

Manual de la librería “Sonidos.h”

1 Introducción

La librería de ‘Sonidos.h’ tiene como excusa la necesidad que sentía para la creación del ANILLO III, de que el jugador pudiese hacerse una idea de las cosas que estaban pululando por el sombrío bosque pantanoso que es la base de dicho proyecto (al menos, de proyecto original que permanecía olvidado en un cuaderno de notas desde hacía años); pero lo cierto es que, finalmente, la librería de manejo de sonidos me ha servido como una zambullida mucho más profunda en las turbulentas tripas de la librería InformatE de lo que me había permitido mi primera aproximación al mundo Inform en Castellano, *Casi Muerto* (<http://www.arrakis.es/~meliton/casi.htm>).

Y, ¿para qué sirve esta librería? Pues en principio se trata de una idea muy sencilla, de generar, transmitir y escuchar ‘sonidos’. Es decir, ampliar y automatizar lo más posible el tratamiento de otro de los sentidos que puede tener el personaje llevado por el jugador, el oído; y también hacer lo mismo si es posible para los PNJs que pululen por el juego.

El ejemplo más sencillo es simplemente que un PNJ diga algo. Típicamente esto se ha resuelto en las aventuras mediante la impresión de un mensaje de la forma:

```
"Pepito, el vendedor dice: ~Hola, ¡bienvenido a mi humilde tienda!~";
```

Y por supuesto antes de decir la frase hay que verificar si el jugador está presente, puede desencadenar este mensaje y si realmente puede ver a Pepito. Cualquier otro caso similar, como que el jugador, oculto tras un biombo, escucha como Pepito dice esta frase a alguien, debe ser tratado en el código como un caso especial, por lo que la complejidad del tratamiento de este tipo de situaciones se dispara en cuanto se quieren poner unas pocas. Con la librería de sonidos que se explica en este manual, bastará (en principio) con declarar Pepito como un objeto que puede hablar, mediante:

```
Object Pepito class ObjetoHablante,  
with  
....
```

e incluir en alguna parte del código:

```
Pepito.habla( "Hola, ¡bienvenido a mi humilde tienda!" );
```

la librería se encargará de generar los mensajes apropiados según si el jugador ve a Pepito o no lo ve, o si está lejos o cerca, o detrás de un biombo, o si, aún sin verlo, puede reconocer la voz de Pepito a distancia. De esta forma crear la situación en la que el jugador puede escuchar una conversación entre dos PNJs detrás de una pared no demasiado gruesa, o incluso guiarse en un laberinto por el sonido de una puerta que cruje, resultan considerablemente más sencillas de implementar.

Además de objetos ‘hablantes’ el programador podrá generar ‘sonidos’ desde cualquier origen sin que sea hablante o incluso asociar un sonido de fondo (un ruido) a un objeto de tal forma que o bien se agregue la presencia del sonido a la descripción de las localidades desde las que se oiga o simplemente se pueda percibir ese sonido bajo un comando de ‘escuchar’ por parte del jugador, o ambas cosas simultáneamente.

2 Inicios básicos: ¿Cómo usarla?

Para poder usar la librería de sonidos es necesario añadir dos includes en tu código de InformatE. El primero debe estar antes de el include de Acciones, consiste en la sustitución de la rutina inicial de la librería de la acción ‘escuchar’:

```
include "NueEsc";  
include "Acciones";
```

el otro include debe estar detrás del include de la gramática. Hay una restricción más, la librería usa (como PNJMóvil) la clase ‘Lugar’ para indicar las localizaciones por dónde se propaga el sonido. La librería aunque se basó en un principio en PNJMóvil, es completamente independiente de ella, pero dado que ambas usan ‘Lugar’ es necesario que si se van a incluir ambas se incluya PNJMóvil antes que Sonidos. Así que finalmente nos quedaría:

```
include "Gramatica";  
include "PNJMovil"; ! Si es que se va a usar  
include "Sonidos";
```

Desde ese punto y en el resto de tu código ya podrás incluir objetos hablantes, oyentes, fuentes de ruido, clases de sonido, etc, etc... como vas a poder ver en las siguientes secciones de este manual.

