

TUTOR DE VISUAL SINTAC

(c)2000 Login:

<http://pagina.de/jsj>
jsj666@hotmail.com

Fecha última modificación: 8/12/2000

INDICE

1. INTRODUCCION	1
2. INSTALAR VISUAL SINTAC	2
3. LA CREACION	4
4. VOCABULARIO	6
5. LOCALIDADES	8
6. TUTOR: CREANDO LOCALIDADES	10
7. EDITOR DE CONEXIONES	12
8. TUTOR: CONECTANDO LOCALIDADES	13
9. OBJETOS	18
10. TUTOR: CREANDO OBJETOS	20
11. PSIs	23
12. TUTOR: SITUANDO AL JUGADOR	24
PROPIEDADES DEFINIDAS POR EL USUARIO	24
PROGRAMACION CON VISUAL SINTAC	26
TIPOS DE DATOS Y VARIABLES	26
ARRAYS	27
EXPRESIONES	27
PROPIEDADES Y METODOS	28
ESTRUCTURAS DE CONTROL	29
Toma de decisiones	29
Estructuras de control	30
Salida inmediata	31
PROCEDIMIENTOS	31
13. MODULOS	33
14. ESTRUCTURA DE UN PROGRAMA	34
15. VISIBILIDAD DE LAS VARIABLES	35
16. LA LIBRERIA ESTANDAR	36
17. MODULO DECLARACIONES	37
MODULO USUARIO	38
PROPIEDADES DE USUARIO USADAS POR LA LIBRERIA ESTANDAR	39
18. TUTOR: CODIFICANDO	41
19. TUTOR: LA PUERTA	43
20. TUTOR: LA TELEVISION	46
21. TUTOR: UN PSI	48

22.	<i>GRAFICOS, SONIDOS Y TIPOS DE LETRA</i>	52
23.	<i>RECURSOS</i>	53
24.	<i>COMPILAR Y DISTRIBUIR LA AVENTURA</i>	55

1. INTRODUCCION

!!! Bienvenido al tutorial de Visual SINTAC !!!

Visual SINTAC es un sistema para crear aventuras conversacionales (SINTAC=Sistema INTEGRado para la creación de Aventuras Conversacionales). La coletilla "visual" es debida a que todo el trabajo se realiza dentro de un entorno... uhmmm... ¡visual!.

Con esto quiero decir que la mayor parte de las cosas tienen sus correspondientes cuadros de diálogo y diseñadores. Pero no todo es "visual", el núcleo de la aventura hay que programarlo y ello se hace desde un editor de procedimientos que no es más que un editor de textos adaptado a Visual SINTAC.

Pero primero definamos qué es una aventura conversacional. Se entiende por aventura conversacional aquella que está basada, casi exclusivamente, en texto. En las aventuras conversacionales se nos describe de forma textual, como en una novela, el entorno que nos rodea. Todas las acciones las introducimos tecleándolas en una línea de entrada.

Esta es la definición más básica que se puede dar. En realidad una aventura conversacional no requiere gráficos pero hay algunas que los llevan, aunque no de la forma masiva en que aparecen en las aventuras gráficas. Normalmente si una aventura conversacional lleva gráficos, estos son estáticos y muestran la localidad en la que nos encontramos. En ningún caso se elimina el texto de la descripción de la localidad, si la aventura conversacional lleva gráficos estos complementan al texto pero NUNCA lo sustituyen. La mayoría incluso permite desactivar los gráficos para contentar a los jugadores más puristas.

Por supuesto, Visual SINTAC soporta la inclusión de elementos multimedia (gráficos y sonido) en nuestras aventuras.

La programación de Visual SINTAC es bastante sencilla porque los cimientos ya están programados. Se incluye una aventura modelo que sirve de base y que provee del esqueleto para las nuestras. La aventura modelo incorpora ya programadas todas las inicializaciones de pantalla y respuestas a comandos estándar de toda aventura conversacional.

Por supuesto cualquiera puede construirse su esqueleto de aventura o modificar el que se suministra. Yo recomiendo que en principio, hasta estar familiarizado con el sistema, se use la aventura modelo como base y se vaya rellenando con lo que falta. Según se consiga experiencia con el sistema, seguramente los más osados decidirán ampliar la aventura modelo o, ¿por qué no?, programar la suya propia partiendo de cero.

Este tutorial mostrará cómo, usando la aventura modelo suministrada junto a Visual SINTAC podemos afrontar la programación de nuestra propia aventura.

2. INSTALAR VISUAL SINTAC

Visual SINTAC es bastante sencillo de instalar si se siguen una serie de pasos. En primer lugar hay que obtener los ficheros necesarios. Estos pueden encontrarse en la página de descargas en <http://pagina.de/jsj>.

Los ficheros imprescindibles son:

- ✍ El 'runtime' completo.

(i) Cuidado aquí porque hay dos 'runtimes', uno 'mini' que sólo sirve para ejecutar el intérprete en caso de que distribuyamos nuestras aventuras y otro completo que es el que permite ejecutar el entorno de programación. En este caso cogeremos el 'runtime' completo que se compone de varios ficheros ZIP, los descomprimiremos en un directorio y ejecutaremos el programa SETUP.EXE incluido.

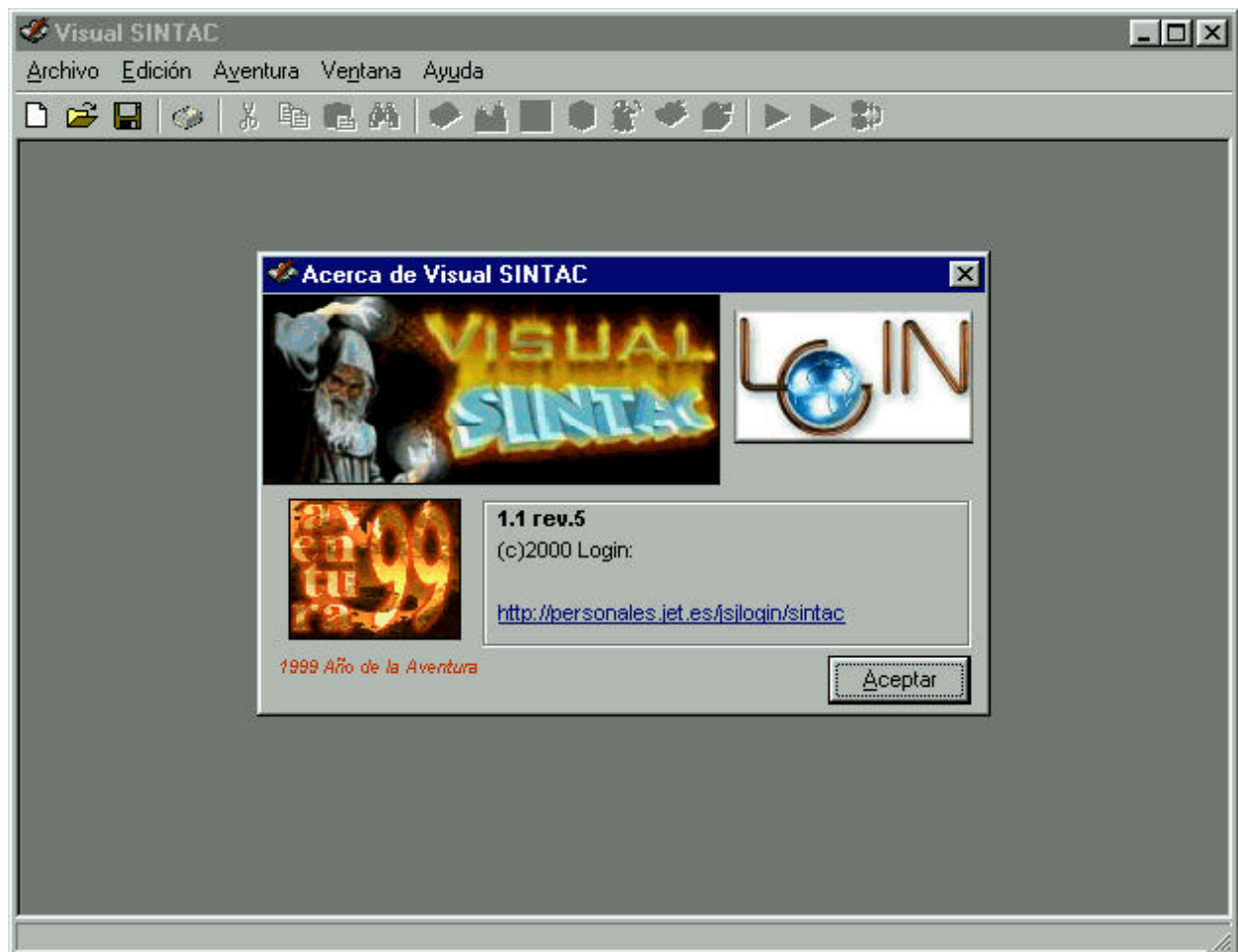
- ✍ Luego necesitaremos el fichero que contiene los programas. En la página de descargas aparecerá un enlace identificado como Visual SINTAC para descargar el fichero del programa. Este fichero es un ZIP que contiene los ejecutables del programa y los ficheros de ayuda y DLLs complementarias. Se debe descomprimir en un directorio y, opcionalmente, crear los accesos directos a los programas (el intérprete y el entorno de desarrollo). Los ficheros incluidos en el ZIP son los siguientes.

VisualSINTAC.exe	el entorno de desarrollo
InterpreteVS.exe	el intérprete
VS.dll	librería de apoyo
VisualSINTAC.hlp	
VisualSINTAC.cnt	ficheros de ayuda

- ✍ Además es conveniente descargarse también los ficheros de datos, un ZIP que contiene entre otras cosas la aventura modelo (AvModelo) que nos servirá como esqueleto para crear las nuestras.

Como se ha mencionado Visual SINTAC incluye dos ejecutables. El intérprete que servirá para que otros ejecuten nuestras aventuras una vez finalizadas y compiladas y el entorno de desarrollo.

Para acceder al entorno de desarrollo ejecutaremos el fichero VisualSINTAC.exe. Podemos crear un acceso directo en el escritorio de Windows para que tenerlo más a mano.



3. LA CREACION

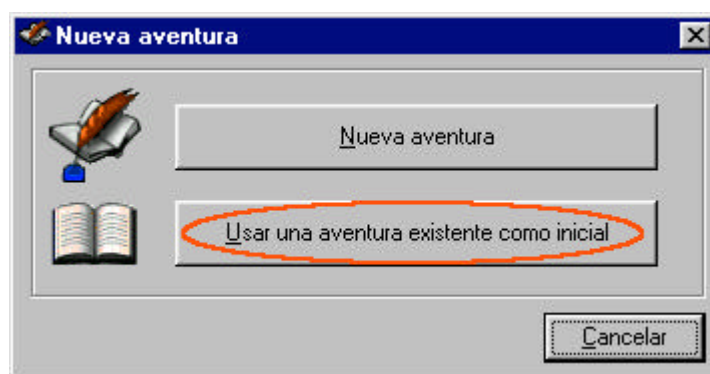
Primero debemos alojar un hueco en nuestro disco duro para la primera aventura. No es mala idea el crearse un directorio específico donde Visual SINTAC pueda guardar los ficheros necesarios de la aventura. Para este tutorial supondremos que se ha creado un directorio denominado 'Tutor' en algún lugar del disco duro.

Una aventura típica de Visual SINTAC está dividida en varios ficheros que pueden identificarse por su extensión. A continuación se muestran los tipos de ficheros que genera Visual SINTAC:

NombreAventura.LOC	Localidades
NombreAventura.OBJ	Objetos
NombreAventura.PSI	PSIs
NombreAventura.VOC	Vocabulario
NombreAventura.VST	Tabla de módulos
NombreAventura?????.VS	Módulos
NombreAventura.RES	Descripción de recursos (gráficos y sonidos)
NombreAventura.VSR	Recursos compilados

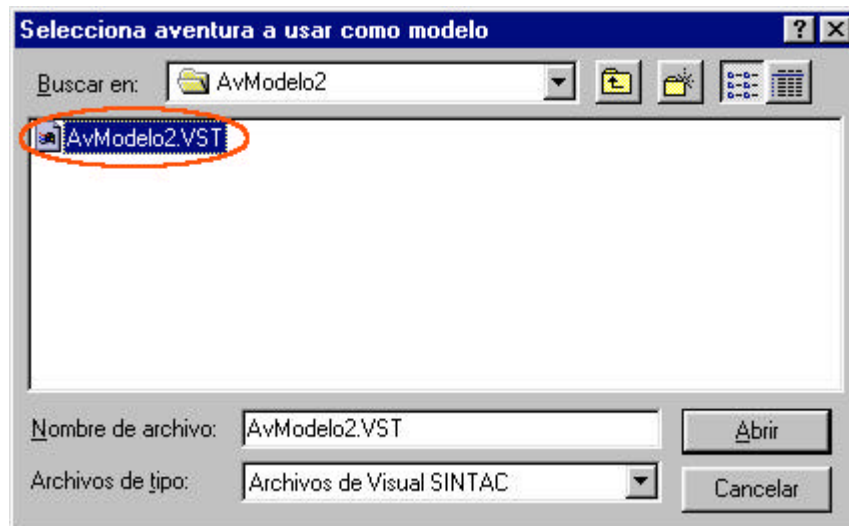
Además puede haber otros ficheros si la aventura incorpora gráficos y sonidos. Estos ficheros tendrán distintas extensiones según su formato (WAV, MOD, JPG, BMP,...).

Para comenzar una nueva aventura ejecutaremos Archivo? Nuevo con lo cual se abrirá una pantalla que nos da a elegir si queremos crear una 'Nueva aventura' o 'Usar una aventura existente como inicial'.

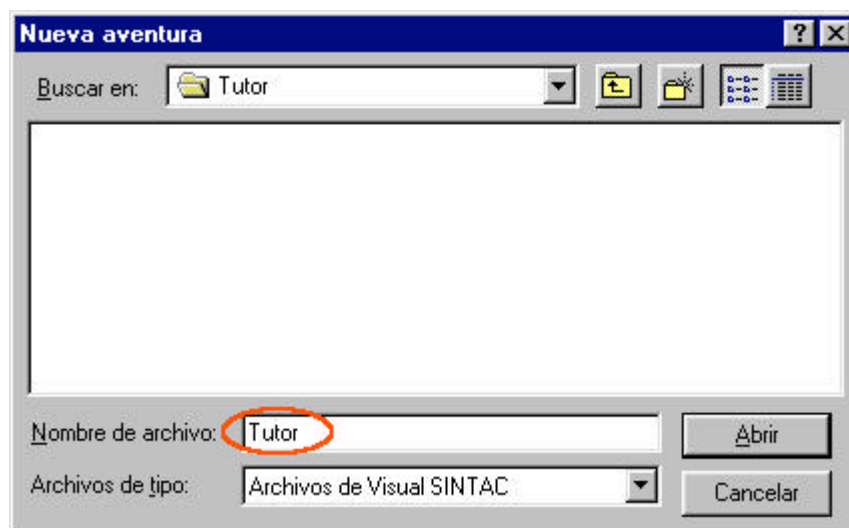


La primera opción creará una aventura totalmente en blanco con lo que sólo es recomendable si lo que queremos es programar nuestra aventura desde cero. En cambio la segunda opción nos permite seleccionar una aventura como modelo. En este caso podemos seleccionar 'AvModelo' que se distribuye con Visual SINTAC y que provee de un esqueleto básico que nosotros rellenaremos.

