI
' Versión 1.0
' Autor Hugo Solís
' Fecha 25 de octubre del 2001
' Este programa hace posible que el BasicStampII lea hasta 15 potenciomtros
' y convierta sus valor en valores MIDI ControlChange 16 - 23 canal 1
' el programa esta diseñado para ser utilizado en el BasicStamp del GAB
' controlador MIDI.
' Para detalles de funcionamiento ver la tesis "Gab 1.0, sistema de
reinterpretación
' electrónica para pianos acústicos" de Hugo Solís
```

```
' Declaración de constantes

midibaudmode con 32780 '31.25kb, 8n1, non-inverted, open collector
controlador con %10110000 'numero binario correspondiente al status byte Control Change canal 1
pinDeSalida con 0
defase con 15
cantidadDeControladores con 8

' Declaración de variables
resultado var word
valor var byte
valorAnterior var byte(cantidadDeControladores)
numeroDePin var nib

' Para seguridad del Microcontrolador, todos los pines no utilizados
' deben de declararse como puertos de salida
DIRS = %1111111111111111 'all outs
OUTS = %0000000000000000 'all low

principal:
        for numeroDePin = 1 to cantidadDeControladores
                high numeroDePin
                pause 1
                RCTIME numeroDePin, 1, resultado
                valor = ((resultado Max 414)* 127 / 414)
```

```
            if valor = valorAnterior(numeroDePin - 1) then noMandaDato
            gosub mandaDato
            noMandaDato:
            valorAnterior(numeroDePin - 1) = valor
        next
        goto principal


mandaDato
'       debug cls, dec numeroDePin, dec valor Max 127 ' quitar la coma inicial de la linea para ver valores en pantalla
        serout pinDeSalida, midibaudmode, [controlador, numeroDePin + defase, valor]
        return
```

3 Conclusiones

El sistema GAB es una herramienta interpretativa, planeada y diseñada por un músico. Si bien es cierto que para su creación fue necesario un trabajo interdisciplinario que involucró aspectos de computación y electrónica, la preocupación musical siempre estuvo presente. De nada serviría todo el trabajo realizado si el resultado musical no fuese interesante. Lo valioso que pueda llegar a ser este proyecto dependerá únicamente de la forma en que sea utilizado. La herramienta esta hecha, viene ahora el proceso de trabajar con ella, de aprovechar sus recursos y conocer sus limitaciones; también de las limitaciones se puede sacar provecho. Las siguientes versiones del sistema GAB y las mejorías que se le vayan haciendo serán únicamente el resultado de trabajar con esta primera versión.

Vivimos una época en donde la tecnología se ha hecho presente en todos los ámbitos del quehacer humano: el arte y la música no son la excepción. Todos estamos en mayor o menor medida, conscientes o inconscientes, inmersos y relacionados con el mundo de las computadoras y los circuitos electrónicos. Sin embargo, mientras más evoluciona la tecnología más difícil es entender cómo funciona. El sistema GAB es un pretexto y una invitación a no hacer de los aparatos y programas musicales "cajas negras" que compramos con precio en dólares y que con sus particulares recursos definen incluso nuestros proyectos musicales. El sistema GAB es sin duda mucho más limitado que los equipos comerciales; sin embargo, tiene una característica peculiar: sé qué tiene adentro, hace lo que quise que hiciera y, sobre todo, puedo decir con mucho gusto: Yo lo hice.

4 Apéndice I: Algunos libros y sitios en Internet relacionados con el sistema GAB y con la música por computadora.

- Anderton, Craig. *MIDI for Musicians*. USA: Amsco Publications, 1986.
- Anderton, Craig, Bob Moses, y Greg Bartlett. *Digital Projects for Musicians*. USA: Amsco Publications, 1994.
- Bencina, Ross. http://www.audiomulch.com/midipic
- Boulanger, Richard. The Csound Book, Perspectives in software synthesis, sound design, signal processing, and programming. USA: MIT Press, 2000.
- Burk, Phil. http://www.softsynth.com
- Cohen, Alan A. *Audio Technology Fundamentals*. USA: Howard W. Sams & Company, 1989.
- Chan, Mark C., Steven W. Griffith, y Anthony F. Iasi. *1001 Java Programmer's Tips*. USA: JAMSA Press, 1997.
- Chadbe, Joel. Electric Sound, the past and promise of electronic music. USA: Prentice Hall, 1997.
- Chamberling, Hal. *Music Applications of Microprocessors*. USA: Hayden Books, 1985.
- Didkovsky, Nick, y Phil Burk. http://www.algomusic.com/jmsl/index.html
- Dodge, Charles y Thomas A. Jerse. *Computer Music*. USA: Schirmer Books, 1997.

- Eckel, Bruce. *Thinking in JAVA*. USA: Prentice Hall PTR, 2000.
- Edwards, Scoot. Programming and Customizing the Basic Stamp Computer. USA: McGraw-Hill, 1998.
- Horn, Delton T. *Basic Electronic Theory*. USA: McGraw-Hill, 1994.
- International MIDI Association. *MIDI 1.0 Detailed Specification*. USA: 1985
- Iovine, John. *PIC Microcontroller Project Book*. USA: McGraw-Hill, 2000.
- Kühnel, Claus y Klaus Zahnert. *BasicStamp*. USA: Newnes, 2000.
- Laboratorio de Investigación de Música por Computadora del Centro Nacional de Creación Musical GRAME. http://www.grame.fr/MidiShare
- Mann, Jeff. http://www.interaccess.org/arg/arg-knowledge/MIDI.BS2
- Messick, Paul. Maximum MIDI, Music Applications in C++. USA: Manning, 1998.
- Moore, F. Richard. *Elements of Computer Music*. USA: PTR Prentice Hall, 1990.
- Otsuka, Akira y Akihiko Nakajima. *MIDI Basics*. Japón: Amsco Publications, 1987.
- ParallaxInc. *Basic Stamp Programming Manual 1.8*. http://www.parallaxinc.com

5 Bibliografía

- Chan, Mark C., Steven W. Griffith, y Anthony F. Iasi. *1001 Java Programmer's Tips*. USA: JAMSA Press, 1997.
- Didkovsky, Nick y Phil Burk. http://www.algomusic.com/jmsl/index.html
- Didkovsky, Nick. Resúmenes para la clase JAVA MUSIC de la Universidad de New York. USA: Documento de trabajo, 2000.
- Edwards, Scoot. Programming and Customizing the Basic Stamp Computer. USA: McGraw-Hill, 1998.
- Iovine, John. *PIC Microcontroller Project Book*. USA: McGraw-Hill, 2000.
- Kühnel, Claus, y Klau Zahnert. *BasicStamp*. USA: Newnes, 2000.
- Laboratorio de Investigación de Música por Computadora del Centro Nacional de Creación Musical GRAME. http://www.grame.fr/MidiShare
- Mann, Jeff. http://www.interaccess.org/arg/arg-knowledge/MIDI.BS2
- ParallaxInc. Basic Stamp Programming Manual 1.8. http://www.parallaxinc.com
- Rumsey, Francis. *Midi Systems & control*. USA: Focal Press, 1994.

- Peter, Manning. *Electronic & Computer Music*. USA: Oxford University Press, 1985.
- Predko, Myke. Programing and Customizing the Pic Microcontroller. USA: McGraw-Hill, 1998.
- Rumsey, Francis. *Midi Systems & control*. USA: Focal Press, 1994.
- Williams, David Brian, y Peter Richard Webster. *Experiencing Music Technology*. USA: Schirmer Books, 1996.
- Winkler, Todd. *Composing Interactive Music*. USA: MIT Press, 1998.