3 Intensidad Sonora y Propagación

La librería usa el concepto de intensidad sonora para averiguar hasta dónde se oye un sonido y cómo se oye. La intensidad sonora no es más que un número y se usan las siguientes variables globales para guardar los diversos umbrales de percepción de intensidad sonora:

```
Global nivelAudible = 2;  
Global nivelSusurro = 4;  
Global nivelMurmullo = 6;  
Global nivelNormal = 8;  
Global nivelAlto = 10;  
Global nivelMuyAlto = 12;
```

Esta ‘escala’ puede imaginarse como niveles de intensidad sonora en algún múltiplo de decibelios. Por supuesto estas variables se pueden cambiar en cualquier punto del programar para simular cualquier efecto como hacer que todo el mundo se vuelva hipersensible a los sonidos, o sordos, o simplemente ajustar al gusto del programador la percepción de los sonidos.

El nivelAudible, marca en la librería normal el nivel mínimo de audición, es decir, con el tipo de voz normal de la librería y sin cambiar nada, el jugador no percibirá ningún sonido que le llegue con intensidad sonora 1 ó 2, sin embargo ObjetosOyentes recibirán el sonido con lo que el programador podrá determinar si tienen un oído mucho más fino que el jugador. Como ejemplo de uso, asignando un valor por encima de 12, por ejemplo, 30 a la variable nivelAudible, el jugador será completamente sordo a todos los sonidos.

La librería puede indicar también la intensidad sonora (como se verá en el apartado de diseño de voces) con la sensación de ‘lejanía’ de la voz, un sonido se considerará (sin cambios por parte del programador, en esta librería casi todo es cambiabile) ‘lejano’ si tiene una intensidad sonora por debajo de ‘nivelMurmullo’ y se considerará ‘muy lejano’ por debajo de ‘nivelSusurro’.

El otro conjunto de variables que controlan la percepción de sonidos en la aventura son las variables que marcan lo que amortigua el sonido las distintas clases de propagación:

```
Global costePorObstaculo = 4;!! Por pared o puerta cerrada interpuesta
Global costePorHabitacion = 2;!! Por puerta abierta
Global costePorContenedor = 1;!! Por contenedor abierto
Global costePorContenedorCerrado = 2;!! Por contenedor cerrado
```

El significado es exactamente el indicado en los comentarios:

- **costePorObstaculo:** indica lo que se amortigua el sonido por pasar por una puerta cerrada, o bien, por una pared (ser verá en la sección de propagación de sonido, más adelante la propiedad 'cercanoA').
- **costePorHabitacion:** indica lo que se amortigua el sonido por pasar de una habitación a otra sin que haya nada interpuesto. Por ejemplo, si tu aventura contiene localidades que representan gran cantidad de espacio abierto, deberías subir bastante el valor de esta variable.
- **costePorContenedor:** indica lo que se amortigua el sonido salir de un contenedor abierto, en la librería se ha dejado el valor de 1, pero muchos probadores de la librería opinaban que debería valer cero, puedes cambiarlo a cero sin que signifique ningún problema.
- **costePorContenedorCerrado:** indica lo que se amortigua el sonido por salir por las tapas de un contenedor cerrado.

4 Mapa sonoro de tu aventura

La librería utiliza el mapeado normal para llevar el sonido de un lado a otro, con una importante restricción: **la librería sólo generará sonidos desde localidades de la clase Lugar y sólo los propagará por la clase Lugar**. Esto es una ventaja más que una desventaja. Por ejemplo, en la típica aventura en la que muchas localidades pertenecen a una misma zona 'cercana' (un edificio, un boque pequeño, un laberinto), pero que se unen en 'localidades de paso' (como un camino, unas praderas que rodean el edificio, etc...), si se crean como de la clase 'Lugar' las zonas que permanecen cercanas y como no 'Lugar' las de paso, los sonidos no pasarán de una zona a otra. En cualquier caso lo cierto es que la librería está pensada fundamentalmente para aventuras en las que el 'tamaño' de cada localización sea bastante homogéneo, de forma que todas sean de la clase 'Lugar' y el sonido se propague de forma similar por todas ellas.