Al seleccionar esta última opción, Visual SINTAC presentará un diálogo para que indiquemos dónde se encuentra ubicada la aventura que usaremos como modelo, en este caso buscaremos y seleccionaremos 'AvModelo'.



Luego nos pregunta por el nombre con el que vamos a guardar nuestra aventura. Aquí seleccionaremos el directorio 'Tutor' que hemos creado previamente y teclearemos 'Tutor' como nombre de la aventura. Ahora ya tenemos todo listo para comenzar a crear nuestra aventura.



4. VOCABULARIO

El vocabulario es una tabla que recoge todas y cada una de las palabras que el 'parser' (analizador sintáctico) puede entender. El 'parser' se encarga de recoger la frase tecleada por el jugador y descomponerla en sus elementos básicos (palabras) de acuerdo al vocabulario definido. Posteriormente debemos, mediante programa, dar respuesta a las posibles acciones del jugador.

Dentro del vocabulario se maneja el concepto de sinónimo. Como cabe esperar, un sinónimo es una palabra que tiene el mismo significado que otra. En Visual SINTAC hay una palabra 'padre' y de esa pueden colgar uno o varios sinónimos. Así el verbo COGER podría ser el 'padre' y COGE, TOMAR y TOMA sus sinónimos.

Por ejemplo, supongamos que el jugador teclea algo como:

```
COGE LA LINTERNA
```

El 'parser' analizará esta frase y nos devolverá algo como:

```
Verbo = COGER  
Nombre1 = LINTERNA
```

¡No!, no nos hemos equivocado. El 'parser' devolverá siempre la palabra 'padre', que en este caso es COGER ya que COGE es uno de sus sinónimos.

El 'parser' de Visual SINTAC es capaz de analizar frases que contengan un verbo, dos nombres con sus adjetivos y una preposición. Por ejemplo:

```
METE LA LINTERNA DENTRO DEL SACO ROJO
```

Que se analizaría como:

```
Verbo = METER  
Nombre1 = LINTERNA  
Adjetivo1 = (ninguno)  
Nombre2 = SACO  
Adjetivo2 = ROJO  
Preposición = DENTRO
```

Además, el 'parser' es capaz de analizar frases concatenadas y distinguir verbos con terminación, por ejemplo:

```
COGE LA LINTERNA Y METELA DENTRO DEL SACO. NORTE.
```

Esto se analizaría como tres frases independientes:

```
Verbo = COGER  
Nombre1 = LINTERNA  
Adjetivo1 = (ninguno)  
Nombre2 = (ninguno)  
Adjetivo2 = (ninguno)  
Preposición = (ninguna)  
  
Verbo = METER
```

```
Nombre1 = LINTERNA
Adjetivo1 = (ninguno)
Nombre2 = SACO
Adjetivo2 = (ninguno)
Preposición = DENTRO

Verbo = NORTE
Nombre1 = (ninguno)
Adjetivo1 = (ninguno)
Nombre2 = (ninguno)
Adjetivo2 = (ninguno)
Preposición = (ninguna)
```

Este tipo de análisis sintáctico cubre la mayoría de acciones que se nos pueden ocurrir dentro de una aventura conversacional. Por supuesto este método no es el más potente pero sí lo suficiente como para estar a la altura de 'parsers' más 'profesionales'.

Como se puede deducir, la programación de respuestas a las acciones del usuario consistiría en una serie de comparaciones de las palabras que devuelve el 'parser'. De esta forma:

```
SI Verbo = VerboMovimiento ENTONCES Movemos_Jugador()
SI Verbo = "COGER" ENTONCES Cogemos_Objeto(NOMBRE1,ADJETIVO1)
SI Verbo = "METER" ENTONCES
    Metemos_Objeto_En(NOMBRE1,ADJETIVO1,NOMBRE2,ADJETIVO2)
```

Se ha usado una notación similar (pero no igual) a la que se usará dentro del lenguaje de programación de Visual SINTAC.

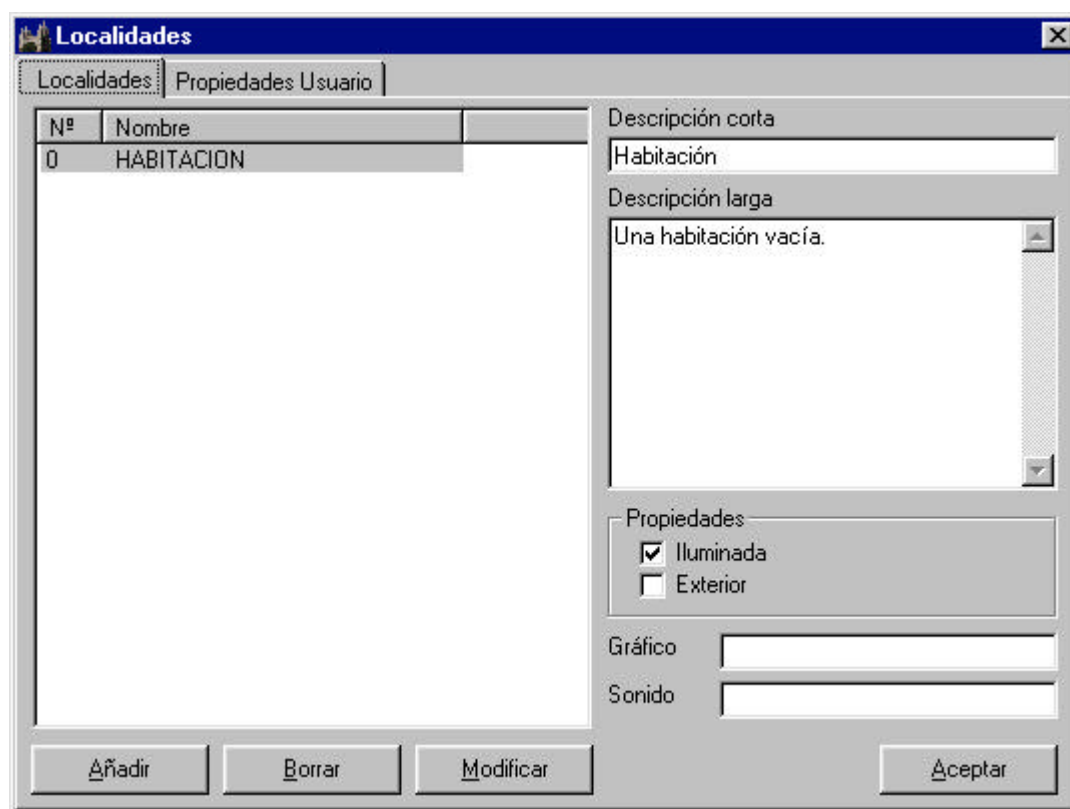
Por supuesto sería bastante pesado tener que programar en cada aventura todas y cada una de las respuestas. No hay que preocuparse por esto ya que la librería que se suministra junto a Visual SINTAC (la aventura modelo) incluye el código de respuesta a las acciones típicas. Esta librería además está preparada para que el usuario modifique la respuesta a las acciones contempladas sin necesidad de modificar toda la librería. Veremos más sobre esto cuando abordemos la programación con Visual SINTAC.

Para terminar hay que mencionar que existe un tipo de palabras en el vocabulario denominadas verbos de movimiento. Estas palabras son especiales en el sentido de que indican las direcciones con las que podemos definir conexiones entre localidades. En esta categoría aparecerán no sólo verbos sino también los nombres de los puntos cardinales.

5. LOCALIDADES

Toda la acción de una aventura conversacional discurre en diversas localidades. Estas son los lugares que el jugador puede visitar. Una localidad no tiene una extensión claramente definida y puede tratarse desde el interior de una habitación hasta un enorme desierto cubierto de dunas.

Para definir las localidades en Visual SINTAC usaremos la opción Aventura? Localidades (o el botón correspondiente de la barra de herramientas).



Dentro del editor de localidades podemos definir los lugares de interés de nuestra aventura. Previamente tenemos que tener claro cómo va a ser la distribución de las localidades y, lo más importante, cómo se conectan unas con otras.

Es común en todas las aventuras conversacionales que las localidades se conecten unas con otras usando los puntos cardinales (norte, sur, este, ...) o direcciones como arriba, abajo, entrar y salir. Esto hace que sea necesario diseñar un mapa de conexiones.

Además, en el diseño de localidades tenemos que tener en cuenta que cada una de ellas debe llevar una descripción corta y una descripción larga. La descripción corta suele ser una identificación de la localidad. Por ejemplo: La habitación. La descripción larga se imprimirá cada vez que el jugador visite la localidad o use el comando MIRAR. Esta puede ser más o menos elaborada en función de nuestras dotes de narrativa.

Además tenemos que decidir por cada localidad aspectos como si es interior o exterior, si está iluminada o a oscuras.

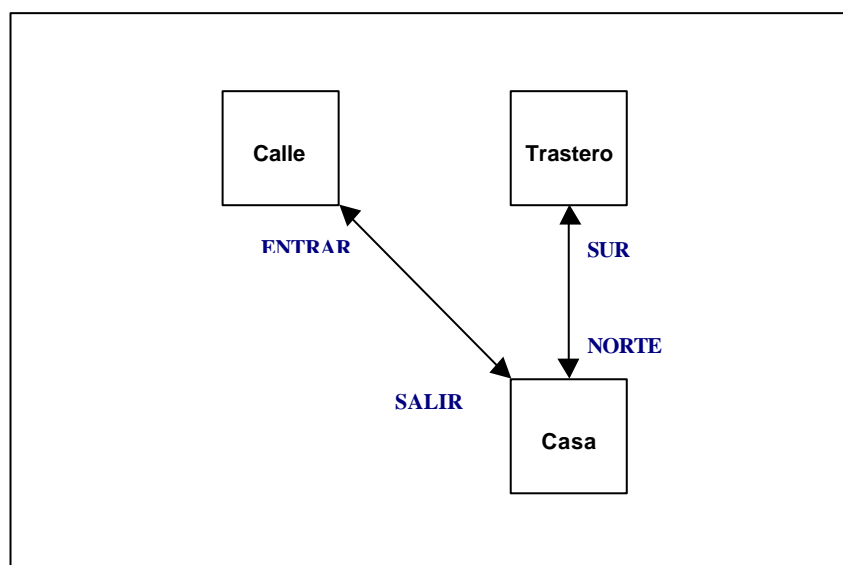
Dentro del editor de localidades, por cada localidad debemos definir:

- ✍ **Descripción corta:** es la descripción que se mostrará en la barra de estado.
- ✍ **Descripción larga:** la descripción que se mostrará al jugador cuando entre en la localidad o siempre que use el comando MIRAR.
- ✍ **Iluminada:** si esta propiedad no está activada la localidad estará a oscuras.
- ✍ **Exterior:** si la localidad es exterior o interior.
- ✍ **Gráfico:** gráfico que se mostrará al entrar en esta localidad (ver el apartado sobre Recursos)
- ✍ **Sonido:** sonido que sonará al entrar en esta localidad (ver el apartado sobre Recursos)

6. TUTOR: CREANDO LOCALIDADES

Lo primero que debemos planificar en toda aventura es el mapa de la misma, es decir las localidades que habrá y cómo están conectadas (realmente lo primero que deberíamos realizar es el guión sobre el que vamos a basar la aventura pero eso queda fuera del ámbito de este tutorial).

En nuestro caso vamos a tener un mapa que consta de 3 localidades conectadas como sigue:



El primer paso es definir las localidades. Para ello abrimos el editor de localidades (Aventura? Localidades).

(i) Si hubiese alguna localidad creada la borraríamos primero. Para ello la seleccionamos y pulsamos sobre el botón 'Borrar'.

Añadiremos una a una las localidades. Pulsamos el botón 'Añadir' y el programa nos mostrará un cuadro donde debemos introducir el nombre que queremos asignar a la localidad. En Visual SINTAC cada localidad, objeto y PSI tiene un nombre. El nombre del objeto debe ser único ya que identifica al objeto. Además del nombre cada localidad, objeto y PSI tiene un número que es el número de orden que se le asigna cuando se crea. Para hacer referencia a una localidad, desde código, podemos usar su número o su nombre. Es preferible que usemos el nombre ya que el número puede cambiar si borramos localidades intermedias (además no hay forma fácil de obtener ese número). Lo mismo ocurre con los objetos y PSIs.

El nombre de la primera localidad será 'CALLE' (los nombres se convierten a mayúsculas automáticamente).

Ya tenemos una nueva localidad. Ahora tendremos que definir sus propiedades:

✎ **Descripción corta:** 'La calle'

✍ **Descripción larga:** 'Estás en la avenida principal, al lado del portal de tu casa. Una gran puerta te permite la entrada.'

✍ **Iluminada:** DESACTIVADO.

✍ **Exterior:** ACTIVADO.

✍ **Gráfico:** en blanco.

✍ **Sonido:** en blanco.

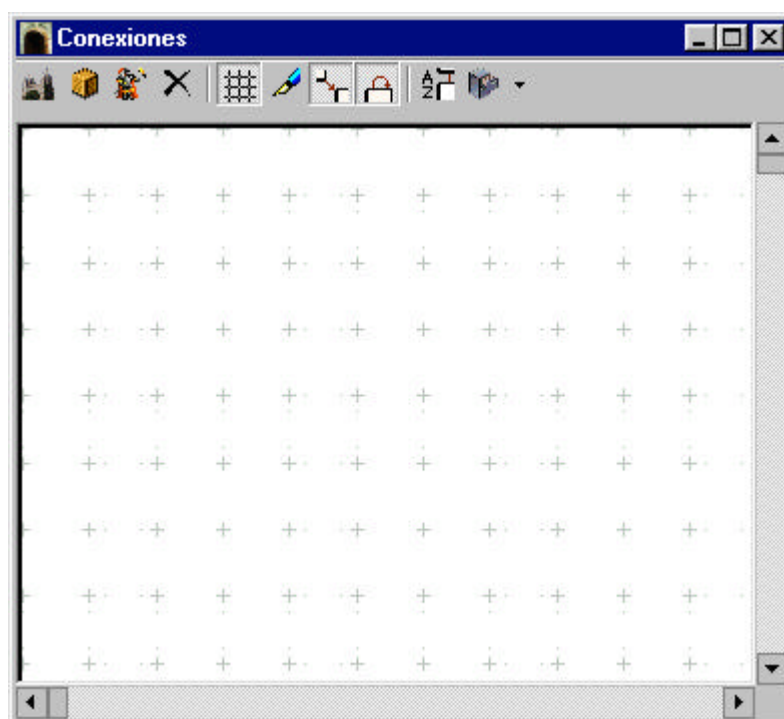
Las otras 2 localidades las definiremos igual:

Nº	Nombre	Desc. Corta	Desc. Larga	Ilum.	Ext.
1	CASA	Interior de la casa	El interior de la casa es amplio y cómodo. Un pasillo conduce al trastero y una puerta a la calle.	Sí	No
2	TRASTERO	El trastero	Dentro del trastero se apilan multitud de viejos objetos.	No	No

7. EDITOR DE CONEXIONES

Una vez que hemos definido las localidades y sus descripciones debemos conectarlas. Como ya se ha mencionado, en cualquier aventura se suele trazar un mapa de la misma, es decir, cómo están interconectadas las distintas localidades. Es común usar las direcciones cardinales para conectar las localidades (norte, sur, este, oeste, ...) a parte de otras direcciones típicas de cualquier aventura conversacional (subir, bajar, entrar, salir).

Para acceder al editor de conexiones usaremos la opción Aventura? Conexiones:



El interface del editor de conexiones presenta una zona cuadriculada (o no, si hemos desactivado la cuadrícula) sobre la que colocaremos las localidades y las conectaremos sin más que trazar la línea de conexión entre ellas.

