Diseñando juegos con el Game Maker

versión 5.0 (Abril 14, 2003)

Written by Mark Overmars

Traducción (versión sin revisar) al español por:

José Jorge Enríquez rodríguez (Geo)
Carlos A. García Ríos (Lestad)
Gabriel Rojas (Faxtrax)
Israel Rodríguez Nava (zurdopower)
Rodrigo Espinoza Huerta (rodeh85)

Contenido

Capítulo 1 Así que quieres crear tus propios juegos de computadora

Jugar juegos de computadora es divertido. Pero es más divertido diseñar tus propios juegos y que otros los jueguen. Desafortunadamente, no es fácil crear videojuegos para computadora. Los juegos comerciales de hoy en día normalmente se llevan de uno a tres años de desarrollo con equipos de entre 10 a 50 personas. Sus presupuestos alcanzan fácilmente millones de dólares. Y todas estas personas tienen bastante experiencia: programadores, diseñadores de arte, técnicos de sonido, etc.

¿Quiere decir que es imposible crear tus propios juegos para computadora? Afortunadamente no. Por supuesto que no debes esperar que puedas crear tu propio Quake o Age of Empires en unas cuantas semanas. Pero tampoco es necesario. Juegos más simples, como Tetris, Pacman, Space Invaders, etc., también son divertidos y son mucho más fáciles de crear. Desafortunadamente aún requieren de buenas habilidades de programación para manejar los gráficos, sonidos, la interacción con el usuario, etc.

Es aquí donde aparece el *Game Maker*. El *Game Maker* ha sido creado para hacer mucho más sencilla la creación de ese tipo de juegos. Sin ninguna necesidad de programar. Una sencilla e intuitiva interfaz de arrastrar y soltar te permite crear tus propios juegos rápidamente. Puedes importar y crear imágenes, sprites (imágenes animadas), sonidos y usarlos. Tú defines fácilmente los objetos en tu juego e indicas su comportamiento. Puedes definir niveles atractivos con fondos deslizables en donde tome lugar el juego. Y si quieres control total sobre el juego existe un sencillo lenguaje de programación dentro del *Game Maker* que te pone al mando de todo lo que sucede en el juego.

El *Game Maker* se enfoca a juegos bidimensionales. Así que nada de mundos 3D como en Quake. Pero no te desanimes. Muchos juegos importantes, como Age of Empires, las series Command and Conquer y Diablo usan tecnología de sprites bidimensionales, aunque parecen tridimensionales. Y diseñar juegos bidimensionales es mucho más fácil y rápido.

Probablemente la mejor parte es que el *Game Maker* puede usarse libre de cargo. Y no hay ninguna restricción en los juegos que creas con él. Ninguna pantalla de publicidad, e incluso puedes venderlos si así lo deseas. Checa el acuerdo de licencia adjunto para más detalles.

Este documento te dirá todo lo que necesitas saber acerca del *Game Maker* y cómo puedes crear tus propios juegos con él. Por favor entiende que, incluso con un programa como *Game Maker*, el diseño de juegos para computadora no es completamente trivial. Hay muchísimos aspectos que son importantes: el game play, los gráficos, los sonidos, la interacción con el usuario, etc. Comienza con ejemplos fáciles y te darás cuenta de que crear juegos es bastante divertido. También checa el sitio web

http://www.cs.uu.nl/~markov/gmaker/index.html (en inglés)

y el foro allí para obtener muchos ejemplos, ideas y ayuda. Y pronto te convertirás en un maestro de la creación de videojuegos. Disfrútalo.

Capítulo 2 Instalación

Probablemente ya lo hiciste pero por si acaso, aquí tienes cómo instalar el *Game Maker*. Simplemente ejecuta el programa `gmaker.exe`. Sigue las instrucciones en pantalla. Puedes instalar el programa donde lo desees pero mejor deja las opciones por defecto. Una vez que finalice la instalación, en el menú Inicio encontrarás un nuevo grupo de programas desde el que puedes ejecutar el *Game Maker* y leer la documentación. Además del programa *Game Maker* también se instala la documentación, junto con el archivo de ayuda.

La primera vez que ejecutes el *Game Maker* se te preguntará si quieres ejecutarlo en modo **Simple** o **Advanced (avanzado)**. Si no has empleado un programa de creación de videojuegos antes y no tienes experiencia como programador, usa el modo simple (selecciona **No)** En el modo simple se muestran menos opciones. Puedes cambiar fácilmente al modo avanzado usando la opción en el menú **File**.

Dentro de la carpeta de instalación (por defecto C:\Archivos de Programa\Game_Maker5) tenemos varias carpetas adicionales:

- `examples`: contiene varios juegos de ejemplo, para que los revises y/o modifiques.
- `lib`: contiene varias librerías de acciones. Si deseas instalar librerías adicionales debes colocarlas dentro de esta carpeta.
- `sprites`: esta carpeta está dedicada a contener los sprites que uses. La instalación incluye algunos sprites, pero en el sitio del *Game Maker* (http://www.gamemaker.nl), puedes descargar varios paquetes de recursos que contienen sprites, sonidos, y fondos adicionales.
- `backgrounds`, `sounds`: carpetas similares que contienen las imágenes para los fondos y los sonidos.

El *Game Maker* requiere una PC moderna con procesador Pentium y sistema operativo Windows 98, NT, 2000, Me, o superior. Requiere una resolución de pantalla de por lo menos 800x600 y 65000 (16-bit) colores. Requiere DirectX. Cuando se diseñan y prueban los juegos, los requerimientos de memoria son de por lo menos 32 MB (preferiblemente más). Cuando solo se están ejecutando los juegos, los requerimientos de memoria son un poco menores pero depende mucho del tipo de juego.

Capítulo 3 Registro

Como se mencionó antes, el *Game Maker* puede ser usado sin cargo alguno. No hay restricción alguna para los juegos que crees. En los juegos no aparece ninguna pantalla publicitaria e inclusive puedes vender tus juegos si así gustas. Lee el acuerdo de licencia adjunto para más detalles.

Pero se te recomienda ampliamente que registres tu copia del *Game Maker*. Mediante el registro ayudas al futuro desarrollo del programa. También se eliminará la pantalla publicitaria en el editor. Se planean futuros beneficios para los usuarios registrados, por ejemplo una competición de juegos.

La cuota de registro del *Game Maker* es de US $15 o de 15 Euros. Hay varias formas en las que puedes llevar a cabo el registro de tu copia del programa. La más fácil es usar el registro online usando un sistema de pago seguro con tarjeta de crédito o una cuenta PayPal. Alternativamente puedes realizar una transferencia a nuestra cuenta bancaria, enviarnos una money order o enviar efectivo. Puedes encontrar los detalles en el sitio de registro del *Game Maker*:

www.gamemaker.nl/registration.html

Para registrar tu copia del *Game Maker* usa el sitio de arriba o selecciona **Registration** desde el menú **Help**. En la parte inferior de la ventana que aparece haz clic en el botón **Registration**. Serás llevado a nuestra página web donde se indican las diferentes opciones de registro, incluyendo el registro online.

Una vez que tu registro ha sido recibido se te enviará un email con el nombre, clave y la información sobre cómo introducir la clave en el programa. Para introducir la clave, selecciona nuevamente **Registration** desde el menú **Help**. En la parte inferior de la ventana presiona el botón **Enter Key**. Introduce el nombre y la clave, y presiona **OK**. Si seguiste estos pasos correctamente el programa ya estará registrado.

Capítulo 4 La idea global

Antes de ahondar en las posibilidades del *Game Maker*, sería bueno tener una idea general acerca del programa. Los juegos creados con el *Game Maker* se llevan a cabo en una o más *habitaciones* (r*ooms*). (Las habitaciones son planas, no 3D, pero pueden contener imágenes con apariencia 3D). En estas habitaciones es donde colocas los objetos, los cuales puedes definir en el programa. Los objetos típicos son las paredes, objetos móviles, el personaje principal, enemigos, etc. Algunos objetos, como las paredes, sólo se encuentran ahí y no realizan ninguna acción. Otros, como el personaje principal, se moverán y reaccionarán a los comandos del jugador (teclado, ratón, joystick). Por ejemplo, cuando el personaje principal toca un enemigo podría ser eliminado. Los objetos son los ingredientes más importantes de los juegos creados con el *Game Maker*, por lo que hablaremos un poco más sobre ellos.