Pero el mapeado 'normal' (es decir, los caminos que pueden seguir el jugador y los PNJs) no es suficiente para tener un mapa sonoro completo en todos los casos. Pueden existir, por ejemplo, localidades separadas por una simple plancha de madera que impide el paso del jugador pero no del sonido, o al menos no completamente. Para esto existe la propiedad 'cercanoA'.

Si se desea que una habitación propague el sonido a otra con la que no está comunicada de forma normal, basta con añadir a esta habitación la ristra de 'dirección' y 'habitación' de las que están cercanas desde el punto de vista de sonido. Ejemplo:

```
Lugar Pasillo "pasillo"
with
    al_n Sala,
    cercanoA obj_arriba Torre;
```

```

Lugar Sala "sala"
with
    al_s Pasillo,
    arriba Mirador;

Lugar Mirador "mirador"
with
    abajo Sala,
    al_s Torre;

Lugar Torre "torre"
with
    al_n Mirador,
    cercanoA obj_abajo Pasillo;

```

En este sencillo mapeado para que el jugador llegue desde el Pasillo hasta la Torre tendría que pasar por la Sala y el Mirador; sin embargo, la Torre está justo encima del Pasillo y gracias a la presencia de 'cercanoA' en ambos 'Lugares' los sonidos del Pasillo se oirán en la Torre a través del suelo, y los de la Torre en el Pasillo a través del techo.

La ristra de valores de 'cercanoA' siempre deben ser pares, el primero de cada par el objeto-dirección y el segundo la habitación cercana, si esto no se cumple los resultados pueden ser completamente extravagante.

Sin embargo, tanto el primero como el segundo de cada par pueden ser sustituidos por una función. Por ejemplo, podemos imaginar una habitación móvil que se mueve sobre unos raíles ya sobre el Comedor o sobre el Pasillo. Digamos que un atributo de la localidad llamada 'estado' indica si está sobre el Comedor (si vale 0) o sobre el Pasillo (si vale 1), la forma correcta de esta extraña habitación sería:

```

Lugar HabitacionMovil "locomotora del techo"
with
    estado 0,
    cercanoA obj_abajo
        [;
            if (self.estado == 0) return Comedor;
            else return Pasillo;
        ];

Lugar Comedor "comedor"
with
    cercanoA
        [;
            if (HabitacionMovil.estado == 0) return obj_arriba;
            else return 0;
        ]
    HabitacionMovil;

```

```

Lugar Pasillo "pasillo"
with
    cercanoA
        [;
            if (HabitacionMovil.estado == 0) return 0;
            else return obj_arriba;
        ]
    HabitacionMovil;

```

5 Objetos Hablantes

El uso más simple que podemos darle a la librería es poner unos objetos que generen sonidos en algunas ocasiones, para ello basta con declarar un objeto como de clase ObjetoHablante con esto el objeto adquiere dos atributos:

- **voz:** indica la clase de voz que usa el objeto para hablar, si no se indica nada se usa una clase de voz implementada en la librería llamada 'tipoPlano' que para gran parte de las aplicaciones es más que suficiente.
- **habla:** es el procedimiento que debe llamarse para que el ObjetoHablante genere sonidos y puede tener hasta tres parámetros:
 1. Una frase entrecomillada a decir, si no se incluye ninguna frase o se pone un 0 en este parámetro, el objeto emitirá un sonido SIN texto, sin decir nada: como una campanilla, una gota de agua, o lo que sea dependiendo de la 'voz' que se use.
 2. Un nivel de intensidad, si no se incluye el parámetro o se pone un 0 se emitirá el sonido a 'nivelNormal' (a lo que indique esa variable global)
 3. Un modo. El concepto de modo, no se usa en la librería pero sí que se transmite a todos los oyentes PNJ. El programador puede usar este valor para incluir cualquier cosa que crea conveniente. Se puede usar para saber si es un grito, un chillido, o cualquier cosa mucho más compleja que se te pueda ocurrir. Como ejemplo muy extravagante se podría crear un sistema de intercambio de mensajes 'mentales' entre los PNJs que el jugador no escuchase utilizando este valor de modo y una clase de 'voz' un poco retocada, como se verá más adelante en la sección de tratamiento avanzado de voces.