Cuando conectamos dos localidades, Visual SINTAC presentará una lista con los verbos de conexión que tenemos disponibles en el vocabulario.

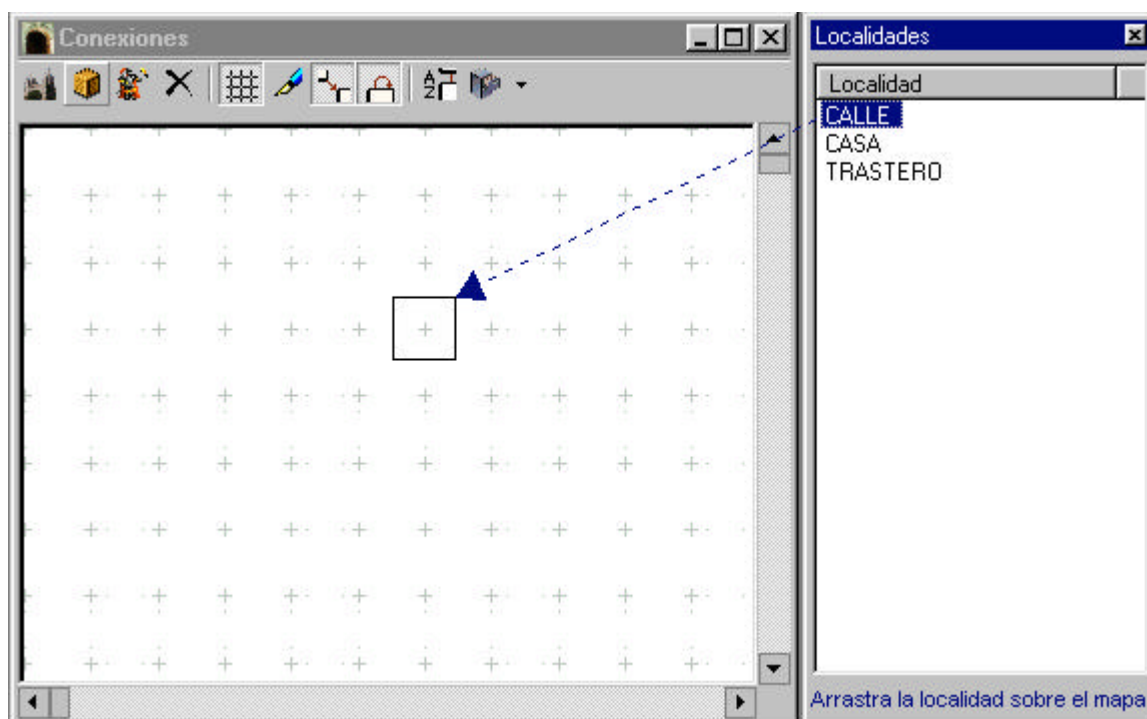
Hay que tener presente que una conexión tiene dos sentidos y hay que definirla en ambos para que el jugador pueda moverse entre las dos localidades. Si sólo definimos conexión en un sólo sentido esta será de ida pero no de vuelta. Realmente lo que se definen son dos conexiones diferentes por lo que si, por ejemplo, hemos conectado una localidad con otra hacia el NORTE podemos conectarla de forma inversa hacia, digamos, el OESTE para crear conexiones como las de los clásicos laberintos (si has jugado la Aventura Original entenderás esto).

Además se nos permite cambiar el estado de las conexiones. Una conexión puede estar abierta o cerrada. Cuando una conexión está cerrada el jugador no podrá 'atravesarla'. Esta característica es útil para implementar puertas u otra clase de obstáculos dentro de las localidades.

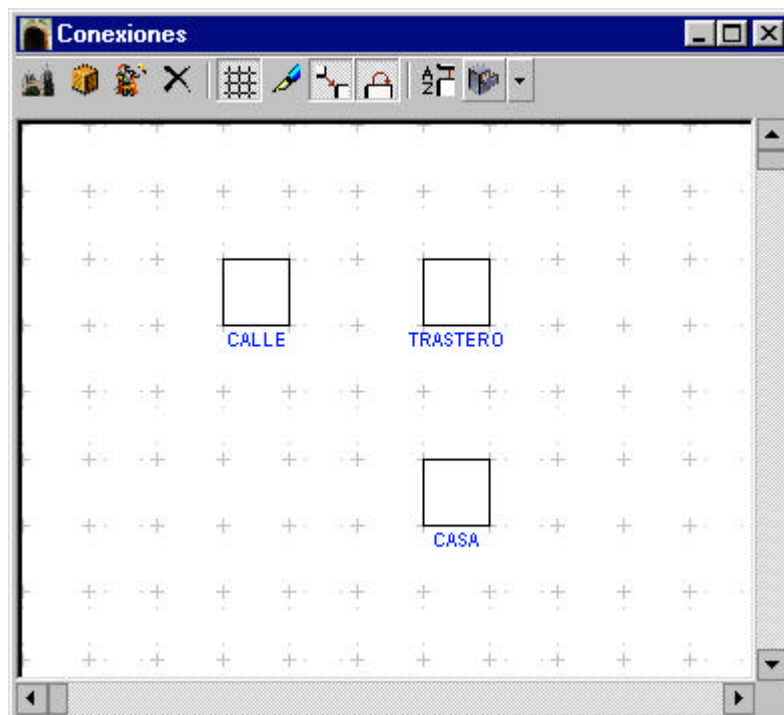
8. TUTOR: CONECTANDO LOCALIDADES

De forma muy sencilla hemos definido las localidades que componen nuestro mapa. Ahora tenemos que conectarlas entre sí. Para ello abriremos el editor de mapas (Aventura? Conexiones).

Para situar las localidades sobre el mapa abrimos la lista de localidades (pulsando el botón correspondiente de la barra de herramientas del editor de conexiones), y las arrastraremos sobre la superficie del mapa.



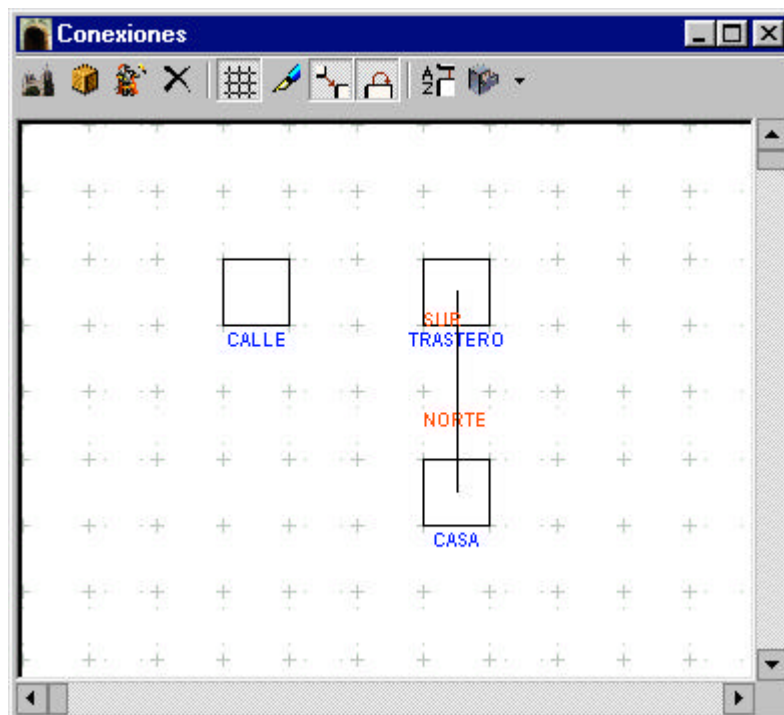
De esta forma iremos situando las localidades. Si queremos desplazar una localidad sobre el mapa pincharemos sobre ella, con lo que quedará seleccionada, y la arrastraremos hacia su nuevo posición.



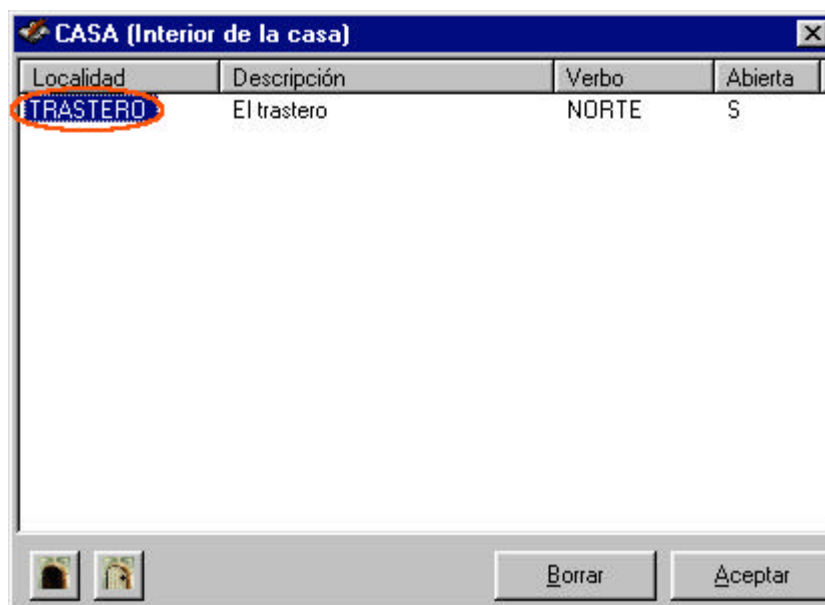
Tenemos que conectar las localidades. Conectaremos la casa con el trastero. El procedimiento es el siguiente:

1. Pinchamos con el botón derecho sobre la localidad de la casa. El editor de conexiones se pondrá en modo de trazado de conexiones (aparecerá una línea punteada que seguirá al ratón).
2. Llevamos la línea punteada a la localidad del trastero y pinchamos con el botón izquierdo. Aparecerá un desplegable con los verbos de conexión disponibles. En este caso elegimos 'NORTE'.

Ya tenemos una conexión desde la casa hacia el trastero. Tenemos que repetir el proceso pero al revés, es decir, unir el trastero con la casa hacia el SUR ya que en caso contrario no podremos ir del trastero a la casa.



Para borrar una conexión que hemos trazado habría que hacer doble-click sobre la localidad de la que parte la conexión. Se abrirá una lista con las conexiones que parten de esta localidad. En esta lista podemos seleccionar la conexión y borrarla:



Quedaría trazar una conexión de la calle a la casa para entrar y otra de la casa a la calle para salir. Lo haremos pero esta vez seleccionaremos ENTRAR para la conexión de la calle a la casa y SALIR para la inversa.

Por defecto, todas las conexiones que creamos estarán abiertas. Esto quiere decir que no habrá ningún obstáculo que impida que el jugador se mueva siguiendo esa conexión.

Podemos cambiar el estado de cualquier conexión y cerrarla. Cuando una conexión está cerrada el jugador no podrá 'recorrerla'. Vamos a cerrar la puerta de entrada de la calle a la casa. Para ello abrimos el editor de conexiones (Aventura? Conexiones) y hacemos doble clic sobre la localidad de la calle:

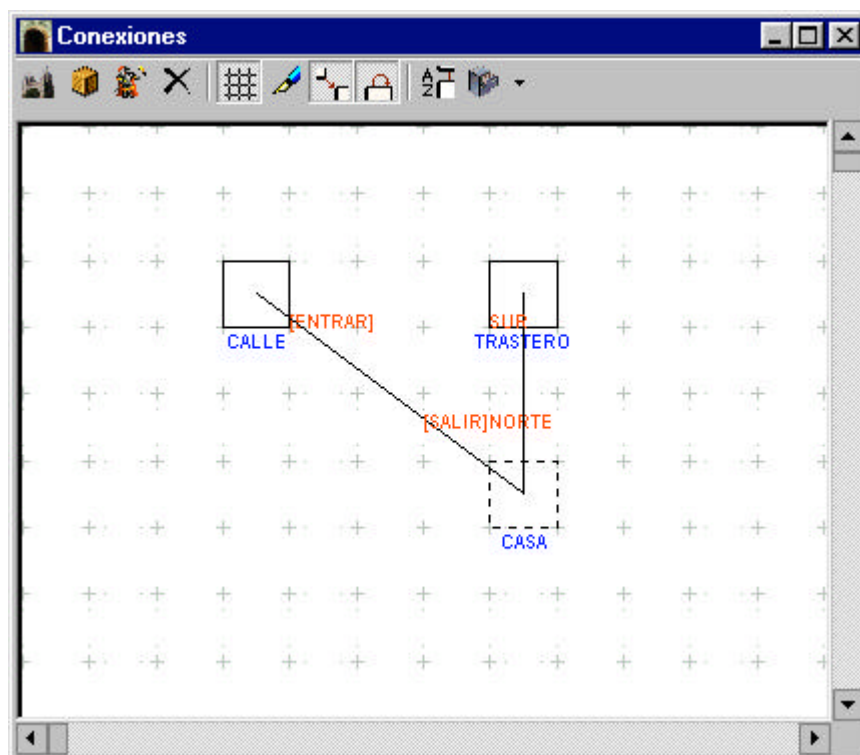


El cuadro de diálogo que se abre muestra las conexiones que parten desde la localidad. En este caso sólo tenemos una conexión que une la calle con la casa y el verbo de movimiento que hay que usar es ENTRAR. Si nos fijamos hay una columna que indica el estado de la conexión. La columna 'Abierta' puede contener S si la conexión está abierta o N si está cerrada.

Para cambiar el estado de una conexión pulsamos sobre ella para seleccionarla y pulsamos el botón de cerrar conexión. Otra opción es pulsar, en esta misma pantalla, sobre la conexión con el botón derecho del ratón y elegir la opción adecuada del menú contextual. En este caso seleccionaríamos 'Cerrar'.

Repetimos el proceso con la localidad de la casa, con la conexión que va de la casa a la calle. En el caso de conexiones bidireccionales habrá que repetir esto ya que hay que cerrar ambos extremos.

Si observamos la pantalla del mapa veremos que las conexiones que están cerradas aparecen marcadas:



Con esto queremos simular que la puerta de la calle está cerrada. Nos quedaría codificar la apertura y cierre de la puerta ya que esto no lo contempla la librería estándar. Más adelante veremos cómo hacer esto.

9. OBJETOS

Los objetos son las entidades que el jugador puede manipular. Dentro de Visual SINTAC cada objeto se define por una serie de propiedades.