Primero que nada, la mayoría de los objetos necesitan alguna imagen para hacerse visibles en la pantalla. Tales imágenes son llamadas *sprites*. Un sprite no siempre es una sola imagen sino una serie de ellas que se muestran una tras otra para crear una animación. De esta forma parece que el personaje camina, que una pelota rueda, que una nave explota, etc. Durante el juego el sprite de un objeto en particular puede cambiar. (Así el personaje puede lucir diferente cuando camina a la izquierda o a la derecha). Puedes crear tus propios sprites en el *Game Maker* o cargarlos de algún archivo (GIFs animados por ejemplo).

A los objetos les pueden ocurrir ciertas cosas. A estas cosas se les llama *eventos*. Los objetos pueden realizar ciertas *acciones* dependiendo del evento que ocurra. Hay una gran cantidad de eventos que pueden suceder y una gran cantidad de acciones diferentes que los objetos pueden realizar. Por ejemplo, hay un *evento creación* cuando el objeto es creado. (Para ser más precisos, cuando una instancia de un objeto es creada; puede haber múltiples instancias del mismo objeto). Por ejemplo, cuando un objeto pelota se crea, puedes darle alguna acción de movimiento para que empiece a moverse. Cuando dos objetos se encuentran se tiene el *evento colisión*. En tal caso puedes hacer que la pelota se detenga o que invierta su dirección. Puedes también reproducir un efecto de sonido. Para este propósito el *Game Maker* te permite definir *sonidos*. Cuando el jugador presiona una tecla en el teclado hay un *evento teclado*, y el objeto puede realizar la acción apropiada, como moverse en la dirección indicada. Espero que entiendas la idea. Para cada objeto que diseñes puedes indicarle acciones para varios eventos, de esta forma se define el comportamiento del objeto.

Una vez que has definido tus objetos es tiempo de definir las *habitaciones* en donde habitarán. Las habitaciones pueden ser empleadas como niveles en el juego o para mostrar diferentes lugares. Hay acciones para moverse de una habitación a otra. Las habitaciones deben tener primero un *fondo.* Este puede ser un simple color de relleno o una imagen. Tales imágenes pueden ser creadas en el *Game Maker* o puedes cargarlas de algún archivo. (El fondo puede hacer muchas cosas pero por el momento, solo considéralo como algo que hace que los niveles se vean bien). Ahora puedes colocar tus objetos en la habitación. Puedes colocar múltiples instancias del mismo objeto en una

habitación. Así, por ejemplo, necesitas definir sólo un objeto de pared y usarlo en muchos lugares. También puedes tener múltiples instancias de los mismos enemigos, mientras requieran del mismo comportamiento.

Ahora estás listo para ejecutar el juego. Se mostrará la primer habitación y los objetos en ella tomarán vida por las acciones en sus eventos de creación. Empezarán a interactuar unos con otros debido a las acciones en sus eventos de colisión y pueden reaccionar a lo que haga el jugador usando las acciones en los eventos del teclado o del ratón.

En resumen, los siguientes elementos (comúnmente llamados recursos) tienen un papel crucial en el juego:

- *Objetos (objects)*: que son las entidades verdaderas en el juego.
- *Habitaciones (rooms)*: los lugares (niveles) en donde habitan los objetos.
- *Sprites*: imágenes (animadas) que se emplean para representar a los objetos.
- *Sonidos (sounds)*: para emplearse en el juego, ya sea como música de fondo o como efectos.
- *Fondos (backgrounds)*: las imágenes usadas como fondo para los cuartos.

De hecho, hay otro tipo de recursos: paths, scripts, data files y time lines. Estos recursos son importantes para juegos más complicados. Solo los verás cuando ejecutas el *Game Maker* en modo avanzado. Se hablará de ellos más tarde en los capítulos avanzados de este documento.

Capítulo 5 Veamos un ejemplo

Sería bueno echar un vistazo de cómo crear un ejemplo muy simple. El primer paso es describir el juego que queremos hacer. (Siempre debieras empezar por esto; te ahorrará mucho trabajo después). El juego será muy simple: hay una pelota rebotando entre varias paredes. El jugador debe intentar hacer clic en la pelota con el ratón. Cada vez que lo logre obtiene un punto.

Como se puede ver, se requieren dos objetos diferentes: la pelota y la pared. También necesitaremos dos sprites diferentes: uno para el objeto pared y otro para el objeto pelota. Finalmente, queremos escuchar algún sonido cuando se logre hacer clic en la pelota con el ratón. Usaremos un sólo nivel en el que se lleve a cabo el juego. (Si no quieres hacer el juego tú mismo, puedes cargarlo de la carpeta `Examples` con el nombre `touch the ball.gmd`.)

Hagamos primero los sprites. Del menú **Add** selecciona **Add Sprite** (puedes también usar el botón apropiado en la barra de herramientas). Aparecerá una ventana. En el campo **Name** escribe "wall". Selecciona el botón **Load Sprite** y elige una imagen apropiada (puedes encontrar una en la carpeta maze). Eso es todo, puedes cerrar la ventana. De esta misma forma, crea un sprite para la pelota.

Ahora definiremos el sonido. Del menú **Add** selecciona **Add Sound.** Aparece una ventana diferente. Dale un nombre al sonido y selecciona **Load Sound**. Selecciona algo apropiado y checa si es un buen sonido presionando el botón reproducir. Si te gusta, cierra la ventana.

El siguiente paso es crear los dos objetos. Hagamos primero la pared. Nuevamente del menú **Add** selecciona **Add Object.** Se abrirá una ventana que se ve bastante más compleja que las que hemos visto hasta ahora. A la izquierda tenemos información global acerca del objeto. Dale al objeto un nombre apropiado y del menú desplegable selecciona el sprite wall. Como una pared es sólida deberías marcar la casilla llamada **Solid.** Eso es todo por el momento. Nuevamente crea un objeto, llámalo ball, y selecciónale un sprite. No haremos sólida a la pelota. Necesitamos definir algún comportamiento para la pelota. Debajo hay un botón marcado **Add Event**. Presiónalo y verás todos los eventos posibles. Selecciona el evento de creación (*create*). Ahora se ha agregado a la lista de eventos. En el extremo derecho puedes ver todas las acciones posibles, en varios grupos. Del grupo **move** selecciona la acción con las 8 flechas rojas y arrástrala a la lista de acciones en el medio. Esta acción hará al objeto moverse en una dirección particular. Una vez que la sueltes en la lista de acciones se mostrará un diálogo en el cual puedes indicar la dirección de movimiento. Selecciona las 8 flechas para seleccionar una dirección al azar. Cierra el diálogo. Ahora la pelota empezará a moverse en el momento en que sea creada. En segundo lugar debemos definir lo que sucederá en caso de que se colisione con la pared. De nuevo presiona **Add Event**. Haz clic en el botón de los eventos de colisión (c*ollision*) y del menú que aparece selecciona el objeto wall. Para este evento necesitamos la acción rebotar. (Puedes ver lo que cada acción hace colocando el ratón sobre el icono que la representa). Finalmente necesitamos definir lo que se hará cuando el

usuario presione el botón izquierdo del ratón sobre la pelota. Agrega el evento correspondiente y selecciona el botón izquierdo del ratón (*Left button*) del menú que se despliega. Para este evento necesitamos unas cuantas acciones: una para reproducir un sonido (la puedes encontrar en el grupo de acciones **main1**), una para cambiar el marcador (se encuentra en el grupo **score**) y dos más para mover la pelota a una posición aleatoria y moverla en una nueva dirección (de la misma forma que en el evento de creación). Para la acción de sonido, selecciona el sonido adecuado. Para la acción del marcador (*score*), introduce un valor de 1 y marca la opción **Relative**. Esto significa que se agregará 1 al marcador actual. (Si cometes algún error puedes hacer doble clic sobre la acción para modificar su configuración).