Objeto hablante muy sencillo, puede ser:

```

ObjetoHablante LoroPegadoALaJaula "loro" Jaula
with
    nombre `loro`,
    daemon
        [;
            if (random(100) <= 33)
                self.habla( "Quiero la llave!");
            else if (random(100)<=50)
                self.habla("Libertad, quiero libertad!");
            else
                self.habla("No quiero vivir sin tí, Amorrrrr!");
        ];

```

Este loro, si es activado en el Inicializar del juego con un ArrancarDaemon cada turno dirá una chorrada, y esta chorrada no sólo se oirá estando junto al loro, sino a distancia o si, por ejemplo, la Jaula está cubierta con una manta.

Si se desea algo más interesante, como un loro que vaya volando por ahí, será necesario incluir algo como PNJMovil. Por ejemplo si se desea usar PNJMovil y suponiendo que hay una forma de liberar al loro de la jaula, el loro podría parecerse a:

```
PNJMovil Loro "loro" Jaula
class ObjetoHablante,
with
    nombre 'loro',
    accion_antes
    [;
        if ( parent(self) ~= Jaula )
            self.habla("Al fin libre, al fin!");
        else if (random(100) <= 33)
            self.habla( "Quiero la llave!");
        else if (random(100)<=50)
            self.habla("Libertad, quiero libertad!");
        else
            self.habla("No quiero vivir sin tí, Amorrrrr!");
    ];
```

6 Voces (tratamiento básico)

Las 'voces' son la forma en la que se emiten los sonidos y contienen por completo las rutinas, mensajes y valores necesarios para que el mensaje apropiado sea mostrado al jugador, los cálculos necesarios los hacen los métodos *diDistancia*, *diPotencia* y sobre todo el método *escuchado* de la voz en cuestión; pero no vamos a estudiar en este apartado el contenido de estos métodos ni la forma de crear unos diferentes porque es bastante complejo. Pero las voces no sólo se pueden cambiar cambiando estos métodos sino que tienen una buena cantidad de atributos que indican su comportamiento.

6.1 Creando nuevas voces

La forma más sencilla de personalizar una voz es simplemente 'instanciando' una nueva y dándole un nombre. Por ejemplo:

```
TipoDeSonido graznidoDeLoro "un graznido de loro";
```

ahora podemos cambiar a nuestro Loro por:

```
PNJMovil Loro "loro" Jaula
class ObjetoHablaante,
with
    nombre 'loro',
    voz graznidoDeLoro,
    accion_antes
    [;
    if ( parent(self) ~= Jaula )
    self.habla("Al fin libre, al fin!");
    else if (random(100) <= 33)
    self.habla( "Quiero la llave!");
    else if (random(100)<=50)
    self.habla("Libertad, quiero libertad!");
    else
    self.habla();
    ];
```

y ahora en lugar de decir 1 de cada tres veces lo del amor perdido, simplemente se imprimirá:

```
El loro emite un graznido de loro.
```

o bien, si el loro está hacia el norte algo similar a:

```
Oyes un graznido de loro proveniente del norte.
```

Como puede verse simplemente creando una nueva voz y dándole un nombre se pueden obtener ya variaciones interesantes. No sé, si ya ha quedado claro, pero si un objeto hablante llama a su método de hablar sin parámetro de texto o con un 0 en este parámetro, el jugador sólo verá el nombre corto de la voz usada. Sin embargo si pasa un texto el jugador verá algo similar a:

```
Oyes una voz proveniente del norte que dice: "Al fin libre, al fin!".
```

6.2 Cambiando algunos atributos de las voces

El siguiente parámetro que se podría querer cambiar de una voz podría ser el verbo con el que el hablante dice cosas. Esto se consigue cambiando el atributo *verboDecir*. Por ejemplo:

```
TipoDeSonido graznidoDeLoro "un graznido"
with
    verboDecir "grazna";
```

con esta nueva voz nuestro loro sacará mensajes como:

```
El loro grazna: "Al fin libre, al fin!".
```

Otro atributo sencillo de entender es *adjetivoDeVoz*, que por defecto en la librería contiene un cero (para que no se califique la voz), pero que puede contener un calificativo de la voz. Por ejemplo:

```
TipoDeSonido graznidoDeLoro "un graznido"
with
    adjetivoDeVoz "abatida",
    verboDecir "grazna";
```

con esta nueva voz nuestro loro sacará mensajes como:

```
El loro grazna con voz abatida: "Libertad, quiero libertad!".
```