- ✍ **Nombre y Adjetivo:** cada objeto se distingue de los demás por estas propiedades. El adjetivo es opcional pero si dos objetos deben tener el mismo nombre hay que asignarles adjetivos diferentes. No puede haber dos objetos LLAVE sin adjetivo pero sí una LLAVE DORADA y otra LLAVE PLATEADA. El nombre y adjetivo del objeto se introducen en el vocabulario automáticamente al ejecutar la aventura. NO ES NECESARIO DEFINIR LOS NOMBRES Y ADJETIVOS DE LOS OBJETOS EXPLICITAMENTE EN EL VOCABULARIO.
- ✍ **Descripción corta:** la descripción del objeto que se imprimirá al mostrarlo en el inventario del jugador o en la lista de objetos de la localidad. No hay que anteponerle el artículo ya que la librería lo añadirá automáticamente en los mensajes siempre que haga falta.
- ✍ **Descripción larga:** el texto que se mostrará cuando el jugador examine el objeto.
- ✍ **Peso y Tamaño:** el peso del objeto y su tamaño.
- ✍ **Contenido en:** especifica dónde se encuentra el objeto. Un objeto puede estar inicialmente en una localidad, dentro de otro objeto (si este último es un contenedor) o lo lleva un PSI.
- ✍ **Femenino:** indica el género del objeto. Esta propiedad determina que artículo se antepondrá a la descripción corta del objeto en los mensajes generados por la librería.
- ✍ **Plural:** si el número del objeto es plural o singular. Esta propiedad afecta al artículo que se antepondrá a la descripción corta del objeto en los mensajes de la librería.
- ✍ **EsContenedor:** si esta propiedad está activada, el objeto actuará como contenedor de otros objetos. Objetos como bolsas, mochilas y cajas tendrán esta propiedad activa.
- ✍ **Luz:** indica si el objeto es una fuente de luz. Las fuentes de luz permitirán ver dentro de localidades que estén a oscuras.
- ✍ **Invisible:** si un objeto es invisible estará presente pero no aparecerá en las descripciones.
- ✍ **TieneTapa:** un objeto contenedor puede tener tapa, si tiene tapa podrá ser abierto y cerrado.
- ✍ **Abierto:** se aplica a los objetos con tapa e indica si está abierto o cerrado.
- ✍ **Prenda:** un objeto marcado como prenda puede ser puesto y quitado de encima.
- ✍ **Puesto:** indica si una prenda está puesta o quitada.
- ✍ **Escenario:** los objetos de escenario aparecen dentro de la descripción de la localidad y no pueden ser movidos del sitio. En los objetos de escenario es importante definir la descripción corta para que 'encaje' con la descripción de la localidad.

- ✍ **Encendido**: una fuente de luz puede estar encendida o apagada. Sólo iluminará si está encendida.
- ✍ **Gráfico**: gráfico del objeto (ver el apartado sobre Recursos).
- ✍ **Sonido**: sonido asociado al objeto (ver el apartado sobre Recursos).

10. TUTOR: CREANDO OBJETOS

Vamos a crear algún objeto para dar al jugador la oportunidad de que interaccione. En Visual SINTAC disponemos del editor de objetos para añadir objetos a nuestra aventura (Aventura? Objetos):

The screenshot shows the 'Objetos' editor window. It has a title bar 'Objetos' and two tabs: 'Objeto' and 'Propiedades Usuario'. The 'Objeto' tab is active, showing a table with columns 'Nº', 'Nombre', and 'Adjetivo'. The 'Propiedades Usuario' tab is also visible, showing fields for 'Descripción corta', 'Descripción larga', 'Propiedades' (Peso, Tamaño, Contenido en, and checkboxes for Femenino, Plural, EsContenedor, Luz), 'Gráfico', and 'Sonido'. At the bottom are buttons 'Añadir', 'Borrar', 'Modificar', and 'Aceptar'.

(i) Si hubiese algún objeto creado lo borraríamos primero. Para ello lo seleccionamos y pulsamos sobre el botón 'Borrar'.

Hemos creado una localidad a oscuras (el trastero) así que sería muy propio el crear un objeto que nos proporcione luz, por ejemplo una LINTERNA. Añadimos un nuevo objeto pulsando el botón 'Añadir'. El programa nos solicita el nombre, en este caso tecleamos 'LINTERNA'.

A continuación rellenaremos el resto de propiedades:

✍ **Descripción corta:** 'linterna' (obsérvese que no se pone 'una linterna' ni 'la linterna' para dejar que el programa sea el que añada los artículos según el género y número de los objetos).

✍ **Descripción larga:** 'Es una de esas linternas con un mango largo donde van introducidas las pilas.'

✍ **Peso:** 1

✍ **Tamaño:** 1

✍ **Contenido en:** marcamos la opción 'Loc.' y en la casilla desplegable elegimos la localidad 'CASA'.

✍ **Gráfico:** en blanco.

✍ **Sonido:** en blanco.

✍ Además de estas propiedades, cada objeto dispone de una serie de propiedades que sólo pueden tener dos estados, activadas o desactivadas y que son:

✍ **Femenino:** ACTIVADO.

✍ **Plural:** DESACTIVADO.

✍ **EsContenedor:** DESACTIVADO.

✍ **Luz:** ACTIVADO.

✍ **Invisible:** DESACTIVADO.

✍ **TieneTapa:** DESACTIVADO.

✍ **Abierto:** DESACTIVADO.

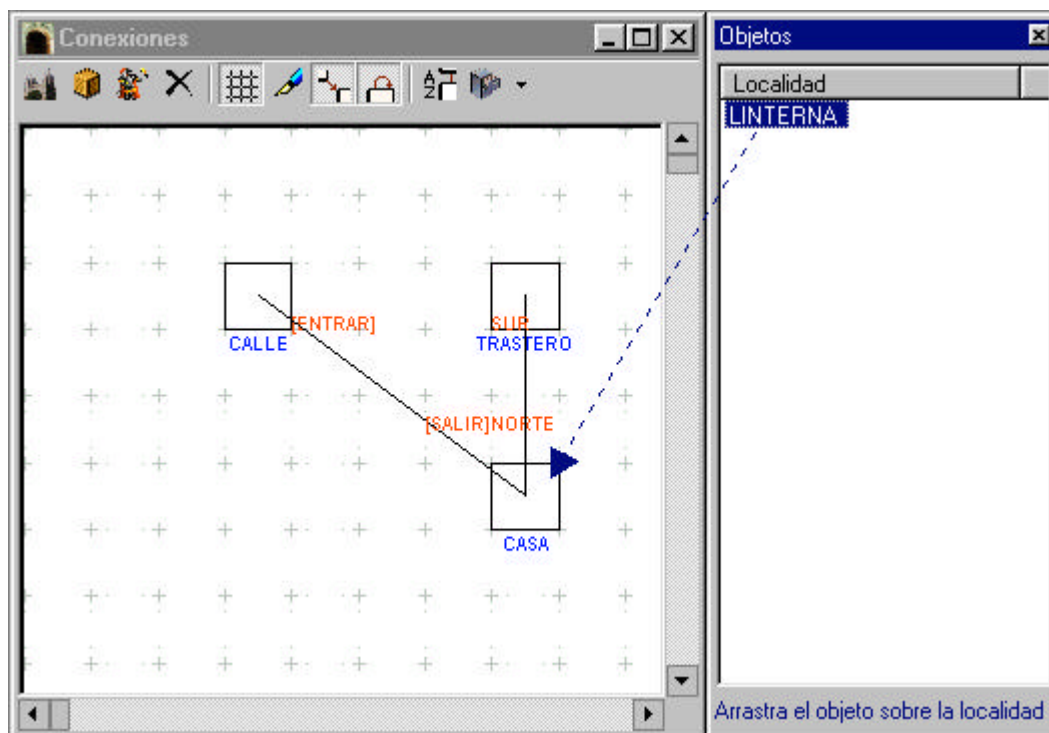
✍ **Prenda:** DESACTIVADO.

✍ **Puesto:** DESACTIVADO.

✍ **Escenario:** DESACTIVADO.

✍ **Encendido:** DESACTIVADO (está apagada de inicio).

Otro método para situar un objeto dentro de una localidad es hacerlo por medio del editor de conexiones (Aventura? Conexiones). Dentro del mismo abriremos la lista de objetos y arrastraremos el objeto sobre la localidad donde lo queremos colocar:



Ahora crearemos otro objeto al que daremos como nombre TELEVISION. En este caso se trata de un objeto de escenario (las propiedades que no se muestran se dejarán con su valor por defecto):

- ✎ **Descripción corta:** 'En el centro de la estancia hay una televisión' (observar que no ponemos el punto final ya que de ello se encargará la librería al tratarse de un objeto de escenario).
- ✎ **Descripción larga:** 'Una vieja tele en blanco y negro.'
- ✎ **Contenido en:** marcamos la opción 'Loc.' y en la casilla desplegable elegimos la localidad 'CASA'.
- ✎ **Femenino:** ACTIVADO.
- ✎ **Escenario:** ACTIVADO.
- ✎ **Encendido:** DESACTIVADO (está apagada de inicio). NOTA: aunque esta propiedad está pensada para usarse con fuentes de luz y este objeto no lo es, la usaremos para controlar el estado de la televisión.

11. PSIs

Los PSIs son los personajes que habitan el mundo creado por el programador. Se diferencian de los objetos en el modo de interaccionar que tiene el jugador con ellos. Por ejemplo el jugador normalmente no sentirá la necesidad de hablar con los objetos (bueno, quizá en momentos de desesperación absoluta sí), en cambio sí que podrá hacerlo con los PSIs.

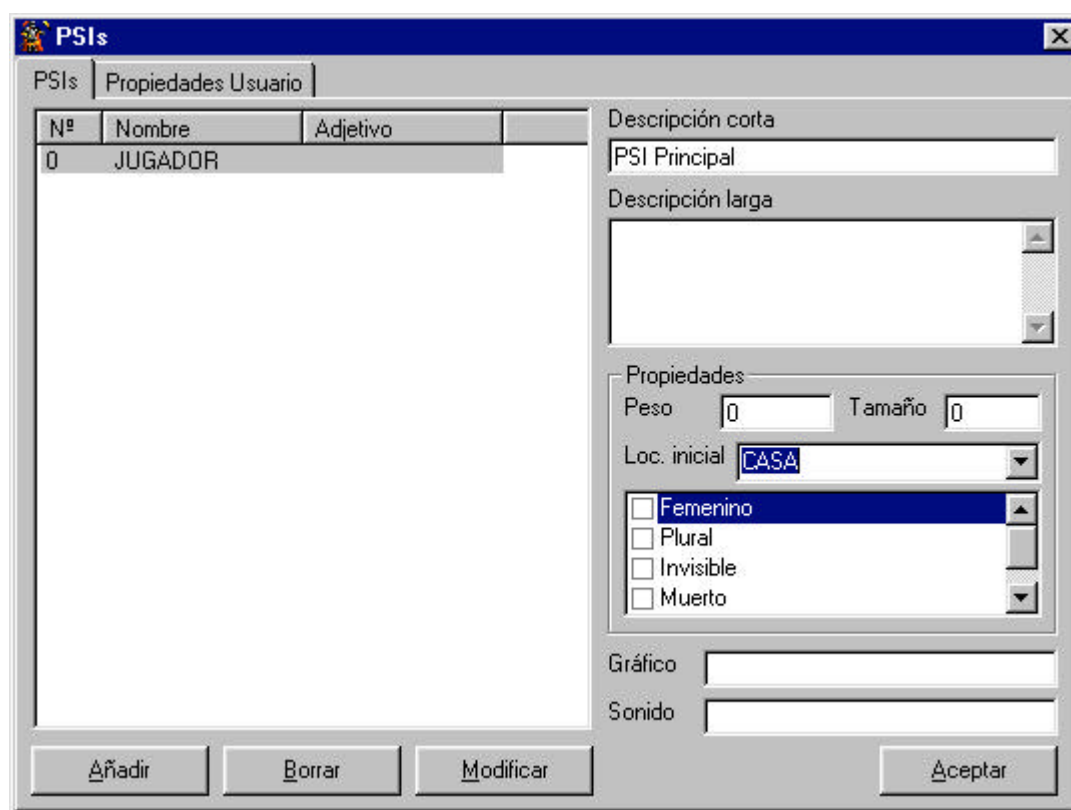
Dentro de Visual SINTAC cada PSI se define por una serie de propiedades.

- ✍ **Nombre y Adjetivo:** cada PSI se distingue de los demás por estas propiedades. El adjetivo es opcional pero si dos PSIs deben tener el mismo nombre hay que asignarles adjetivos diferentes. No puede haber dos PSIs SOLDADO sin adjetivo, pero sí un SOLDADO GORDO y otro SOLDADO FLACO. El nombre y adjetivo del PSI se introducen en el vocabulario automáticamente al ejecutar la aventura. NO ES NECESARIO DEFINIR LOS NOMBRES Y ADJETIVOS DE LOS PSIS EXPLICITAMENTE EN EL VOCABULARIO.
- ✍ **Descripción corta:** la descripción del PSI que se imprimirá al mostrarlo en la lista de PSIs de la localidad. No hay que anteponerle el artículo ya que la librería lo añadirá automáticamente en los mensajes siempre que haga falta.
- ✍ **Descripción larga:** el texto que se mostrará cuando el jugador examine al PSI.
- ✍ **Peso y Tamaño:** el peso del PSI y su tamaño.
- ✍ **Localidad inicial:** la localidad dónde el PSI está al iniciar la aventura. Si se deja en blanco el PSI estará en 'ningún sitio'.
- ✍ **Femenino:** indica el género del PSI. Esta propiedad determina que artículo se antepondrá a la descripción corta PSI en los mensajes generados por la librería.
- ✍ **Plural:** si el número del PSI es plural (son varios, por ejemplo: los soldados) o singular. Esta propiedad afecta al artículo que se antepondrá a la descripción del PSI en los mensajes de la librería.
- ✍ **Invisible:** si un PSI es invisible estará presente pero no aparecerá en las descripciones.
- ✍ **Muerto:** un PSI muerto está 'desactivado'.
- ✍ **Escenario:** (no se usa).
- ✍ **Gráfico:** gráfico del PSI (ver el apartado sobre Recursos).
- ✍ **Sonido:** sonido asociado al PSI (ver el apartado sobre Recursos).

Es importante tener en cuenta que debe haber al menos un PSI definido que es el que controlará el jugador. Visual SINTAC permite que el jugador tome el control de cualquier PSI siempre y cuando el programador así lo decida.

12. TUTOR: SITUANDO AL JUGADOR

Abrimos el editor de PSIs y observaremos que hay un PSI genérico definido. Este será el que controle el jugador por defecto. Lo único que modificaremos es la localidad donde comienza que, en este caso, será la CASA.