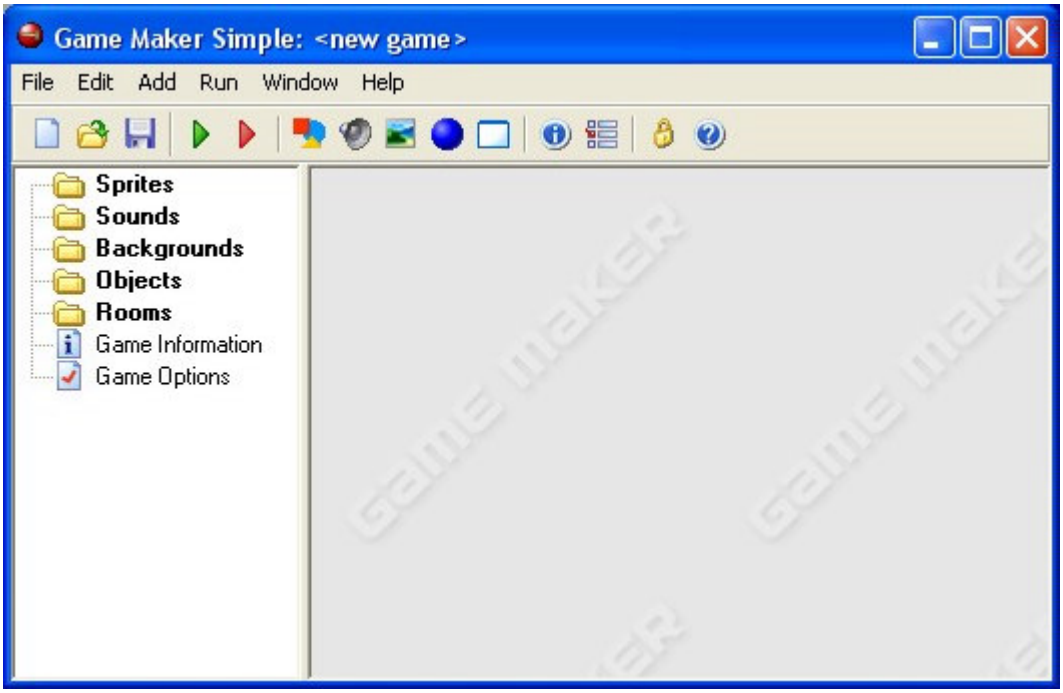
Nuestros objetos ahora están listos. Lo que queda es definir la habitación. Agrega un nuevo nivel al juego, de nuevo desde el menú **Add**. A la derecha verás la habitación vacía. A la izquierda verás algunas propiedades que puedes modificar, como el ancho y alto de la habitación. En la parte inferior izquierda puedes seleccionar un objeto del menú desplegable. Puedes colocar instancias de ese objeto haciendo clic en el cuarto. Puedes eliminar las instancias usando el botón derecho del ratón. Crea un espacio cerrado alrededor del nivel usando el objeto wall. Finalmente coloca 1 o 2 objetos ball en el nivel. Nuestro juego está listo.

Es hora de probar nuestro juego. Presiona el botón **Run** y ve lo que pasa. Si no cometiste algún error la pelota empieza a moverse alrededor. Intenta hacer clic en ella con el ratón y ve qué sucede. Puedes detener el juego presionando la tecla <Esc>. Ahora puedes hacer más cambios.

Felicitaciones. Haz creado tu primer juego. Pero creo que es hora de aprender un poco más sobre el *Game Maker*.

Capítulo 6 La interfaz del usuario

Al iniciar el *Game Maker* se muestra la siguiente ventana:



(De hecho, esto es lo que ves cuando ejecutas el *Game Maker* en modo simple. En el modo avanzado se muestran algunas opciones más. Ve el Capítulo 14 para más detalles). A la izquierda puedes ver los diferentes *recursos*, mencionados antes: Sprites, Sounds (Sonidos), Backgrounds (Fondos), Scripts, Objects (Objetos), Rooms (Habitaciones) y dos más: Game Information (Información del juego) y Game Options (Opciones del juego) En la parte superior está el ya conocido menú y la barra de herramientas. En este capítulo describiré brevemente las diferentes opciones del menú, los botones, etc. En capítulos posteriores trataremos varios de ellos con más detalle. Recuerda que muchas cosas se pueden lograr de diferentes maneras: seleccionando un comando del menú, haciendo clic en un botón, o haciendo clic derecho sobre un recurso.

6.1 Menú File

En el menú file puedes encontrar los comandos usuales para cargar y guardar archivos, además de algunos especiales:

- **New**. Selecciona este comando para empezar a crear un juego nuevo. Si el juego actual sufrió modificaciones se te pregunta si quieres guardarlo. También hay un botón en la barra de herramientas para hacer esto.
- **Open**. Abre un archivo de juego. Los archivos del *Game Maker* tienen la extensión .gmd. Hay un botón en la barra de herramientas para este comando. Puedes también abrir un juego arrastrando el archivo a la ventana del *Game Maker*.
- **Recent Files**. Usa este submenú para reabrir los archivos abiertos recientemente.

- **Save**. Guarda el archivo de diseño del juego con el nombre actual. Si no se había especificado un nombre, se te pide dar uno. Puedes usar este comando solo cuando el archivo ha cambiado. Como con los anteriores, hay un botón en la barra de herramientas para realizar esta acción.
- **Save As**. Guarda el diseño del juego con otro nombre. Se te pide un nuevo nombre.
- **Create Executable**. Una vez que tu juego esté listo probablemente querrás distribuirlo. Mediante este comando puedes crear una versión ejecutable de tu juego. Es simplemente un ejecutable que puedes darle a otros para que jueguen tu juego. Encontrarás más información sobre la distribución de juegos en el Capítulo 14.
- **Advanced Mode**. Al hacer clic en este comando el *Game Maker* cambiará entre los modos simple y avanzado. En el modo avanzado se tienen disponibles comandos y recursos adicionales.
- **Exit**. Probablemente obvio. Presiónalo para cerrar el *Game Maker*. Si hiciste cambios al juego actual se te preguntará si quieres guardarlo.

6.2 Menú Edit

El menú edit contiene varios comandos que se relacionan con el recurso seleccionado (objeto, sonido, etc.) o grupo de recursos. Dependiendo del tipo de recurso algunos de los comandos pueden no estar disponibles.

- **Insert resource**. Inserta una nueva instancia del tipo de recurso actualmente seleccionado, se inserta antes del mismo. (Si seleccionaste un grupo de recursos el recurso es agregado al grupo). Se abrirá un cuadro de diálogo en el que puedes cambiar las propiedades del recurso. Esto se tratará con más detalle en los capítulos siguientes.
- **Duplicate**. Crea y agrega una copia del recurso actual. Se abre una ventana en la que puedes modificar el recurso.
- **Delete**. Borra el recurso seleccionado (o grupo de recursos). Ten cuidado. No puede deshacerse. Por lo que se te hará una advertencia.
- **Rename**. Dale un nuevo nombre al recurso. Esto se puede hacer también en la ventana de propiedades del recurso. También puedes seleccionar el recurso y entonces hacer clic en el nombre.
- **Properties**. Usa este comando para mostrar la ventana de edición de propiedades. Nota que todas las ventanas de propiedades se muestran en la ventana principal. Puedes editar muchas de ellas al mismo tiempo. Puedes también editar las propiedades haciendo doble clic en el recurso.

Nota que estos comandos puedes también aplicarse de otra forma. Haz clic derecho en un recurso o en un grupo de recursos, y aparecerá el menú apropiado.

6.3 Menú Add

En este menú puedes agregar un nuevo recurso de cada uno de los diferentes tipos que hay. Nota que para cada uno de ellos hay también un botón en la barra de herramientas y un acceso directo con el teclado.

6.4 Menú Window

En este menú puedes encontrar los comandos comunes para manejar las diferentes propiedades de las ventanas de la ventana principal:

- **Cascade**. Coloca todas las ventanas en cascada de forma que cada una sea parcialmente visible.
- **Arrange Icons**. Ordena todas las ventanas. (Útil en particular cuando cambias de tamaño la ventana principal).
- **Close All**. Cierra todas las ventanas de propiedades, preguntando si se desea guardar o no los cambios realizados.