Todos estos parámetros se pueden sustituir por rutinas que devuelvan diversas cadenas de mensajes, por ejemplo en nuestro caso sería mucho más apropiado crear la voz del loro como:

```
TipoDeSonido graznidoDeLoro "un graznido"
with
    adjetivoDeVoz
        [;
            if ( parent(Loro) == Jaula ) return "abatida";
            else return "jubilosa";
        ],
    verboDecir "grazna";
```

Otro atributo muy interesante de una voz es *reconocible* que indica si el jugador puede reconocer la voz sin ver la fuente. Por ejemplo, en nuestro caso el *graznidoDeLoro* puede empezar el juego con *reconocible* a false y cuando el jugador vea al loro por primera vez (al levantar la manta), se puede poner a true con lo que empezaría a ver mensajes de la forma:

```
Oyes la voz jubilosa del loro proveniente del norte que grazna: "Al fin
libre, al fin!".
```

Casi nada!!

6.3 Los atributos de la voz básica de la librería

La clase *TipoDeSonido* contiene los siguientes atributos que se pueden modificar:

- **reconocible:** por defecto a *false*, indica si el jugador puede reconocer el dueño de la voz aunque sea a distancia.
- **verboOir:** por defecto a “*Oyes*”, verbo usado cuando el jugador oye o escucha o lo que sea algo.
- **verboDecir:** por defecto a “*dice*”, verbo usado cuando el jugador oye que alguien dice una frase.
- **verboEmitir:** por defecto a “*emite*”, verbo usado cuando el jugador oye que algo genera un sonido sin frase. Hay un caso especial, si este atributo contiene 0, la librería usará lo que contenga ‘verboDecir’ y no incluirá el nombre corto del sonido. Esto es especialmente útil en voces de animales, para que los gatos maúllen, en lugar de ‘emitir un maullido’, además para los textos generados si usarPotencia está a true se escogerán los valores usados cuando se dice una frase, no los de sonidos sin frase.
- **adjetivoDeVoz:** por defecto a 0, calificativo de la voz usada, simplemente para darle ‘color’ a la voz.
- **usarDistancia:** por defecto a *false*, si se pone a true la librería calificará las voces según su intensidad en ‘lejanas’ o ‘muy lejanas’.
- **usarPotencia:** por defecto a *true*, si está en true la librería añadirá un mensaje al final indicando como de bien o mal se oye la voz según su intensidad.
- **usarDireccion:** por defecto a *true*, si se pone a true la librería indicará la dirección de la que parece provenir el sonido, si se pone a false siempre dirá que no se sabe de dónde procede. Ejemplo de uso: si en tu aventura hay algunas habitaciones ‘brumosas’ que impiden determinar con claridad de dónde vienen los sonidos te bastará con definir estas habitaciones como:

```
Class HabitacionBrumosa
    class Lugar;
y crear voces de la clase
Class vozQueNoSeVeEnBrumosa
    class TipoDeSonido
    with
    usarDireccion
    [;
        if (localizacion of class Brumosa ) return false;
        else return true;
    ];
```

- **adjetivoLejana:** por defecto a “*lejana*”, adjetivo usado para indicar que una voz es ‘lejana’ en femenino.
- **adjetivoLejano:** por defecto a “*lejano*”, adjetivo usado para indicar que una voz es ‘lejano’ en masculino.
- **adjetivoMuyLejana:** por defecto a “*muy lejana*”, adjetivo usado para indicar que una voz es ‘muy lejana’ en femenino.
- **adjetivoMuyLejano:** por defecto a “*muy lejano*”, adjetivo usado para indicar que una voz es ‘muy lejano’ en masculino.
- **noVozSusurro:** por defecto a “*que apenas se oye*”, frase para calificar la potencia de una voz que se ha usado sin texto a nivel de Susurro.