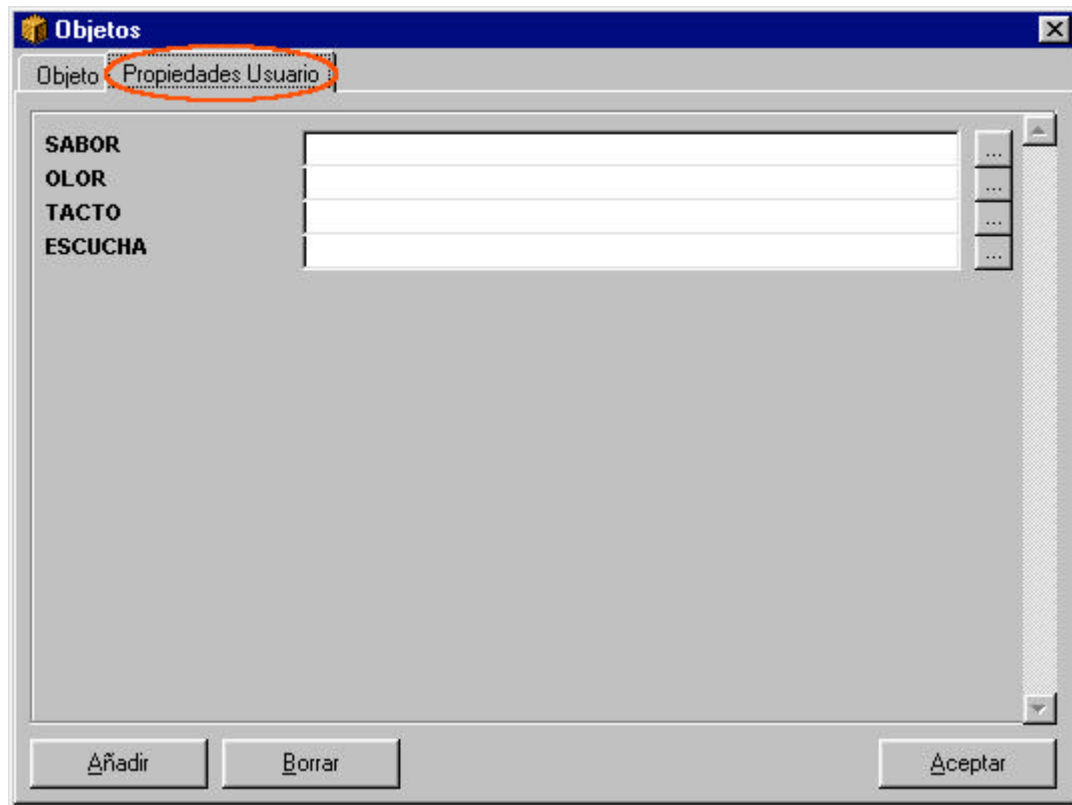


PROPIEDADES DEFINIDAS POR EL USUARIO

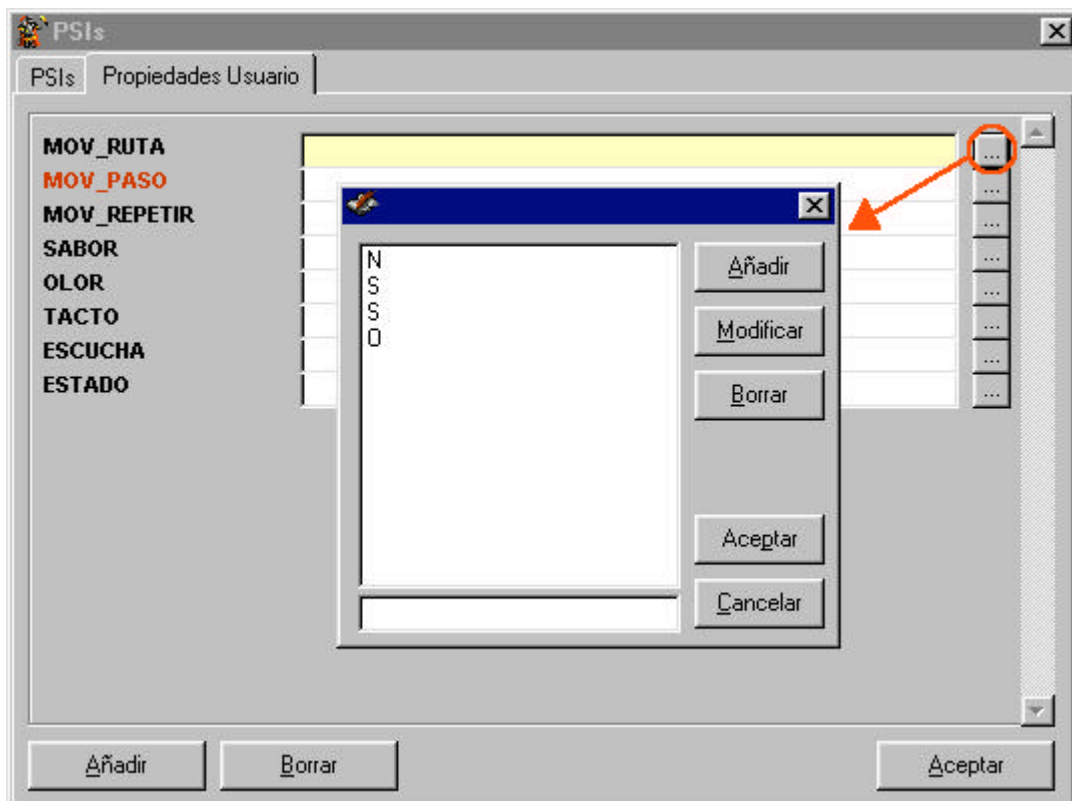
Tanto en las localidades como en los objetos y PSIs se pueden definir otras propiedades a parte de las que proporciona automáticamente Visual SINTAC. Estas propiedades se denominan, lógicamente, propiedades definidas por el usuario.

Dentro de la interface de Visual SINTAC, las propiedades definidas por el usuario, se presentan agrupadas en un apartado, separadas del resto de propiedades para no confundirlas.

Las propiedades definidas por el usuario pueden contener cualquier tipo de valor de los que soporta Visual SINTAC. Desde la interface podemos introducir, en las propiedades definidas por el usuario, valores numéricos, cadenas de texto y arrays (listas de valores).



Si la propiedad contiene un array de valores estos se deben editar desplegando el editor de listas. Se puede acceder al mismo pulsando el botón situado junto al valor de la propiedad. Cuando una propiedad de usuario contiene un array, su casilla de valor se muestra en otro color y su contenido sólo será visible desde el editor de listas.



La librería estándar de Visual SINTAC incluye una serie de propiedades de usuario que son usadas por ciertas acciones de la propia librería. En el apartado sobre la librería estándar se ampliará la información relativa a estas propiedades.

PROGRAMACION CON VISUAL SINTAC

La lógica que controla la aventura se debe programar, para ello Visual SINTAC dispone de un sencillo, pero potente, lenguaje de programación similar al BASIC.

Los que hayan programado alguna vez no deberían tener problemas a la hora de enfrentarse al lenguaje de Visual SINTAC, aunque encontrarán que Visual SINTAC no provee de ciertas facilidades que tienen los lenguajes de programación de uso general. Aquellos que no han usado nunca un lenguaje de programación quizá deberían consultar algún libro de programación en BASIC o cualquier otro lenguaje (Pascal, C) para obtener los conocimientos básicos y comprender mejor las bases de la programación.

TIPOS DE DATOS Y VARIABLES

En todo programa de ordenador tendremos que manejar datos. Estos datos pueden ser de muchos tipos y en Visual SINTAC estaremos continuamente manejando uno de estos tres tipos de datos:

✍ **Números:** valores numéricos enteros, Visual SINTAC no admite números decimales.

✍ **Textos:** cadenas de texto.

✍ **Arrays:** listas de valores de uno de los otros tipos.

Estos tipos de datos pueden aparecer como constantes dentro de una expresión o podemos tenerlos contenidos dentro de una variable. Una variable es un almacén donde guardamos un valor para su posterior uso. En Visual SINTAC hay que declarar las variables antes de su uso, la sentencia que permite esto es `Declare(variable)`. Por ejemplo:

```
Declare(x)           // declara la variable 'x'
Declare(mensaje)     // declara la variable 'mensaje'
```

Los nombres de variables tienen que cumplir unas reglas básicas:

✍ No deben comenzar por un número.

✍ Sólo puede contener caracteres alfanuméricos y el símbolo de subrayado.

Una variable recién declarada no contendrá ningún valor, estará vacía. Para asignarle un valor a una variable usaremos el operador de asignación que es `:=`, por ejemplo:

```
x:=10
mensaje:="Hola."
```

Como observamos, las cadenas de texto deben ir metidas entre comillas dobles, no así los números.

ARRAYS

Los arrays son listas de valores numéricos o cadenas de texto. Estos se crean y manipulan usando una serie de funciones predefinidas en el lenguaje de Visual SINTAC:

```
Array(elemento1, elemento2, elemento3, ...)
ArrayLen(array)
ArrayItem(array,n)
InArray(array,elemento)
SubArray(array,inicio,longitud)
ArrayInsert(array,posición,elemento)
ArrayLet(array,posición,valor)
ArrayFormat(array,separador,separador_final,punto_final,texto_vacio)
```

Estas funciones están documentadas en la ayuda. Por ejemplo:

```
Declare(lista)
lista:=Array("uno","tres","cuatro")           // la variable 'lista' tendrá:
                                                //  "uno", "tres", "cuatro"

lista:=ArrayInsert(lista,2,"dos")              // la variable 'lista' tendrá:
                                                //  "uno", "dos", "tres", "cuatro"

Declare(valor)
valor:=ArrayItem(lista,2)                     // la variable 'valor' tendrá: "dos"
lista:=ArrayLet(lista,3,"TRES")               // la variable 'lista' tendrá:
                                                //  "uno", "dos", "TRES", "cuatro"
```

EXPRESIONES

Podemos operar con los valores usando los operadores básicos de Visual SINTAC. Estos se dividen en operadores aritméticos y operadores lógicos.

Los operadores aritméticos disponibles son:

Suma	+
Resta	-
Multiplicación	*
División	/

El operador suma tiene doble función dependiendo del tipo de los operandos:

- ✍ Si todos los operandos son numéricos, se realiza la operación aritmética de la suma.
- ✍ Si alguno de los operandos es una cadena, se concatenan todos los operandos convirtiendo, previamente, los numéricos en cadenas.

Se pueden usar paréntesis para cambiar el orden de evaluación de las operaciones.

(i) Actualmente el analizador de expresiones de Visual SINTAC no distingue precedencia de operadores y los evalúa de izquierda a derecha en el orden en que los encuentra. Así la expresión: $3 + 2 * 5$ dará 25 y no 13 como sería de esperar.

Ejemplo:

```
58 + 16 * (23 / a)
"La casa " + adjetivo
"Tienes " + (ptos + 1) + " puntos."
```

Se pueden usar, además, los operadores lógicos siguientes:

Igual	=
Mayor que	>
Menor que	<
Mayor o igual que	>=
Menor o igual que	<=
Distinto de	<>
Y	&
O	
Negación	Not(expresión)

(i) Se deben usar paréntesis para limitar cada expresión lógica. En Visual SINTAC se considera el valor 1 como verdadero y el valor 0 como falso.

Ejemplo:

```
loc=3
(loc="LLANURA") & (vida>5)
((loc="CASA") & (peso=0)) | (loc="POSADA")
Not(loc="CASA") & (peso=10)
```

PROPIEDADES Y METODOS

Las localidades, objetos y PSIs disponen de una serie de propiedades y métodos predefinidos (tal y como se describen en la ayuda).

Las propiedades definen valores de las localidades, objetos o PSIs que determinan su estado y comportamiento. Los métodos permiten ejecutar acciones específicas sobre la localidad, objeto o PSI especificados.

Para referirse a las propiedades y métodos de las localidades se usa la siguiente nomenclatura:

```
LOC[localidad].Propiedad
LOC[localidad].Metodo([parametro1,parametro2,...])
```

dónde 'localidad' es el nombre que identifica la localidad, por ejemplo:

```
LOC["HABITACION"].Exterior
```

En el caso de objetos usaremos la siguiente nomenclatura:

```
OBJ[objeto].Propiedad  
OBJ[objeto].Metodo([parametro1,parametro2,...])
```

dónde 'objeto' es el nombre que identifica al objeto, por ejemplo:

```
OBJ["SACO ROJO"].Objetos()
```

Y en el caso de PSIs:

```
PSI[psi].Propiedad  
PSI[psi].Metodo([parametro1,parametro2,...])
```

dónde 'psi' es el nombre que identifica al PSI, por ejemplo:

```
PSI["NIÑA"].Localidad
```

Para obtener más información sobre las propiedades y métodos disponibles se debe consultar la ayuda.

ESTRUCTURAS DE CONTROL

Cualquier programa que realicemos suele consistir en un flujo de datos e instrucciones el cual podemos bifurcar o redirigir en función de una serie de decisiones que tomamos durante la ejecución. Todo programa tiene un punto de entrada y uno o varios puntos de salida dependiendo de por donde se bifurque la ejecución del mismo. Estas decisiones y bifurcaciones se controlan mediante las estructuras de control del lenguaje de programación.

Visual SINTAC soporta las estructuras de control más típicas. Las veremos una a una.

Toma de decisiones

If...Then...Else...EndIf

```
If condición Then  
...  
Else  
...  
EndIf
```

Esta estructura es una de las más básicas. Permite evaluar una condición y actuar de acuerdo a si es verdadera o falsa. Por ejemplo:

```
If (turnos>100) Then  
  Print("Se acabó.")  
  Exit()  
Else  
  Print("Turnos: " + turnos)  
EndIf
```


El código entre el **if** y el **Else** se ejecutan si la condición se cumple. Si esta no se cumple se ejecuta el código entre el **Else** y el **EndIf**.

El **Else** es opcional y puede no aparecer, por ejemplo:

```
If (turnos>50) Then
    Print("Cuidado, te quedan pocos turnos.")
EndIf
```

Select...Case...EndSelect

```
Select valor
    Case opc1
    ...
    Case opc2
    ...
    [Case *]
    ...
EndSelect
```

Se compara el valor con cada una de las opciones de los **Case**, y se ejecuta el código situado bajo la sentencia **Case** que coincida. Si no se encuentran coincidencias se ejecutará el código bajo el **Case *** (si aparece ya que es opcional). Ejemplo:

```
Select vida
    Case 1
        Print("Estás casi muerto.")
    Case 2
        Print("Cuidado, tu estado es crítico.")
    Case 3
        Print("Te encuentras muy mal.")
    Case 4
        Print("Has estado mejor en otras ocasiones pero aguantarás.")
    Case 5
        Print("Estás bastante bien.")
    Case *
        Print("Estás en plena forma.")
EndSelect
```

Estructuras de control

For...To...Next

```
For variable := valor1 To valor2
    ...
Next
```

Permite repetir la ejecución de un bloque de código un número determinado de veces. Se usa una variable contador que se incrementa desde el valor inicial (**valor1**) hasta el valor final (**valor2**) de uno en uno. Ejemplo:

```
For i := 1 To 10
    Print("...y ya van " + i)
Next
```

El código, en este caso, se ejecutará 10 veces y la variable **i** tomará los valores del 1 al 10.

While...Loop

```
While expresión
...
Loop
```

Permite ejecutar un bloque de código mientras se cumpla una condición. Ejemplo:

```
Declare(i)
i:=1
While i<10
    Print("Vamos " + i)
    i:=i+1
Loop
```

Salida inmediata

Exit: finaliza la ejecución del programa.

PROCEDIMIENTOS

En Visual SINTAC existen una serie de procedimientos definidos en el propio lenguaje. Un procedimiento se ejecuta llamándole por su nombre y pasándole (si es necesario) una serie de parámetros. Se puede obtener una referencia de los procedimientos definidos en Visual SINTAC consultando la ayuda.

***(i)** A veces se usará el término 'comando' para referirse a los procedimientos definidos en el propio lenguaje de programación.*

Un ejemplo de comando es **Print**. A este se le debe pasar el texto a imprimir, si no le pasamos nada imprimirá una línea en blanco. Ejemplo:

```
Print("Hola.")
Print(3 + 2)
Print()
```

Se puede observar claramente que los parámetros deben pasarse entre paréntesis y, si son varios, se separarán por comas, por ejemplo:

```
WindowPos(10,25)
```

Por supuesto, algunos procedimientos no reciben parámetros pero, en este caso, es necesario poner los paréntesis vacíos. Por ejemplo:

```
Pause()
```

Algunos procedimientos devuelven valores que podemos usar o no. Por ejemplo:

```
Declare(a)
a:=Input("Teclea algo: ")

If ArrayItem(lista,2)="dos" Then
```

```
        Print("DOS.")
    EndIf
```

También se pueden crear procedimientos propios. Los comandos del lenguaje que permiten esto son **Sub** y **Return**:

```
Sub Nombre_Procedimiento([par1,par2,...])
...
Return [valor]
```

El nombre de un procedimiento sigue las mismas restricciones que el de una variable. Ejemplos de declaración de procedimientos:

```
Sub Hola()
    Print("Hola.")
Return

Hola()      // imprimirá "Hola."

Sub Suma(valor1,valor2)
    Declare(resultado)

    resultado := valor1 + valor2
Return resultado

Suma(3,2)   // devolverá 5
```

13. MODULOS

Un módulo en Visual SINTAC es un conjunto de procedimientos agrupados. La división en módulos de los procedimientos es una forma conveniente de tener organizado nuestro código. Además los módulos de Visual SINTAC permiten la creación fácil de librerías de procedimientos.