6.5 Menú Help

Aquí encontrarás algunos comandos de ayuda:

- **Contents**. Te permite acceder a la versión en formato de ayuda de este documento.
- **How to use help**. En caso de que no sepas, algo de ayuda sobre cómo usar la ayuda.
- **Registration**. Aún cuando el *Game Maker* puede ser usado libre de cargo, se te alienta a registrar el programa. De esta forma se eliminará la ventana publicitaria que en ocasiones aparece en la ventana de edición y ayudarás al futuro desarrollo del programa. Aquí puedes encontrar información sobre cómo registrar el programa. Si ya lo registraste puedes usar esta opción para introducir la clave de registro que recibieste.
- **Web site**. Te conecta al sitio del *Game Maker* en donde puedes encontrar información sobre la versión más reciente del *Game Maker* y colecciones de juegos y recursos para el *Game Maker*. Te recomiendo que visites el sitio al menos una vez al mes para ver nueva información.
- **Forum**. Te conecta al foro del *Game Maker* donde mucha gente te ayudará con tus dudas.
- **About Game Maker**. Breve información sobre esta versión del *Game Maker*.

6.6 El explorador de recursos

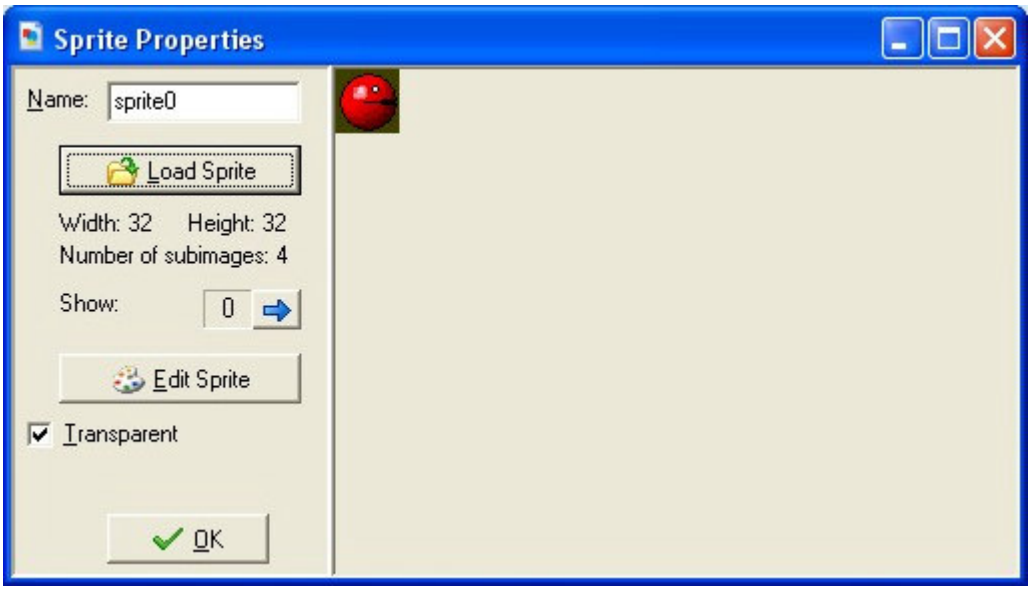
En la parte izquierda de la forma principal encontrarás el explorador de recursos. Aquí tienes una lista ramificada de todos los recursos de tu juego. Trabaja de la misma forma que el explorador de Windows con el cual muy probablemente ya estás familiarizado. Si un elemento tiene un signo + a su lado puedes hacer clic en el signo para ver los recursos dentro de él. Haciendo clic en el signo – los oculta nuevamente. Puedes cambiar el nombre de un recurso (excepto los de más alto nivel) seleccionándolo (con un solo clic) y luego haciendo clic en el nombre. Haz doble clic en un recurso para editar sus propiedades. Usa el botón derecho del ratón para acceder a los comandos del menú **Edit**. Puedes cambiar el orden de los recursos haciendo clic en ellos y manteniendo presionado el botón del ratón. Ahora arrastra el recurso (o grupo) al lugar apropiado. (Por supuesto que el lugar debe ser el apropiado. Por ejemplo no puedes arrastrar un sonido a la lista de sprites).

Capítulo 7 Definiendo los sprites

Los sprites son las representaciones visuales de todos los objetos en el juego. Un sprite es o una imagen simple, dibujada con cualquier programa que te guste, o un juego de varias imágenes que, pueden ser reproducidas una tras otra, creando un efecto de movimiento (animación). Por ejemplo, las siguientes cuatro imágenes forman un sprite para un Pacman moviéndose hacia la derecha.

Cuando haces un juego empiezas por reunir los distintos sprites para los objetos en tu juego. Con el *Game Maker* viene incluida una colección de sprites estáticos y animados. Puedes encontrar otros sprites en Internet, normalmente como archivos gif animados.

Para agregar un sprite, selecciona el comando **Add Sprite** del menú **Add**, o usa el botón correspondiente de la barra de herramientas. Aparecerá la siguiente ventana.



En la parte superior puedes indicar el nombre del sprite. Todos los sprites (y todos los demás recursos) tienen un nombre. Será mejor que le des a cada sprite un nombre descriptivo y fácil de recordar. Asegúrate de que todos los recursos tengan diferentes nombres. Aunque no es estrictamente requerido, se recomienda firmemente usar sólo letras, dígitos y el guión bajo (_) en el nombre de un sprite (y de cualquier otro recurso) y empezar el nombre con una letra. En particular no hagas uso de espacios en los nombres. Esto pasará a ser parte importante una vez que empieces a usar código.

Para cargar un sprite, haz clic en el botón **Load Sprite**. Se muestra un diálogo de archivo en el que debes indicar el sprite. El *Game Maker* puede abrir varios tipos de archivos de

imagen. Cuando cargas un gif animado, las diferentes subimágenes forman las imágenes del sprite. Una vez que el sprite es cargado, se muestra la primera subimagen a la derecha. Cuando hay múltiples subimágenes, puedes verlas usando los botones con las flechas.

La opción **Transparent** indica si el fondo deberá ser considerado como transparente. La mayoría de los sprites son transparentes. El fondo es determinado por el color del pixel que se encuentra en la esquina inferior izquierda de la imagen. Así que asegúrate de que ningún pixel de la imagen que se mostrará tenga este color. (Los archivos gif a menudo definen su color de transparencia, este color no es usado en el *Game Maker*).

Con el botón **Edit Sprite** puedes editar el sprite, o incluso crear un sprite completamente nuevo. Para más información sobre la creación y modificación de sprites, ve el Capítulo 14**.**

Capítulo 8 Sonidos y música

La mayoría de los juegos tienen ciertos efectos de sonido y música de fondo. El *Game Maker* incluye algunos efectos de sonido de utilidad. Puedes encontrar muchos más en la web.

Para agregar un recurso de sonido a tu juego, usa el comando **Add Sound** en el menú **Add** o usa el botón correspondiente en la barra de herramientas. Aparecerá la siguiente ventana.

Para cargar un sonido, presiona el botón **Load Sound**. Aparece un diálogo de selección de archivo en el que debes especificar el archivo de sonido. Hay dos tipos de archivos de sonido, archivos wave (*.wav) y archivos midi (*.mid). (Para información sobre archivos mp3 ve el Capítulo 17). Los archivos wave son usados para efectos breves de sonido. Usan mucha memoria pero se reproducen instantáneamente. Úsalos para todos los efectos de sonido en el juego. Los archivos midi tratan la música en una forma diferente. Como resultado usan mucha menos memoria, pero están limitados a música instrumental de fondo. Además, sólo se puede reproducir un archivo midi en algún momento.

Una vez que cargues un archivo se muestra su tipo y duración. Puedes escuchar el sonido usando el botón reproducir. También hay un botón **Save Sound** para guardar en un archivo el sonido actual. Este botón no es realmente requerido pero pudieras necesitarlo si perdieras el sonido original.

Capítulo 9 Fondos

El tercer tipo de recursos básicos son los fondos. Los fondos son normalmente imágenes grandes usadas como fondo para los niveles en los que se realiza el juego. A menudo las imágenes de fondo son hechas con tiles (mosaicos) de manera que pueden formar una imagen en un área sin errores visuales. De esta manera puedes llenar el fondo con un patrón. Varios tiles de fondo de este tipo vienen incluidos con el *Game Maker*. Puedes encontrar muchos más en la web.

Para agregar un fondo a tu juego, usa el comando **Add Background** en el menú **Add** o usa el botón correspondiente de la barra de herramientas. Aparecerá la siguiente ventana.