- **vozSusurro:** por defecto a “*pero apenas se entiende*”, frase para calificar la potencia de una voz que se ha usado con texto a nivel de Susurro.
- **noVozMurmullo:** por defecto a “*que suena muy bajito*”, frase para calificar la potencia de una voz que se ha usado sin texto a nivel de Murmullo.
- **vozMurmullo:** por defecto a “*pero suena como un susurro*”, frase para calificar la potencia de una voz que se ha usado con texto a nivel de Murmullo.
- **noVozNormal:** por defecto a 0, frase para calificar la potencia de una voz que se ha usado sin texto a nivel Normal.
- **vozNormal:** por defecto a 0, frase para calificar la potencia de una voz que se ha usado con texto a nivel Normal.
- **noVozAlto:** por defecto a “*que se oye muy claramente*”, frase para calificar la potencia de una voz que se ha usado sin texto a nivel de Alto.
- **vozAlto:** por defecto a “*con un torrente de Voz*”, frase para calificar la potencia de una voz que se ha usado con texto a nivel de Alto.
- **noVozMuyAlto:** por defecto a “*de tal intensidad que casi te deja sordo*”, frase para calificar la potencia de una voz que se ha usado sin texto por encima del nivel de Alto.
- **vozMuyAlto:** por defecto a “*con tal fuerza que casi te deja sordo*”, frase para calificar la potencia de una voz que se ha usado con texto por encima del nivel de Alto.

7 Generando sonidos desde no hablantes

Aunque realmente no era necesario se ha incluido la función *TocaDesde* que permite que cualquier objeto sin declararlo ‘hablante’ pueda emitir un sonido. La única ventaja puede residir en el caso de objetos cuya voz tenga que cambiar constantemente. Esta función recibe los parámetros:

- **tipo:** la voz que debe usarse para generar el sonido.
- **origen:** lo que genera el sonido.
- **frase:** el texto que dice la voz, como siempre se puede no incluir o poner un 0 para que simplemente se vea el ‘nombre’ de la voz.
- **intensidad:** si no se incluye o se pone a 0, se emitirá el sonido a nivel Normal, sino al que se incluya aquí.
- **modo:** como ya se ha dicho la librería no trata el modo pero lo propaga por si el programador desea usarlo para algo, por defecto su valor será 0.

De esta forma podríamos incluir la voz:

```
TipoDeSonido risaDemencial "una risa de loco";
```

y en alguna parte del código incluir:

```
TocarDesde( risaDemencial, Loro );
```

con lo que habremos conseguido que el loro de vez en cuando emita una risa de loco sin haber cambiado realmente su voz normal, el graznido.

8 La función *VozLejana*

Esta función genera una voz que se escucha por igual en todas partes de la aventura. Se trata de una especie de voz en off, o espectral, o global o lo que sea. Por ejemplo, podría salir de un sistema de megafonía que ‘esta_en’ todas partes. Esta función recibe los parámetros:

- **frase:** el texto que dice la voz, como siempre se puede no incluir o poner un 0 para que simplemente se vea el ‘nombre’ de la voz.
- **intensidad:** si no se incluye o se pone a 0, se emitirá el sonido a nivel Normal, sino al que se incluya aquí.
- **tipo:** la voz que debe usarse para generar el sonido, si es 0 o no se incluye se usará el tipoPlano básico de la librería.
- **origen:** lo que genera el sonido, si es 0 o no se incluye será de origen desconocido.
- **modo:** como ya se ha dicho la librería no trata el modo pero lo propaga por si el programador desea usarlo para algo, por defecto su valor será 0.

9 Objetos Oyentes

La utilidad de la librería sería un poco limitada si no se pudiese hacer reaccionar a los PNJs frente a los ‘sonidos’ que escuchasen. Para que un PNJ reciba los sonidos que circulan por la aventura hay que declararlo como ObjetoOyente. Al hacer esto el objeto habrá adquirido un método llamado *oído* a través del cual recibirá los sonidos. La implementación de este método que hace la librería es bastante tonto, simplemente si se recibe un sonido a un nivel Audible y cuyo origen no sea el propio objeto oyente, y si el jugador está presente para verlo, se dice que el objeto oyente ha oído algo.