Esto permite que se actualice fácilmente el código de la librería sin afectar al código de usuario. De la misma forma un usuario podría crear una serie de procedimientos y agruparlos en un módulo para distribuirlos de forma fácil.

El editor de módulos de Visual SINTAC permite:

- ✍ **Exportar un módulo:** el módulo se guardará en un fichero de extensión VX. Este fichero es un fichero ASCII normal que puede incluso editarse con cualquier editor de textos.
- ✍ **Importar un módulo:** se importa un fichero VX y se añade a los módulos de la aventura.
- ✍ **Sustituir módulo:** el sistema permite seleccionar un fichero VX y se sustituye el módulo que tenemos seleccionado en el editor de Visual SINTAC por el contenido del mismo. Esta opción se usará normalmente para sustituir módulos que hayan sido actualizados.

14. ESTRUCTURA DE UN PROGRAMA

Al contrario que en otros lenguajes de programación, el de Visual SINTAC no tiene ningún procedimiento específico donde se inicie la ejecución. En cambio, la ejecución se inicia en la primera línea del primer módulo y continúa hasta el final o hasta encontrar un comando de salida (**Exit**).

La estructura de un programa en Visual SINTAC es similar a esta:

```
Declare(a)           // declaración de variables globales
Declare(b)
Declare(c)

Instruccion 1         // inicio del código
Instruccion 2
Instruccion 3
...
Instruccion n
Exit                  // fin

// inicio de la declaración de procedimientos

Sub Procedimiento1()
...
Return

Sub Procedimiento2()
...
Return

Sub Procedimiento3()
...
Return
```

Por supuesto los procedimientos pueden estar en diferentes módulos como ya se ha visto.

15. VISIBILIDAD DE LAS VARIABLES

Cuando declaramos una variable, con la sentencia **Declare**, podemos hacerlo bien al inicio del programa, fuera de todos los procedimientos, o dentro de un procedimiento.

Las variables declaradas al inicio del programa, antes de la definición del primer procedimiento, se consideran globales. Son visibles desde su declaración hasta que finaliza el programa.

En cambio aquellas que se declaran dentro de un procedimiento sólo son visibles dentro de este y se destruyen al salir del mismo. Estas son variables locales.

En caso de ambigüedad (que nombremos una variable local igual que una global) tienen precedencia las variables locales sobre las globales.

16. LA LIBRERÍA ESTÁNDAR

Como ya se ha comentado, Visual SINTAC incorpora una aventura modelo (AvModelo) que contiene la librería estándar. La librería estándar incluye la lógica que da respuesta a los comandos básicos de toda aventura. Además de eso contiene el código necesario para inicializar la pantalla, controlar los turnos del jugador, mostrar la barra de estado, las descripciones de las localidades, el inventario del jugador y la interacción básica de los PSIs con el entorno.

Esta librería se ha desarrollado pensando en facilitar la tarea de programar una aventura a todo tipo de usuarios y a la vez intentar reducir el código que es necesario introducir. La aventura modelo está dividida en varios módulos:

- ✍ **Declaraciones:** contiene la definición de las variables y los parámetros necesarios que configuran ciertos aspectos de la aventura como la disposición de la pantalla, colores, el título de la aventura,...
- ✍ **Librería:** contiene todos los procedimientos básicos de la librería estándar, este módulo es el único que no necesita ser modificado por el usuario (a menos que se quieran modificar aspectos básicos de la librería).
- ✍ **Usuario:** contiene procedimientos de "enganche" de la librería. Estos procedimientos permiten introducir código que dará respuesta a acciones no contempladas en la librería sin tener que modificar esta. En este módulo se realizará la mayor parte de codificación que sea necesaria. Aquí se codificarán los comportamientos o acciones propias de cada aventura. Por ejemplo todas las aventuras disponen de la acción COGER, y como tal está codificada dentro del módulo Librería, pero no todas tienen la acción REGISTRAR y por lo tanto esta deberá ser codificada por el usuario, y el mejor lugar para hacer es precisamente este módulo.

17. MODULO DECLARACIONES

Este módulo consiste, básicamente, en una lista de declaraciones de variables globales usadas por la librería estándar:

- ✍ **turnos**: turnos de juego.
- ✍ **tipo_letra**: tipo de letra normal. (Ej.: "`\[Arial]\[10]`").
- ✍ **tipo_letra_info**: tipo de letra de barra de estado. (Ej.: "`\[Times New Roman]\[10]`").
- ✍ **color_fondo**: color de fondo de ventana de texto (Ej.: "`\[RGBF(000030)]`").
- ✍ **color_info**: color de fondo y texto de línea de estado. (Ej.: "`\[RGBF(FFFFFF)]\[RGB(000000)]`").
- ✍ **color_letra**: color de letra normal (Ej.: "`\[RGB(FFFFFF)]`").
- ✍ **dialogo_psi1**: texto que se imprime antes de lo que dice el PSI. (Ej.: " --- ").
- ✍ **dialogo_psi2**: texto que se imprime después de lo que dice el PSI. (Ej.: " --- ").
- ✍ **hay_graficos**: indica si la aventura tiene gráficos (puede ser **TRUE** o **FALSE**). Si está a **TRUE** se colocará una zona de gráficos justo encima de la de texto al estilo clásico de las conversacionales antiguas.
- ✍ **hay_sonido**: indica si la aventura tiene sonido (puede ser **TRUE** o **FALSE**).
- ✍ **todapantalla**: indica si la ventana principal ocupará toda la pantalla (puede ser **TRUE** o **FALSE**).
- ✍ **tamx**: ancho de la ventana (en pixels).
- ✍ **tamy**: alto de la ventana (en pixels).
- ✍ **posx**: posición X de la ventana (en pixels desde la esquina superior izquierda de la pantalla).
- ✍ **posy**: posición Y de la ventana (en pixels desde la esquina superior izquierda de la pantalla).
- ✍ **tamy_info**: altura de la ventana de estado (en pixels).

*(i) El tamaño de la zona de texto será el total de la ventana menos el tamaño de la ventana de estado y la de gráficos (si **hay_graficos** es **TRUE**).*

- ✍ **tamy_grf**: altura de la ventana de gráficos (en pixels).

(i) El tamaño de la zona de texto será el total de la ventana menos el tamaño de la ventana de estado y la de gráficos (si `hay_graficos` es `TRUE`).

✍ **prompt:** prompt del input que se muestra al jugador. (Ej.: "> ")

✍ **título:** título de la aventura (se muestra en el título de la ventana).

✍ **version:** versión de la aventura. (Ej.: "2.0")

La distribución de la pantalla es la clásica de las conversacionales:

✍ Una barra de estado donde se muestra la descripción corta de la localidad actual.

✍ Una zona de gráficos donde se presentará el gráfico de la localidad actual. Esta zona sólo existirá si ponemos `hay_graficos` a `TRUE`.

✍ Una zona de texto donde aparecerán las descripciones de las localidades y la respuesta a las acciones del jugador.

El tamaño total de estas zona estará determinado por el tamaño que hayamos definido para la ventana donde se ejecuta la aventura (`tamx`, `tamy`).

En este módulo también colocaremos la declaración de las variables globales que necesitemos definir para nuestra aventura en particular.

(i) `TRUE` y `FALSE` son dos variables globales, definidas en el módulo *Declaraciones*, que contienen los valores 1 y 0 respectivamente. Visual SINTAC usa el 0 para representar el valor falso y el 1 para representar el verdadero en las operaciones lógicas.

MODULO USUARIO

En este módulo se encuentran definidos una serie de procedimientos que permiten enganchar código a la librería estándar. Estos procedimientos de enganche permiten o bien definir acciones nuevas, no contempladas en la librería, o bien contemplar casos especiales sobre las acciones definidas.


No se van a documentar aquí todos los procedimientos de enganche ya que se encuentran documentados ampliamente dentro del propio módulo, con ejemplos de uso.

Sí que nos vamos a centrar en completar nuestro tutorial incluyendo el código necesario para la aventura que venimos desarrollando.





PROPIEDADES DE USUARIO USADAS POR LA LIBRERÍA ESTÁNDAR

La librería estándar necesita que las localidades, objetos y PSIs tengan definidas una serie de propiedades de usuario que pasamos a enumerar a continuación:






Localidades:

-  **ESCUCHA:** cadena de texto que se imprimirá cuando el jugador teclee OIR o ESCUCHAR en esa localidad, si se deja en blanco la librería imprime un mensaje estándar.

Objetos:

-  **SABOR:** cadena de texto que describe el sabor del objeto cuando el jugador intenta SABOREARLO o CHUPARLO. Si se deja en blanco la librería imprime un mensaje estándar.
-  **OLOR:** cadena de texto que describe el olor del objeto cuando el jugador intenta OLERLO. Si se deja en blanco la librería imprime un mensaje estándar.
-  **TACTO:** cadena de texto que describe el tacto del objeto cuando el jugador intenta TOCARLO o ACARICIARLO. Si se deja en blanco la librería imprime un mensaje estándar.
-  **ESCUCHA:** cadena de texto que describe el sonido del objeto cuando el jugador intenta ESCUCHARLO. Si se deja en blanco la librería imprime un mensaje estándar.

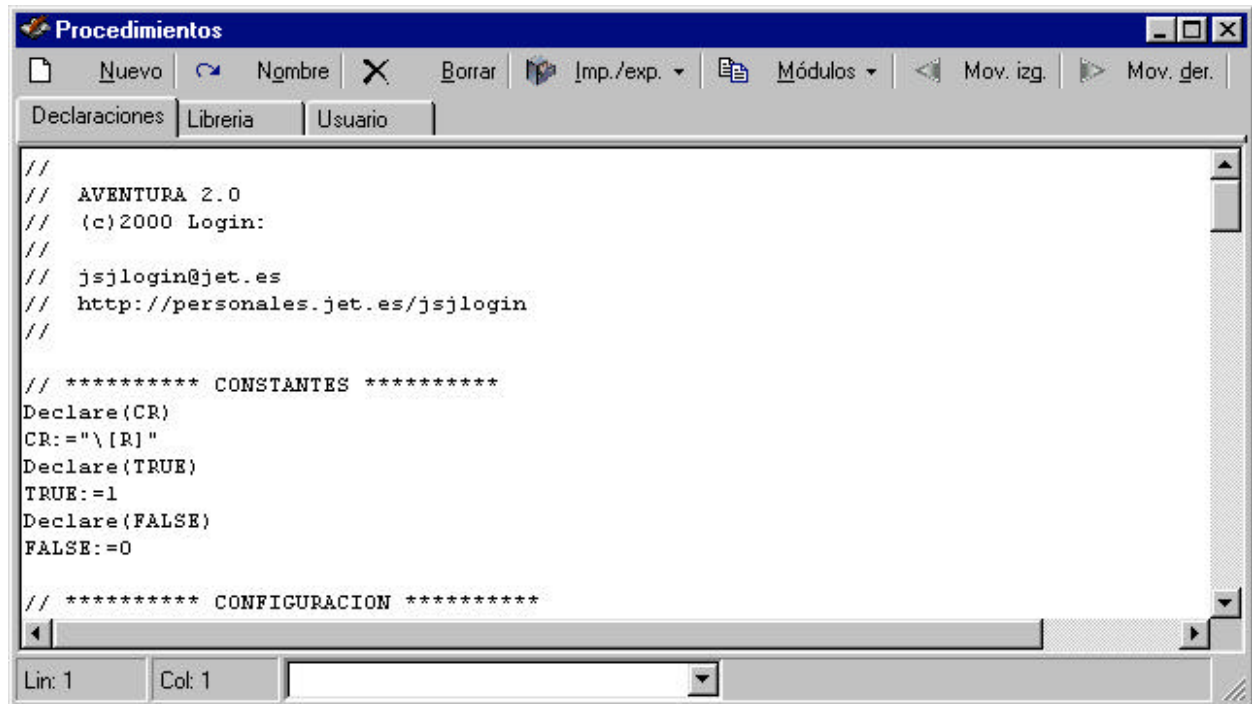
PSIs:

-  **MOV_RUTA:** puede tomar alguno de los siguientes valores:
 - ? La cadena "SEGUIR" si se desea que el PSI siga al jugador.
 - ? Un array de verbos de movimiento que describen la ruta del PSI.
-  **MOV_PASO:** contiene el índice del array del siguiente paso de la ruta en caso de que **MOV_RUTA** contenga un array de verbos de movimiento describiendo la ruta del PSI. En este último caso debería inicializarse a 1.
-  **MOV_REPETIR:** si es 1 el PSI repetirá la ruta desde el principio, cuando se llegue al final de la misma, para ello se debería definir una ruta circular, es decir, que acabe en la misma localidad donde se inició. Si es 0 el PSI se parará cuando complete la ruta.
-  **SABOR:** cadena de texto que se muestra cuando el jugador intenta CHUPAR al PSI. Si se deja en blanco la librería imprime un mensaje estándar.
-  **OLOR:** cadena de texto que se muestra cuando el jugador intenta OLER al PSI. Si se deja en blanco la librería imprime un mensaje estándar.

- ✍ **TACTO:** cadena de texto que se muestra cuando el jugador intenta TOCAR o ACARICIAR al PSI. Si se deja en blanco la librería imprime un mensaje estándar.
- ✍ **ESCUCHA:** cadena de texto se muestra cuando el jugador intenta ESCUCHAR al PSI. Si se deja en blanco la librería imprime un mensaje estándar.

18. TUTOR: CODIFICANDO

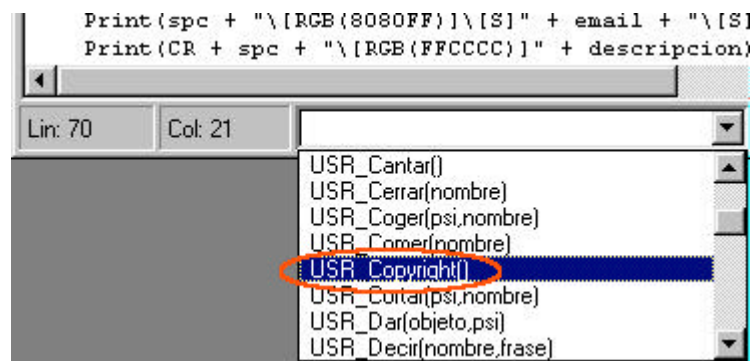
Abrimos el editor de procedimientos (Aventura? Procedimientos).



Primero personalizaremos algunos aspectos de nuestra aplicación. Dentro del módulo Declaraciones (que es el que se abre por defecto), buscaremos y cambiaremos las siguientes líneas:

```
titulo:="Mi Aventura"
version:="1.0"
```

Ahora nos situamos en el módulo Usuario para lo cual pinchamos sobre la pestaña correspondiente. Lo primero que vamos a hacer es modificar el mensaje de copyright que sale al iniciar la aventura. Para ello, buscaremos el procedimiento `USR_Copyright`. Lo podemos hacer usando el desplegable de selección de procedimientos:



Este procedimiento contiene el mensaje estándar que muestra el título de la aventura y el copyright, a parte de otra información como las versiones de la aventura y el intérprete y datos sobre el autor. Lo modificaremos de acuerdo a nuestros gustos. Por ejemplo:

```
Sub USR_Copyright()
    Declare(spc)
    Declare(copyright)
    Declare(email)
    Declare(web)
    Declare(descripcion)

    spc:="          "
    copyright:="(c)2000 Pepito Aventurero"
    email:="pepitoaventurero@correo.es"
    web:="http://www.pepitoaventurero.com"
    descripcion:="Mi primera aventura."

    Print(CR + "          \[Times New Roman]\[12]\[RGB(FFFFCC)]\[N]" + titulo + "\[N]"
+ CR)
    Print("\[10]\[C]" + spc + copyright + "\[C]" + CR)
    Print(spc + "\[RGB(8080FF)]\[S]" + email + "\[S]" + web)
    Print(CR + spc + "\[RGB(FFCCCC)]" + descripcion)
    Print(CR + spc + "Versión " + version + " / intérprete " + VerVS)
    // inicializamos el tipo de letra y color del texto
    Print(tipo_letra + color_letra + CR + CR)

Return
```

Las líneas marcadas son las que hemos modificado. Si ejecutásemos ahora la aventura podríamos ver qué aspecto va tomando todo. Para ello usaremos la opción Aventura? Ejecutar.