Presiona el botón **Load Background** para cargar una imagen de fondo. El *Game Maker* soporta varios formatos de imagen. ¡Las imágenes de fondo no pueden ser animadas! La opción **Transparent** indica si el fondo será parcialmente transparente. La mayoría de los fondos no son transparentes, por lo que la opción por defecto es no. Como color transparente se usa el color del pixel en la esquina inferior izquierda.

Puedes cambiar el fondo o crear uno nuevo usando el botón **Edit Background**. Para más información, ve el Capítulo 18.

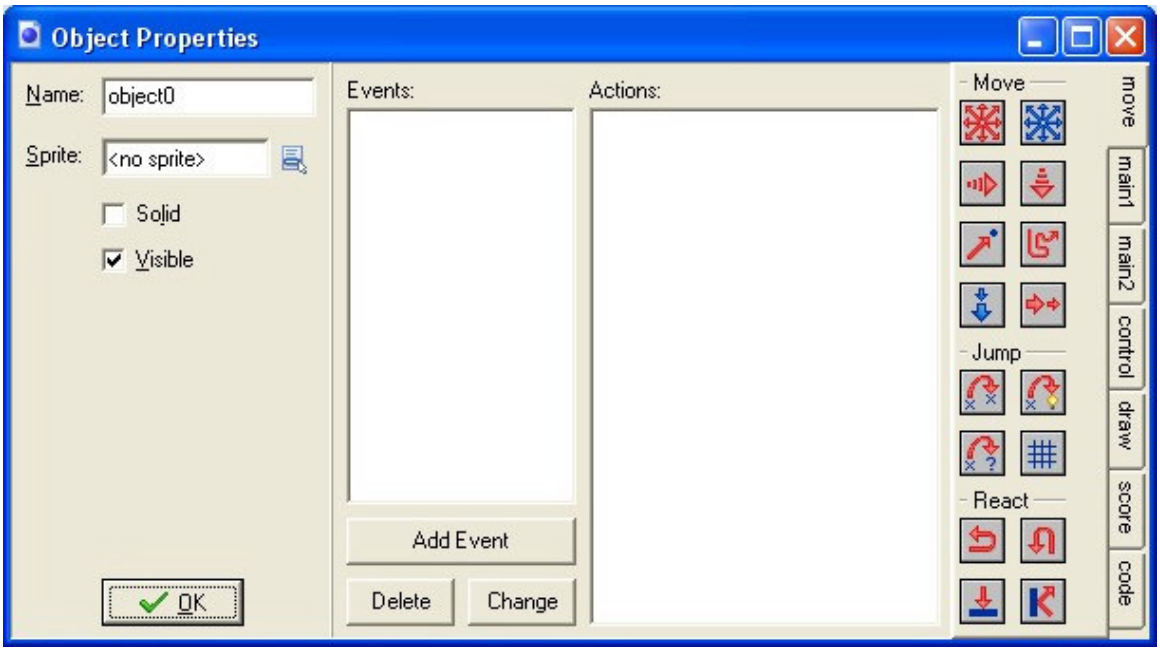
Capítulo 10 Definiendo objetos

Hemos agregado imágenes y sonidos al juego, pero ninguno de ellos realiza ninguna acción. Ahora llegamos al recurso más importante dentro del *Game Maker*: los objetos. Los objetos son las entidades que hacen las cosas en el juego. La mayoría de las veces tienen un sprite como representación gráfica para que puedas verlos. Tienen un comportamiento porque pueden reaccionar a ciertos eventos. Todas las cosas que ves en el juego (excepto el fondo) son objetos. (O para ser más precisos, son instancias de objetos). Los personajes, los enemigos, las pelotas, las paredes, etc. Puede haber también ciertos objetos que no ves pero que controlan ciertos aspectos del juego.

Por favor nota la diferencia entre los sprites y los objetos. Los sprites son solo imágenes (animadas) que no tienen ningún comportamiento. Los objetos normalmente tienen un sprite que los representa pero también tienen un comportamiento. ¡Sin objetos no hay juego!

También nota la diferencia entre los objetos y las instancias. Un objeto describe cierta entidad, por ejemplo un enemigo. Puede haber múltiples instancias de este objeto en el juego. Cuando hablamos de instancia nos referimos a una instancia en particular del objeto. Cuando hablamos de un objeto nos referimos a todas las instancias de este objeto.

Para agregar un objeto al juego, selecciona el comando **Add Object** del menú **Add**. Aparecerá la siguiente ventana:



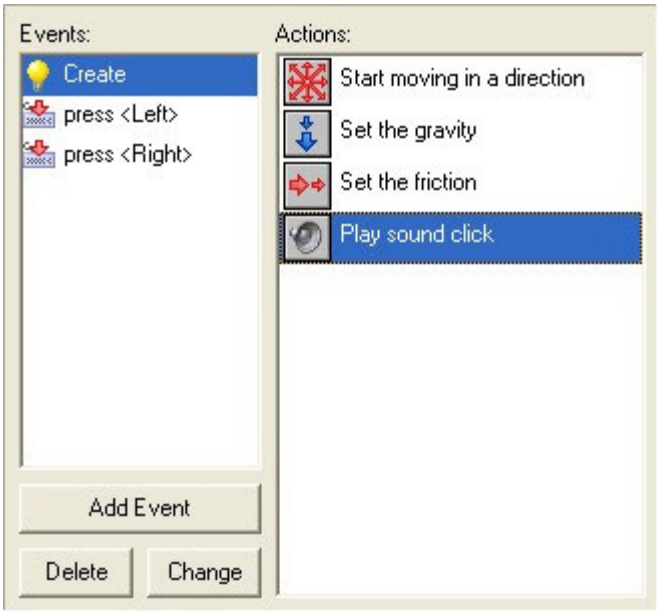
Esta es bastante compleja. A la izquierda hay cierta información general acerca del objeto. En el medio hay varios eventos posibles que pueden ocurrirle a un objeto. Ve el siguiente capítulo para más detalles. A la derecha hay varias acciones que puede realizar el objeto. Estas serán tratadas en el Capítulo 12.

Como siempre, puedes (y debes) agregar un nombre al objeto. Después puedes indicar el sprite para el objeto. Para hacerlo, haz clic con el botón izquierdo del ratón en la caja de sprite o en el botón de menú junto a esta. Aparecerá un menú con todos los sprites disponibles. Selecciona el que quieras usar para el objeto. Debajo hay dos opciones. **Solid** indica si este objeto es sólido (como una pared). Las colisiones con objetos sólidos son tratadas de forma diferente que las colisiones con objetos que no lo son. Ve el siguiente capítulo para más información. La opción **Visible** indica si las instancias de este objeto serán visibles. Obviamente, la mayoría de los objetos son visibles, pero algunas veces es útil tener objetos invisibles. Por ejemplo, puedes usarlos como guías de movimiento para un enemigo. Los objetos invisibles reaccionan a eventos y otras instancias pueden colisionar con ellos.

Capítulo 11 Eventos

El *Game Maker* emplea lo que se conoce como programación orientada a eventos. Esto es, en todo tipo de situaciones las instancias de los objetos reciben eventos (como mensajes que indican que algo ha sucedido). Entonces los objetos pueden reaccionar a estos mensajes ejecutando ciertas acciones. Para cada objeto debes indicar a qué eventos responderá y qué acciones debe realizar. Puede parecer complicado pero en realidad es bastante sencillo. Primero que nada, para la mayoría de los eventos los objetos no tienen que hacer nada. Para los eventos donde algo suceda puedes usar muy simples comandos de arrastrar y soltar para indicar las acciones.

En medio de la ventana de propiedades de objeto hay una lista de los eventos a los cuales el objeto puede reaccionar. Al principio está vacía. Puedes agregar eventos presionando el botón **Add Event**. Aparecerá un pequeño menú con todos los diferentes tipos de eventos. Aquí debes seleccionar el evento que deseas agregar. En ocasiones se mostrará un nuevo menú con opciones extra. Por ejemplo, para el evento del teclado debes seleccionar la tecla. Más abajo encontrarás una completa lista con descripciones de los eventos. Seleccionaremos un evento de la lista. Este será el evento que modificaremos. Puedes cambiar el evento seleccionado haciendo clic sobre él. A la derecha están todas las acciones representadas por pequeños iconos. Se encuentran agrupadas en varias páginas / fichas de opciones. En el siguiente capítulo describiré todas las acciones y lo que realizan. Entre los eventos y las acciones se encuentra la lista. Esta lista contiene las acciones para el evento actual. Para agregar acciones a la lista, arrástralas desde la derecha a la lista. Serán colocadas una bajo la otra, con una breve descripción. Para cada acción se te pedirán algunos parámetros. Estos se describirán también en el siguiente capítulo. Después de agregar algunas acciones, tendrías algo como esto:



Ahora puedes agregar acciones a otro evento. Haz clic con el botón izquierdo del ratón sobre el evento adecuado para seleccionarlo y arrastra la acción a la lista.