El método *oído* recibe un parámetro:

- **_sonido:** que es una clase que contiene toda la información relevante sobre el sonido recibido, esta clase tiene los atributos:
 - o **clase:** el tipo de voz con el que se ha producido el sonido.
 - o **origen:** verdadera fuente de emisión del mensaje, 0 si es desconocida.
 - o **modo:** no es usado por la librería pero se recibe según se haya creado en la emisión.
 - o **intensidad:** el nivel sonoro con el que percibe el sonido.
 - o **sePuedeVer:** indica si el objeto oyente puede ver realmente o no la fuente del sonido.
 - o **seEscuchaEn:** si no se ve realmente la fuente, esto indica de dónde parece proceder, si contiene 0 no se sabe de dónde proviene.
 - o **texto:** el mensaje transportado por el sonido.

Veamos un ejemplo. Creamos una nueva voz, que emitiría un supuesto gato que rondase por nuestra aventura:

```
TipoDeSonido Maullido "un maullido bastante feroz";
```

El loro se asusta y se preocupa si oye el maullido incluso a un nivel subsónico, no audible por el jugador, con esto el loro nos quedaría como:

```
PNJMovil Loro "loro" Jaula
class ObjetoHablante ObjetoOyente,
with
    nombre 'loro',
    voz graznidoDeLoro,
    accion_antes
    [;
    if ( parent(self) ~= Jaula )
    self.habla("Al fin libre, al fin!");
    else if (random(100) <= 33)
    self.habla( "Quiero la llave!");
    else if (random(100)<=50)
    self.habla("Libertad, quiero libertad!");
    else
    self.habla();
    ],
    oido
    [ _sonido;
    if ( _sonido.clase == Maullido )
    self.habla( "Horror, me ha parecido oir un lindo gatito.");
    ];
```

Muchas otras cosas se podrían hacer para reaccionar ante el sonido, como moverse hacia el sonido o alejarse de él, etc... ¡intenta imaginar cómo!

A modo de ejercicio intenta fabricar un oido para nuestro loro de tal forma que repita todo lo que digan las 'voces humanas' que oiga (excepto la del jugador, eso sería bastante más complejo).

10 Ruidos (fuentes de sonido continuas)

Las fuentes de sonido 'continuas' (es decir, que no cesan nunca, que son como ruido de fondo), se tratan de una manera parcial, pero espero que aceptablemente suficiente en la librería. Estas fuentes de sonido están básicamente orientadas al jugador, esto es, realmente los PNJs las oirán pero sólo cuando el jugador realice activamente algo que pueda provocar que las escuche, estas cosas son dos: moverse y pedir un comando de 'escuchar'.

La clase básica para fuentes de sonido continuas es *Ruido*, que contiene los siguientes atributos parametrizables:

- **voz:** por defecto con valor *tipoPlano*, clase de sonido utilizado para generar el ruido..
- **frase:** por defecto con valor 0, frase que repite constantemente, si contiene 0 no dice nada sólo toca el sonido y, como siempre, se puede sustituir por una rutina que, por ejemplo, devuelva una frase aleatoria cada vez, o bien siga una secuencia fija, etc...
- **intensidad:** por defecto con valor 0, intensidad sonora del ruido en su origen, si se indica cero se usa a nivel Normal,

- **origen:** por defecto con valor *0*, el objeto que genera el sonido. Todo ruido debe tener un origen o simplemente no se escuchará.
- **sonando:** por defecto con valor *true*, indica si el ruido está activado o no.

Además los ruidos disponen

11 Uso de algunos atributos estándares de la librería en las voces y los ruidos

La librería usa dos atributos estándares de la librería para caracterizar voces y ruidos:

- **femenino:** se utiliza el atributo masculino/femenino, para distinguir si la voz es ‘masculina’ o ‘femenina’ a la hora de generar los adjetivos de distancia ‘lejano’ o ‘lejana’. Así si se trata de una voz que es ‘un maullido’ debería ser masculina, pero si es ‘una campanilla’ debería ser femenina.
- **oculto:** se utiliza en el caso de los ruidos para indicar si se debe mostrar en la descripción de los lugares donde se oye, o si sólo se oye bajo una petición expresa del jugador del comando ‘escuchar’. Es decir, si se quiere que sólo se perciban bajo un comando de ‘escuchar’ hay que añadirles el atributo de oculto.