En los siguientes tutoriales codificaremos algunas acciones propias de nuestra aventura.

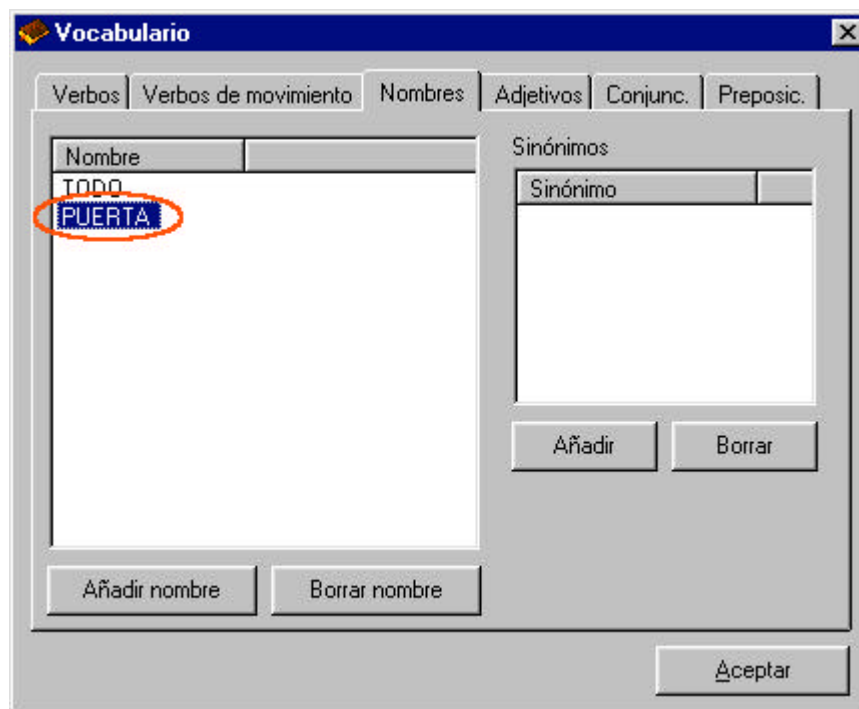
19. TUTOR: LA PUERTA

La casa tiene una puerta que inicialmente está cerrada ya que, cuando creamos las conexiones, decidimos cerrar la conexión entre la casa y la calle.

Las acciones que afectan a la puerta, y que deberemos codificar, básicamente son las siguientes:

```
EXAMINAR PUERTA
ABRIR PUERTA
CERRAR PUERTA
```

Primero tenemos que definir el nombre PUERTA en el vocabulario para que Visual SINTAC lo reconozca.



Una vez hecho esto iremos al editor de procedimientos, nos situaremos en el módulo Usuario y buscaremos el procedimiento `USR_Examinar`. Este procedimiento nos permitirá contemplar casos especiales del comando EXAMINAR como la puerta de este ejemplo.

Justo detrás de la línea que pone `ret:=FALSE` vamos a añadir el siguiente código:

```
If (PSI[PSIJugador].Localidad="CASA") | (PSI[PSIJugador].Localidad="CALLE") Then
  Select ParseNombre1
    Case "PUERTA"
      Print("Es de madera. ")
      If LOC["CASA"].Abierta("SALIR") Then
        Print("Está abierta." + CR)
      Else
        Print("Está cerrada." + CR)
      EndIf
      ret:=TRUE
    EndSelect
```

```
EndIf
```

La primera línea:

```
If (PSI[PSIJugador].Localidad="CASA") | (PSI[PSIJugador].Localidad="CALLE") Then
```

comprueba que estemos en la casa o en la calle (las dos localidades desde donde podemos ver la puerta). Luego comprobamos si el jugador se refiere a la puerta comprobando la variable **ParseNombre1** que contiene el nombre del comando que ha tecleado el jugador.

Para comprobar si la puerta está abierta o cerrada, miramos el estado de la conexión entre la casa y la calle:

```
If LOC["CASA"].Abierta("SALIR") Then
```

podríamos comprobar la conexión de la calle a la casa pero como vamos a abrirla y cerrarlas simultáneamente, es indiferente.

Para abrir la puerta tenemos que localizar el procedimiento **USR_Abrir** e insertaremos el siguiente código justo detrás de la línea **ret:=FALSE**:

```
If nombre="PUERTA" Then
  Select PSI[PSIJugador].Localidad
    Case "CASA"
      If LOC["CASA"].Abierta("SALIR") Then
        Print("La puerta ya está abierta." + CR)
      Else
        Print("Abres la puerta." + CR)
        LOC["CASA"].Abrir("SALIR")
        LOC["CALLE"].Abrir("ENTRAR")
      EndIf
      ret:=TRUE
    Case "CALLE"
      If LOC["CALLE"].Abierta("ENTRAR") Then
        Print("La puerta ya está abierta." + CR)
      Else
        Print("Abres la puerta." + CR)
        LOC["CALLE"].Abrir("ENTRAR")
        LOC["CASA"].Abrir("SALIR")
      EndIf
      ret:=TRUE
  EndSelect
EndIf
```

Este código controla la posibilidad de que abramos la puerta desde cualquiera de sus dos lados. Es importante observar que cuando abrimos la puerta tenemos que abrir la conexión entre la casa y la calle por sus dos extremos.

Ahora haremos lo mismo para la acción **CERRAR PUERTA**. Buscamos el procedimiento **USR_Cerrar** e introducimos el siguiente código (siempre detrás de la línea **ret:=FALSE**):

```
If nombre="PUERTA" Then
  Select PSI[PSIJugador].Localidad
    Case "CASA"
      If LOC["CASA"].Abierta("SALIR") Then
        Print("Cierras la puerta." + CR)
        LOC["CASA"].Cerrar("SALIR")
        LOC["CALLE"].Cerrar("ENTRAR")
      Else
```

```
        Print("La puerta ya está cerrada." + CR)
    EndIf
    ret:=TRUE
Case "CALLE"
    If LOC["CALLE"].Abierta("ENTRAR") Then
        Print("Cierras la puerta." + CR)
        LOC["CASA"].Cerrar("SALIR")
        LOC["CALLE"].Cerrar("ENTRAR")
    Else
        Print("La puerta ya está cerrada." + CR)
    EndIf
    ret:=TRUE
EndSelect
EndIf
```


20. TUTOR: LA TELEVISION

En el salón hay una televisión que deberíamos poder encender y apagar para ver qué programación están poniendo. En este caso debemos codificar tres posibles acciones:

```
ENCENDER TELEVISION
APAGAR TELEVISION
EXAMINAR TELEVISION
```

Primero introduciremos, en el procedimiento `USR_Encender`, el código que permite encender la televisión (como siempre después de la línea `ret:=FALSE`):

```
If PSI[PSIJugador].Localidad="CASA" Then
  Select nombre
    Case "TELEVISION"
      If OBJ[nombre].Encendido Then
        Print("La televisión ya está encendida." + CR)
      Else
        Print("Enciendes la televisión." + CR)
        OBJ[nombre].Encendido:=TRUE
      EndIf
      ret:=TRUE
    EndSelect
  EndIf
```

Para poder apagar la televisión tenemos que insertar el siguiente código en el procedimiento `USR_Apagar`:

```
If PSI[PSIJugador].Localidad="CASA" Then
  Select nombre
    Case "TELEVISION"
      If OBJ[nombre].Encendido Then
        Print("Apagas la televisión." + CR)
        OBJ[nombre].Encendido:=FALSE
      Else
        Print("La televisión ya está apagada." + CR)
      EndIf
      ret:=TRUE
    EndSelect
  EndIf
```

Sólo nos queda añadir algún texto cuando examinemos la televisión. Al tratarse la televisión de un objeto y tener ya su propia descripción, lo que haremos es añadir este texto a la descripción propia del objeto. Esto lo podemos hacer dentro del procedimiento `USR_ExaminarDetalles`, lo haremos insertando el siguiente código entre las líneas:

```
txt:=" "
```

y:

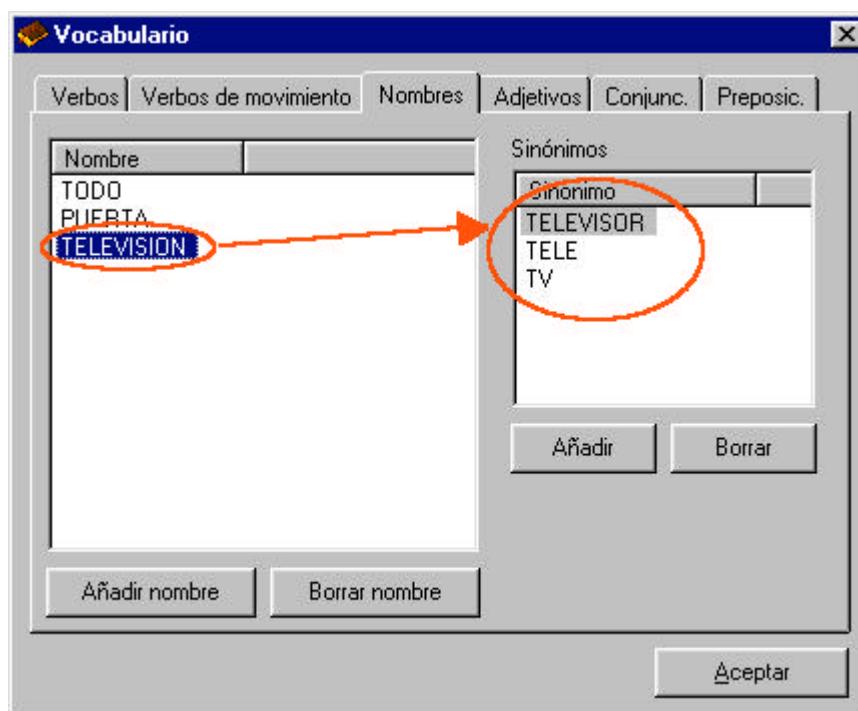
```
If txt<>" " Then
  txt:=" " + txt
EndIf
```

el código a insertar es:

```
If Not(psi) Then
  If nombre="TELEVISION" Then
    If OBJ[nombre].Encendido Then
      txt:="Imágenes de una película de acción parpadean en la vieja
pantalla del televisor."
    Else
      txt:="La pantalla aparece negra, seguramente porque la televisión
está apagada."
    EndIf
  EndIf
EndIf
EndIf
```

Hay que observar que lo primero que hacemos es comprobar que no se está refiriendo a un PSI ya que en este procedimiento se entrará tanto si el jugador examina a un PSI como a un objeto.

Para terminar con la televisión, sería interesante añadir unos cuantos sinónimos para que el jugador pueda teclear TELE, TELEVISOR o TV al referirse a ella. Si queremos hacer esto debemos insertar el nombre TELEVISION en el vocabulario y asignarle esos sinónimos.



21. TUTOR: UN PSI

Siempre se ha considerado que una de las cosas más complicadas al escribir una aventura conversacional es conseguir implementar unos personajes mínimamente creíbles dentro de la misma. Por eso he dejado este tutor para el final. Vamos a implementar un PSI simple (bueno, quizá no tan simple). Como personaje se ha elegido un animal, un gato.

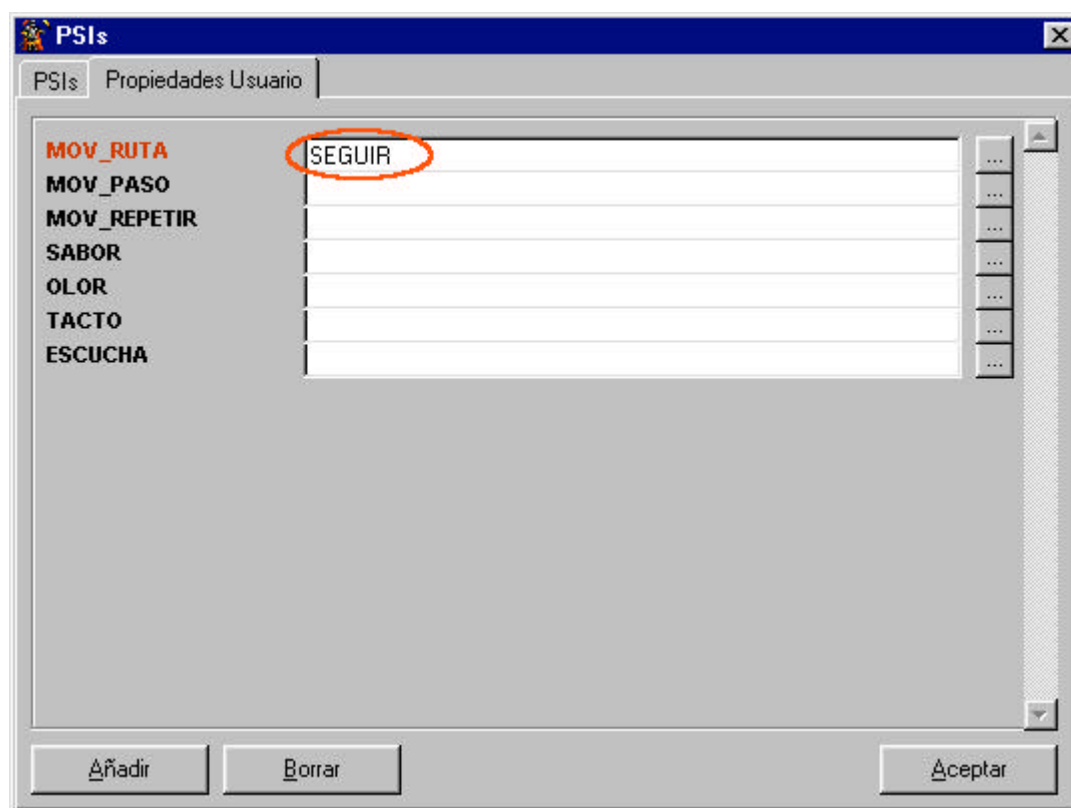
Primero definiremos al PIS. Vamos al editor de PSIs e insertamos uno nuevo cuyo nombre va a ser GATO, con las siguientes características:

✍ **Descripción corta:** El gato Malquías

✍ **Descripción larga:** Es un lindo gatito.

✍ **Loc. inicial:** CASA

Si ejecutamos la aventura tendremos ya al gato dentro de la casa. Fácil, ¿no?. Bueno... pero este gato parecerá de adorno porque ni siquiera se mueve. Haremos que nos siga a donde vayamos. Para ello, debido a que la librería estándar contempla esta posibilidad, simplemente abriremos el editor de PSIs, seleccionaremos al gato y en la propiedad de usuario **MOV_RUTA** introduciremos **SEGUIR**.