Puedes cambiar el orden de las acciones en la lista arrastrando los iconos. Si mantienes presionada la tecla <Ctrl.> mientras arrastras una acción, crearás una copia de dicha acción. Puedes inclusive arrastrar acciones entre diferentes listas de diferentes objetos. Cuando haces clic con el botón derecho sobre una acción, se muestra un menú desde el cual puedes eliminar la acción (también puedes hacer esto usando la tecla <Supr>), o copiar y pegar acciones. Cuando mantienes el cursor del ratón sobre una acción, se muestra una descripción más detallada sobre la misma. En el siguiente capítulo hay más información sobre las acciones.

Para eliminar el evento seleccionado y todas sus acciones, presiona el botón **Delete**. (Los eventos sin acciones son eliminados automáticamente cuando cierras la ventana por lo que no hay necesidad de que lo hagas tu mismo). Si deseas asignar las acciones a un evento diferente (porque por ejemplo, has decidido emplear una tecla diferente para las acciones) presiona el botón Change y selecciona el nuevo evento. (¡El nuevo evento no debe haber sido empleado antes!).

Como se mencionó arriba, para agregar un evento, presiona el botón **Add Event**. Se muestra la siguiente ventana:

Aquí seleccionas el evento que deseas agregar. Algunas veces aparece un menú con opciones extra. A continuación una descripción de los eventos. (Recuerda que normalmente sólo haces uso de algunos de ellos).

Evento Create

Este evento ocurre cuando se crea una instancia del objeto. Normalmente se usa para establecer el movimiento de la instancia y/o establecer ciertas variables de la misma.

Evento Destroy

Este evento ocurre cuando la instancia es destruida. Para ser precisos, ocurre justo antes de que sea destruida, ¡por lo que la instancia aún existe cuando el evento es ejecutado! La mayoría de las veces este evento no se usa pero puedes por ejemplo emplearlo para cambiar el marcador o para crear algún otro objeto.

Eventos Alarm

Cada instancia tiene 8 relojes de alarma. Puedes configurar estos relojes usando ciertas acciones (ver el siguiente capítulo). El reloj de alarma hace una cuenta regresiva hasta

que llega a 0, que es cuando se genera el evento de alarma. Para indicar las acciones para un reloj de alarma, necesitas primero seleccionarlo en el menú. Los relojes de alarma son muy útiles. Puedes usarlos para hacer que ciertas cosas sucedan de tiempo en tiempo. Por ejemplo un enemigo puede cambiar su dirección de movimiento cada 20 pasos. (En cuyo caso una de las acciones en el evento debe configurar nuevamente la alarma).

 **Eventos Step**

El evento step sucede a cada paso/frame del juego. Aquí puedes colocar acciones que requieren ejecutarse continuamente. Por ejemplo, si el objeto debe seguir a otro, puedes adaptar aquí la dirección de movimiento hacia el objeto que se está siguiendo. Sin embargo ten cuidado con este evento. No coloques muchas acciones muy complicadas en el evento step. Esto podría alentar el juego. Para ser más preciso, hay tres diferentes eventos step. Normalmente solo necesitas el evento por defecto. Pero usando el menú puedes también seleccionar el inicio y el final del evento step. El inicio del evento step es ejecutado al inicio de cada paso, antes de que ocurra cualquier otro evento. El evento step normal es ejecutado justo antes de que se coloquen las instancias en sus nuevas posiciones. El final del evento step se ejecuta al final de cada paso, justo antes de que se dibuje la escena. Este evento se usa normalmente para, por ejemplo, cambiar el sprite dependiendo de la dirección actual.

 **Eventos Collision**

En el momento en que dos instancias colisionan (esto es, que sus sprites se sobreponen) aparece un evento collision (colisión). Bien, para ser precisos, ocurren dos eventos de colisión, uno por cada instancia. La instancia puede reaccionar a este evento de colisión. Para este fin, selecciona en el menú el objeto con el cual quieres definir el evento de colisión. Después coloca las acciones.

Hay una diferencia entre lo que sucede cuando la instancia colisiona con un objeto sólido y cuando lo hace con uno no sólido. Primero que nada, cuando no hay acciones en el evento de colisión, no sucede nada. La instancia actual simplemente continúa moviéndose; aún cuando el otro objeto sea sólido. Cuando el evento de colisión contiene acciones sucede lo siguiente:

Cuando el otro objeto es sólido, la instancia se coloca de vuelta al lugar previo (antes de que ocurriera la colisión). Entonces se ejecuta el evento. Finalmente, la instancia es llevada a su nueva posición. Así que si por ejemplo, el evento invierte la dirección de movimiento, la instancia rebota contra la pared sin detenerse. Si hay una nueva colisión, la instancia se mantiene en su lugar previo. De manera que efectivamente deje de moverse.

Cuando el otro objeto no es sólido, la instancia no se regresa. El evento simplemente se ejecuta con la instancia en su posición actual. Tampoco se checa una nueva colisión. Si lo piensas, esto es lo que debería ocurrir lógicamente. Porque cuando el objeto no es sólido, podemos simplemente movernos sobre él. El evento nos notifica lo que está sucediendo.

Existen muchos usos para el evento collision. Las instancias pueden usarlo para rebotar contra las paredes. Puedes usarlo para destruir algún objeto cuando es tocado por una bala, etc.

Eventos Keyboard

Cuando el usuario presiona una tecla, ocurre un evento keyboard para todas las instancias de todos los objetos. Hay un evento diferente para cada tecla. En el menú puedes seleccionar la tecla para la que quieres definir el evento de teclado y después arrastrar las acciones. Es claro que solo unos cuantos objetos necesitan eventos para unas cuantas teclas. Se tiene un evento en cada paso cuando el jugador presiona la tecla. Hay dos eventos de teclado especiales. Uno es el <No key>. Este evento ocurre en cada paso cuando no hay ninguna tecla presionada. El segundo se llama <Any key> y ocurre cuando se presiona cualquier tecla. A propósito, cuando el usuario presiona varias teclas, ocurren los eventos para todas ellas. Las teclas en el teclado numérico solo producen sus eventos correspondientes cuando <BloqNum> está activada.

Eventos Mouse

Un evento mouse ocurre para una instancia cuando el cursor del ratón se encuentra sobre el sprite que representa a la instancia. Dependiendo de qué botón se presione, se obtienen los eventos no button (ningún botón), left button (botón izquierdo), right button (botón derecho), o middle button (botón central). Estos eventos son generados en cada paso mientras el jugador mantenga el botón del ratón presionado. Los eventos press se generan una sola vez cuando el botón es presionado. Recuerda que estos eventos sólo ocurren cuando el ratón está sobre la instancia. Si el jugador presiona un botón del ratón en algún lugar donde no haya ninguna instancia, no se genera el evento. Pero en ocasiones es importante reaccionar a cualquier clic del ratón. Esto puede conseguirse creando un sprite del tamaño del cuarto. Ahora crea un objeto con el sprite que cubre todo el cuarto. Puedes hacer a este objeto invisible. Colócalo en el cuarto, y generará eventos en cualquier momento que el jugador presione algún botón del ratón.

Otros eventos

Hay otros eventos que pueden ser útiles en ciertos juegos. Los encuentras en este menú. Los eventos que puedes encontrar son:

- **Outside**: Este evento ocurre cuando la instancia se encuentra completamente fuera del cuarto. Este es normalmente un buen momento para destruirla.
- **Boundary**: Este evento ocurre cuando la instancia llega al borde del cuarto.
- **Game start**: Este evento se genera para todas las instancias en el primer cuarto donde inicia el juego. Ocurre antes que el evento room start (ver abajo) y aún antes de los eventos create de las instancias en el cuarto. Este evento se define normalmente en un solo objeto "controlador" y se usa para reproducir alguna música de fondo y para dar valores iniciales a algunas variables, o para cargar información
- **Game end**: El evento ocurre para todas las instancias cuando termina el juego. De nuevo, es común que solo un objeto lo defina. Se emplea por ejemplo para guardar cierta información en un archivo.

- **Room start**: Este evento se genera para todas las instancias (en el cuarto) cuando se inicia un cuarto. Ocurre antes que los eventos de creación.
- **Room end**: Este evento se genera para todas las instancias existentes cuando finaliza el cuarto.
- **No more lives**: El *Game Maker* tiene un sistema de vidas interno. Hay una acción para especificar y cambiar el número de vidas. En el momento que el número de vidas sea menor o igual a 0, se genera este evento. Normalmente empleado para terminar o reiniciar el juego.
- **No more health**: El *Game Maker* tiene un sistema interno de energía. Hay una acción para especificar y cambiar el nivel de energía. En el momento que la energía sea menor o igual a 0, ocurre este evento. Normalmente se emplea para disminuir el número de vidas o para reiniciar el juego.
- **End of animation**: Como se indicó antes, una animación consiste en varias imágenes que se muestran una después de otra. Después de que se muestra la última se inicia nuevamente con la primera. El evento ocurre en ese preciso momento. Esto se puede usar para por ejemplo cambiar la animación, o destruir la instancia.
- **End of path**: Este evento ocurre cuando la instancia que sigue una trayectoria predeterminada (path) llega al final de la misma. Ve al **Capítulo 18** para encontrar más información sobre las trayectorias.
- **User defined**: Hay ocho de estos eventos. Normalmente nunca ocurren a menos que tú mismo los llames desde alguna pieza de código.

 **Evento Drawing**

Las instancias, cuando son visibles, dibujan su sprite en la pantalla en cada paso. Cuando especificas acciones en el evento de dibujo, no se dibuja el sprite, sino que se ejecutan estas acciones en su lugar. Esto se puede usar para dibujar algo más que un sprite, o para primero hacer algunos cambios a los parámetros del sprite. Hay varias acciones de dibujo especialmente diseñadas para usarse en el evento drawing. Recuerda que el evento drawing se ejecuta solamente cuando el objeto es visible. También recuerda que, independientemente de lo que dibujes aquí, los eventos de colisión se basan en el sprite que está asociado a la instancia

 **Eventos Key press**

Este evento es similar al evento keyboard con la diferencia de que éste ocurre solo una vez cuando se presiona la tecla, y no continuamente. Este es útil cuando deseas que una acción ocurra solo una vez.

 **Eventos Key release**

Este evento es similar al evento keyboard, pero ocurre solo una vez cuando la tecla es liberada, en lugar de ocurrir continuamente.

En algunas situaciones es importante conocer el orden en el cual el *Game Maker* procesa los eventos. Como sigue.

- Eventos Begin Step

- Eventos Alarm
- Eventos Keyboard, Key press, y Key release
- Eventos Mouse
- Eventos Step normales
- (ahora todas las instancias son colocadas en sus nuevas posiciones)
- Eventos Collision
- Eventos End Step
- Eventos Drawing

Los eventos de creación (create), destruir (destroy) y otros son generados en el momento pertinente.

Capítulo 12 Acciones

Las acciones indican lo que sucede en un juego creado con el *Game Maker*. Las acciones se colocan en los eventos de los objetos. Cuando el evento ocurre estas acciones se llevan a cabo, resultando en cierto comportamiento para las instancias del objeto. Hay una gran cantidad de acciones disponibles y es importante que entiendas lo que hacen. En este capítulo describiré las acciones por defecto. Acciones adicionales pueden encontrarse en librerías de acciones. Estas librerías extienden las posibilidades del *Game Maker* checa el sitio web por posibles librerías de acciones adicionales.

Todas las acciones se encuentran en las páginas / fichas a la derecha de la ventana de propiedades de objeto. Hay siete grupos de acciones. Puedes ver el grupo haciendo clic en la ficha correcta. Cuando mantienes el ratón sobre una de las acciones, se muestra una breve descripción para recordar su función.

Permíteme repetir brevemente: Para colocar una acción en un evento, solo arrástrala de las páginas / fichas a la lista de acción. Puedes cambiar el orden de la lista, arrastrando y soltando los iconos. Si presionas la tecla <Ctrl> mientras arrastras puedes hacer una copia de la acción. (Puedes arrastrar y copiar acciones entre las listas en diferentes ventanas de propiedades de objetos). Usa el botón derecho del ratón para borrar acciones (o usa la tecla <Supr>) o para copiar y pegar acciones.

Cuando colocas una acción, la mayoría de las veces aparece una ventana de diálogo, en la cual puedes especificar ciertos parámetros para la acción. Dos tipos de parámetros aparecen en muchas acciones. En la parte superior puedes indicar a qué instancia aplica la acción. El valor por defecto es self, que es la instancia para la cual se está realizando la acción. La mayoría de las veces esto es lo adecuado. En el caso de un evento collision, puedes especificar también si se aplica la acción a la otra instancia involucrada en la colisión. De esta forma puedes por ejemplo destruir la otra instancia. Finalmente, puedes elegir si aplicas la acción a todas las instancias de un objeto. De esta manera puedes por ejemplo cambiar todas las pelotas rojas por azules. El segundo tipo de parámetro es la casilla marcada **Relative**. Al marcar esta casilla, los valores que introduzcas serán relativos a los valores actuales. Por ejemplo, de esta forma puedes agregar algo al marcador actual, en lugar de cambiar el marcador actual por su nuevo valor. Los otros parámetros se describen abajo. Puedes revisar después los parámetros haciendo doble clic en la acción.

12.1 Acciones de movimiento (página / ficha move)

El primer grupo de acciones consiste de aquellas relacionadas al movimiento de objetos. Tenemos las siguientes:

 **Start moving in a direction**

Usa esta acción para que la instancia empiece a moverse en una dirección en particular. Puedes indicar la dirección usando las flechas. Usa el botón del medio para detener el movimiento. También debes especificar la velocidad de movimiento. Esta velocidad está

dada en pixeles por paso. El valor por defecto es de 8. De preferencia no uses valores de velocidad negativos. Puedes especificar múltiples direcciones. En este caso se elige una al azar. De esta forma puedes por ejemplo hacer que un enemigo se mueva hacia la izquierda o hacia la derecha.

 **Set direction and speed of motion**
Esta es la segunda forma en la que puedes especificar un movimiento (con las flechas azules). Aquí puedes indicar una dirección con más precisión. Con un ángulo de entre 0 y 360 grados. 0 significa hacia la derecha. La dirección es contra las manecillas del reloj. Así por ejemplo 90 indica la dirección hacia arriba. Si quieres una dirección arbitraria, puedes poner `random(360)`. Como verás más abajo la función `random` da un número al azar menor que el valor indicado. Como podrás haber notado hay una casilla llamada **Relative**. Si la seleccionas, el nuevo movimiento es sumado al anterior. Por ejemplo, si la instancia se está moviendo hacia arriba y le agregas un pequeño movimiento hacia la izquierda, el nuevo movimiento será hacia arriba y a la izquierda.

 **Set the horizontal speed**
La velocidad de una instancia consiste de una parte horizontal y otra vertical. Con esta acción puedes cambiar la velocidad horizontal. Una velocidad horizontal positiva representa movimiento hacia la derecha. Una negativa movimiento hacia la izquierda. La velocidad vertical permanecerá constante. Usa la opción **Relative** para incrementar la velocidad horizontal (o disminuirla introduciendo un valor negativo).

 **Set the vertical speed**
De manera similar, con esta acción puedes cambiar la velocidad vertical de la instancia.

 **Move towards a point**
Esta acción da otra manera de especificar movimiento. Puedes indicar una posición y velocidad, y la instancia comienza a moverse con esa velocidad hacia ese punto. (¡El objeto no se detendrá en ese punto!). Por ejemplo, si quieres que una bala se mueva hacia la posición de una nave (llamada spaceship) puedes usar como posición `spaceship.x, spaceship.y`. (Aprenderás más sobre el uso de variables como estas más adelante). Si seleccionas la casilla **Relative**, especificas la posición relativa a la posición actual de la instancia. (¡La velocidad no es tomada como relativa!)