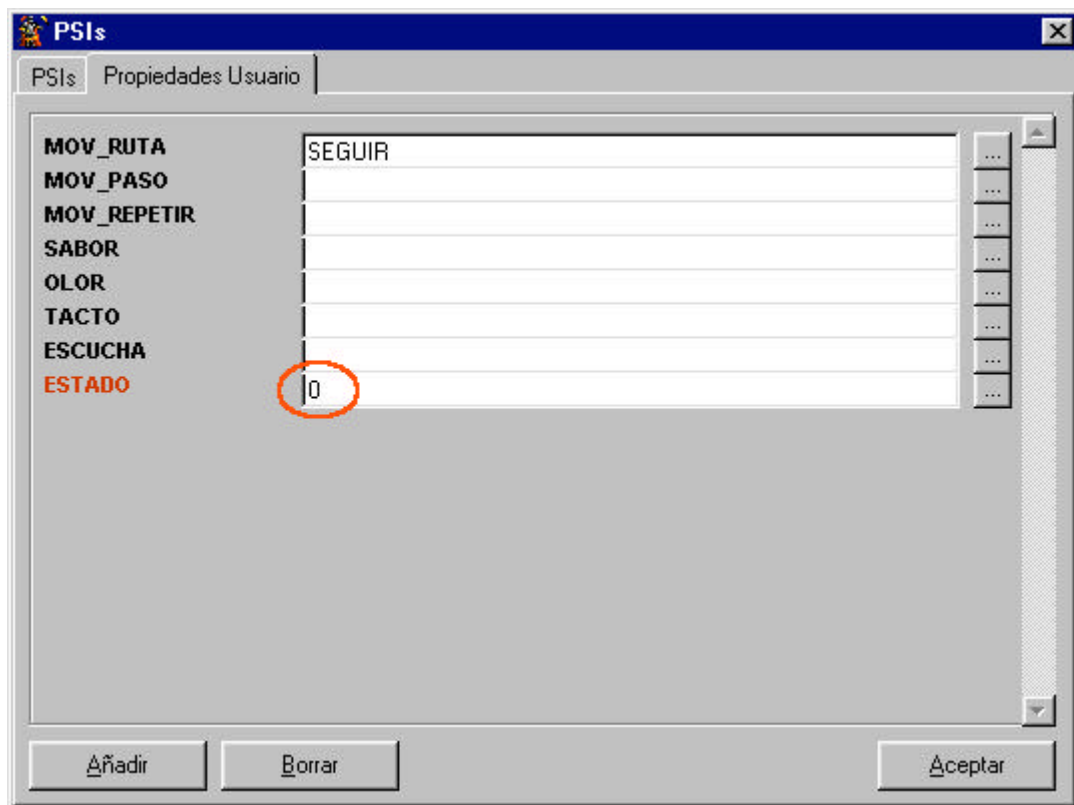


Esto hará que el gato nos siga allá donde vayamos. Seguidamente vamos a hacer que, como buen gato que es, maúlle un poco. Decidimos que dará un maullido cada 2 turnos. Hay un par de procedimientos de usuario que se ejecutan cada turno del jugador y son:

✍ **USR_Ant:** se ejecuta después de analizar el comando tecleado pero antes de que se ejecute la acción asociada (si hubiese alguna).

✍ **USR_Post:** se ejecuta después de procesar el comando tecleado por el jugador.

Necesitamos una nueva propiedad de usuario a los PSIs, a la que llamaremos ESTADO y que nos servirá para controlar el estado del gato malaquías, es decir, cuando le tocará maullar.



De los dos procedimientos nombrados, el más adecuado para lo que queremos hacer es **USR_Post**, así que en este insertaremos el siguiente código:

```
If PSI["GATO"].Estado=0 Then
    PSI["GATO"].Estado:=1
Else
    PSI["GATO"].Estado:=0
    If PSI["GATO"].Localidad=PSI[PSIJugador].Localidad Then
        HablaPSI("MIAUUUUUU...", "")
    EndIf
EndIf
```

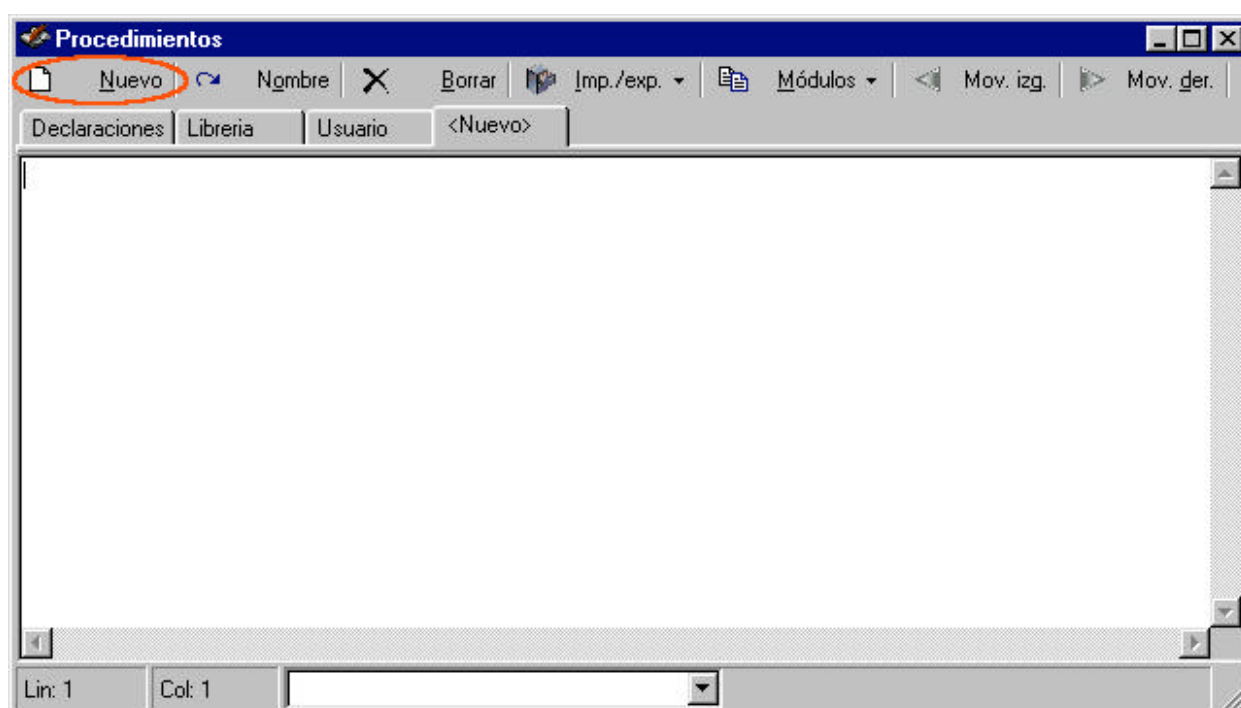
Ya hemos conseguido un sencillo PSI pero... ¿habéis probado a teclear DECIR A GATO "HOLA"? Curiosamente, y contra todas las leyes gatunas, el gato ¡¡¡nos responde!!!. Esto es debido a que la librería estándar contempla diálogo con los PSIs pero, en este caso, resulta una situación absurda.

Pues nada, haremos que el gato no hable. Insertaremos el siguiente código en el procedimiento **USR_Decir** (recordemos que se debe insertar detrás de la línea **ret:=FALSE**):

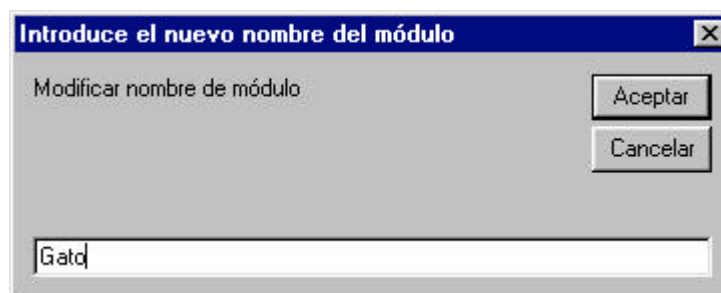
```
If nombre="GATO" Then
    GATO_Habla(frase)
    ret:=TRUE
EndIf
```

No queremos que el gato hable pero sí que sería interesante que entendiese algunas órdenes básicas. Vamos a crear un procedimiento nuevo, al que llamaremos **GATO_Habla**, que dé respuesta a lo que le digamos al gato; por supuesto el gato entenderá sólo unos cuantos comandos muy básicos. Esto servirá como ejercicio de programación de conversaciones de PSIs.

El nuevo procedimiento lo añadiremos en un nuevo módulo, ya que no es conveniente recargar en exceso el módulo Usuario con más procedimientos y, además, es buena costumbre organizar un poco nuestro código. El nuevo módulo lo añadiremos con el comando Nuevo del editor de procedimientos.



El nombre por defecto del nuevo módulo es poco descriptivo así que lo renombraremos con el comando Nombre del editor de procedimientos. El nombre de este módulo será Gato.



Dentro de este nuevo módulo vamos a añadir el procedimiento **GATO_Habla**, este procedimiento recibirá la frase tecleada entre comillas y dará respuesta si es alguna de las que el gato entiende.

```

Sub GATO_Habla(frase)

    Parse(frase)
    If EsVerboMov Then
        If PSI["GATO"].Mover(ParseVerbo) Then
            Print(PSI["GATO"].DescCorta + " " + Direccion(ParseVerbo) + "." + CR)
            PSI["GATO"].Mov_Ruta:=""
        Else
            HablaPSI("MIAU-MIAU-MIAU", "")
        EndIf
    Else
        HablaPSI("¿MARRA-MIAU-MIAU?", "")
    EndIf

Return

```

El gato entenderá los verbos de movimiento y se dirigirá hacia la dirección que le indiquemos. Para ello "parseamos" la frase entrecomillada con el comando **Parse** y comprobamos si es o no un verbo de movimiento mediante la variable **EsVerboMov**. Si es un verbo de movimiento intentamos mover al gato en esa dirección y además vaciamos la propiedad de usuario **MOV_RUTA** del gato para que ya no nos siga a todas partes.

Se hace una llamada a un procedimiento denominado **Direccion** que se encargará de imprimir el texto correspondiente a la dirección en la que se mueve el gato. Este procedimiento lo insertaremos también en el módulo Gato y su código es el siguiente:

```

Sub Direccion(mov)
    Declare(txt)

    Select Ucase(mov)
        Case "NORTE"
            txt:="va hacia el norte"
        Case "SUR"
            txt:="va hacia el sur"
        Case "ESTE"
            txt:="va hacia el este"
        Case "OESTE"
            txt:="va hacia el oeste"
        Case "NORESTE"
            txt:="va hacia el noreste"
        Case "NOROESTE"
            txt:="va hacia el noroeste"
        Case "SURESTE"
            txt:="va hacia el sureste"
        Case "SUROESTE"
            txt:="va hacia el suroeste"
        Case "SUBIR"
            txt:="sube"
        Case "BAJAR"
            txt:="baja"
        Case "ENTRAR"
            txt:="entra"
        Case "SALIR"
            txt:="sale"
        Case *
            txt:="se va"
    EndSelect

Return txt

```

Con esto ya tenemos a nuestro gato. No es un PSI muy complejo pero sirve de ejemplo de las técnicas básicas a seguir a la hora de programar personajes.

22. GRAFICOS, SONIDOS Y TIPOS DE LETRA

Visual SINTAC permite incluir elementos multimedia en la aventura; podemos incluir gráficos, sonidos y tipos de letra.

Los comandos que manipulan gráficos son los siguientes (para una descripción más detallada se debe recurrir a la ayuda en línea de Visual SINTAC):

```
WindowBackground(fondo)
ScreenImg(pantalla, imagen)
```

Los comandos de sonido disponibles son los siguientes (para una descripción más detallada se debe recurrir a la ayuda en línea de Visual SINTAC):

```
LoadSong(modulo)
PlaySong(IDmodulo)
UnloadSong(IDmodulo)
Volume(volumen)
LoadWav(wav)
PlayWav(IDwav, frecuencia, volumen)
UnloadWav(IDwav)
```

Los comandos que permiten cargar disitintos tipos de letra son los siguientes (para una descripción más detallada se debe recurrir a la ayuda en línea de Visual SINTAC):

```
Font(fuente)
```

En algunos de ellos hay que especificar como uno de sus parámetros el origen de los datos, es decir, dónde se ubica la imagen, el sonido o el tipo de letra que queremos cargar. El origen de los datos puede ser de dos tipos:

✍ El nombre de un fichero del disco.

✍ Un identificador de recurso.

En el caso de que se trate de un fichero de disco, desde el cual queremos que se cargue la imagen, el sonido o el tipo de letra, debemos pasar la ruta completa al mismo, por ejemplo:

```
ScreenImg("c:\imagenes\img001.jpg")
LoadSong("sonido.mod")
LoadWav("pajaro.wav")
Font("c:\av\letras\cour2.ttf")
```

Si los ficheros están ubicado en el mismo directorio que la aventura se puede usar la variable global **RutaDat**, por ejemplo:

```
ScreenImg(RutaDat + "\img001.jpg")
LoadSong(RutaDat + "\sonidos\cancion.s3m")
```

En cualquier caso una forma más conveniente de guardar los elementos multimedia de la aventura es usando recursos.

Desde código haremos referencia a un recurso mediante su identificador, anteponiendo el carácter almohadilla (#). Así #10 hará referencia al recurso 10. Dependiendo del contexto Visual SINTAC sabe distinguir a qué tipo de recurso nos referimos. Así, si es dentro de un comando de sonido, sabrá que tiene que buscar un recurso de tipo SND, y si es dentro de un comando de gráficos buscará un recurso de tipo IMG.

Para que Visual SINTAC reconozca los recursos, cuando nos referimos a ellos por su identificador, debemos compilarlos. Esto se hace desde dentro del propio editor de recursos usando el botón Compilar. Debemos compilar los recursos antes de ejecutar la aventura desde dentro del entorno, si hemos hecho cambios en estos.

Al compilar los recursos se genera un fichero con el mismo nombre que la aventura pero con extensión VSR que contiene los recursos compilados.

Los comandos que pueden usar recursos son todos los relativos a gráficos, sonidos y tipos de letra. Por ejemplo:

```
ScreenImg( "#10" )  
LoadSong( "#10" )
```

24. COMPILAR Y DISTRIBUIR LA AVENTURA

Mediante la opción Aventura? Compilar aventura podemos generar un fichero que podemos distribuir.

Cuando se compila la aventura se genera un fichero VSR que además de todos los recursos contiene el 'ejecutable' de la misma. Este fichero no se debe confundir con el VSR que se genera al compilar los recursos. Al compilar los recursos se genera un VSR que contiene únicamente los recursos pero no el 'ejecutable' de la aventura. Para distribuir la aventura la debemos compilar mediante la opción 'Compilar aventura'.

El fichero VSR procedente de la compilación de la aventura necesita del intérprete para ejecutarse. Básicamente el usuario de nuestra aventura necesitará lo siguiente:

- ✍ El 'runtime' mini que se puede descargar de <http://pagina.de/jsj>. También puede usarse el 'runtime' completo pero si sólo se pretende ejecutar aventuras y no programarlas es más conveniente descargar el 'mini'.
- ✍ El intérprete (**InterpreteVS.exe** y **VS.dll**) también disponible para descarga en la web.
- ✍ El fichero VSR que contiene nuestra aventura y todos los recursos que esta utiliza.

No es mala idea distribuir los ficheros **InterpreteVS.exe**, **VS.dll** y el runtime 'mini' con nuestra aventura. Debido a que el runtime 'mini' puede ocupar demasiado, dependiendo de cómo distribuyamos la aventura, podemos obviar este y en su lugar indicar la URL desde dónde se puede descargar.