 **Set a path for the instance**
(Only available in advanced mode.) With this action you can indicate that the instance should follow a particular path. You indicate the path, the speed and the position in the path where to start (0=beginning, 1=end). See 0 for more information on paths.

 **Set the gravity**
Con esta acción puedes especificar una gravedad para un objeto en particular. Especificas una dirección (un ángulo entre 0 y 360 grados) y una velocidad, y a cada paso esta

cantidad de velocidad en la dirección especificada se aumenta al movimiento actual de la instancia del objeto. Normalmente necesitas un muy pequeño incremento de velocidad (como 0.01). Normalmente se emplea una dirección hacia abajo (270 grados). Si seleccionas la casilla **Relative** incrementas la velocidad de gravedad y la dirección. Nota que, contrario a la vida real, diferentes objetos pueden tener diferentes direcciones de gravedad.

 **Set the friction**
La fricción disminuye la velocidad de las instancias cuando se mueven. Puedes especificar la cantidad de fricción. En cada paso esta cantidad se resta de la velocidad hasta llegar a 0. Normalmente usas un número muy pequeño (como 0.01).

 **Jump to a given position**
Usando esta acción puedes colocar una instancia en una posición en particular. Simplemente especificas las coordenadas 'x' y 'y', y la instancia es colocada con su punto de referencia en esa posición. Si la casilla **Relative** está marcada, la posición es relativa a la posición actual de la instancia.

 **Jump to the start position**
Esta acción coloca la instancia de vuelta a la posición donde fue creada.

 **Jump to a random position**
Esta acción mueve la instancia a una posición aleatoria cuarto. Solo se usan posiciones en donde la instancia no intercepte ninguna instancia sólida. Puedes especificar la rapidez (snap) usada. Si especificas valores positivos, las coordenadas serán elegidas con múltiplos enteros de los valores indicados. Esto puede ser usado por ejemplo para mantener una instancia alineada con las celdas de tu juego (si la hay). Puedes especificar la rapidez horizontal y la vertical por separado.

 **Snap to grid**
Con esta acción puedes alinear la posición de una instancia a una cuadrícula. Puedes indicar los valores horizontal y vertical (esto es, el tamaño de las celdas de la cuadrícula). Esto puede ser muy útil para asegurarte que las instancias se mantengan en una cuadrícula.

 **Reverse horizontal direction**
Con esta acción inviertes el movimiento horizontal de la instancia. Esto lo puedes usar por ejemplo cuando el objeto colisiona con una pared vertical.

 **Reverse vertical direction**
Con esta acción inviertes el movimiento vertical de la instancia. Esto lo puedes usar por ejemplo cuando el objeto colisiona con una pared horizontal.

 **Move to contact position**
With this action you can move the instance in a given direction until a contact position with an object is reached. If there already is a collision at the current position the instance is not moved. Otherwise, the instance is placed just before a collision occurs. You can specify the direction but also a maximal distance to move. For example, when the instance is falling you can move a maximal distance down until an object is encountered. You can also indicate whether to consider solid object only or all objects. You typically put this action in the collision event to make sure that the instance stops in contact with the other instance involved in the collision.

 **Bounce against objects**
Cuando colocas esta acción en el evento collision de algún objeto, la instancia rebota en este objeto de manera natural. Si configuras el parámetro **precise** como falso, solo las paredes horizontales y verticales se tratan correctamente. Cuando configuras **precise** a verdadero también se tratan las paredes inclinadas (y aún las curveadas) de manera correcta. Sin embargo esto es más lento. También puedes indicar si el objeto rebotará solo contra objetos sólidos o contra todos los objetos. Por favor nota que el rebote no es completamente correcto porque depende de muchas propiedades. Pero en muchas situaciones el efecto es lo suficientemente bueno.

12.2 Acciones principales, grupo 1 (página / ficha main1)

El siguiente grupo de acciones está relacionado con la creación, el cambio y la destrucción de las instancias de objetos.

 **Create an instance of an object**
Con esta acción puedes crear una instancia de un objeto. Debes especificar el objeto a crear y la posición para la nueva instancia. Si marcas la casilla **Relative**, la posición es relativa a la posición de la instancia actual. La creación de instancias durante el juego es muy útil. Una nave crea balas; una bomba puede crear una explosión, etc. En muchos juegos tendrás un objeto de control que cada cierto tiempo creará enemigos u otros objetos. El evento de creación es ejecutado para la nueva instancia al crearse.

 **Change the instance**
Con esta acción puedes cambiar la instancia actual por otro objeto. Por ejemplo, puedes cambiar una instancia de una bomba en una explosión. Todos los parámetros, como el movimiento o los valores de variables, se mantendrán iguales. Puedes indicar si llevar a cabo o no el evento de destrucción para el objeto actual y el evento de creación para el nuevo.

 **Destroy the instance**
Con esta acción destruyes la instancia actual. Ese ejecuta el evento de destrucción de la instancia.

 **Destroy instances at a position**
Con esta acción puedes destruir todas las instancias cuya caja límite contiene una posición dada. Esto es útil por ejemplo cuando empleas la explosión de una bomba. Cuando marcas la casilla **Relative,** la posición es relativa a la posición de la instancia actual.

 **Change the sprite**
Usa esta acción para cambiar el sprite de la instancia. Debes indicar un nuevo sprite. También puedes indicar un factor de escalado. Un factor de 1 significa que el sprite no es escalado. El factor de escalado debe ser mayor que 0. Por favor recuerda que el escalado reducirá la velocidad del dibujo. El cambio de sprites es una característica importante. Por ejemplo, a menudo querrás cambiar el sprite de un personaje de acuerdo a la dirección a la que se dirige. Esto se puede lograr haciendo diferentes sprites para cada una de las (cuatro) direcciones. En los eventos del teclado para las teclas del cursor estableces la dirección de movimiento y el sprite.

 **Play a sound**
Con esta acción reproduces uno de los recursos de sonido que agregaste al juego. Puedes indicar el sonido a reproducir y si debiera reproducirse solo una vez (el valor por defecto) o hacerlo continuamente Se pueden reproducir varios archivos wave al mismo tiempo, pero sólo puede reproducirse un midi a la vez. Así que si inicia el sonido de un midi, el midi actual se detiene. A menos que el sonido tenga múltiples buffers (ve el Capítulo 17) solo se puede reproducir una instancia del sonido. Por lo que si el mismo sonido se está reproduciendo, se detiene y vuelve a empezar.

 **Stop a sound**
Esta acción detiene el sonido indicado. Si hay varias instancias del sonido reproduciéndose, todas son detenidas.

 **If a sound is playing**
If the indicated sound is playing the next action is performed. Otherwise it is skipped. You can select **Not** to indicate that the next action should be performed if the indicated sound is not playing. For example, you can check whether some background music is playing and, if not, start some new background music. For more information on actions that test certain questions, see Section **¡Error! No se encuentra el origen de la referencia.**.

 **Go to previous room**
Ir al cuarto anterior. Puedes indicar el tipo de transición entre los cuartos. Experimenta para que veas cuál es el que te agrada. Si estás en el primer cuarto, se da un error.

 **Go to next room**
Ir al siguiente cuarto. Puedes indicar la transición

 **Restart the current room**
Se reinicia el cuarto actual. Puedes indicar el efecto de transición.

 **Go to a different room**
Con esta acción vas a un cuarto en particular. Puedes indicar el cuarto y el efecto de transición.

 **If previous room exists**
This action tests whether the previous room exists. If so, the next action is executed. You normally need this test before moving to the previous room.

 **If next room exists**
This action tests whether the next room exists. If so, the next action is executed. You normally need this test before moving to the next room.

12.3 Acciones principales, grupo 2 (página/ficha main2)

Aquí tenemos algunas acciones más, relacionadas con el tiempo, mostrar mensajes al usuario, y al juego en conjunto.

 **Set an alarm clock**
Con esta acción puedes configurar uno de los ocho relojes de alarma para la instancia. Puedes indicar el número de pasos y la alarma. Después del número indicado de pasos, la instancia recibirá el correspondiente evento de alarma. También puedes incrementar o decrementar el valor marcando la casilla **Relative**. Si configures la alarma a un valor menor o igual que 0 la desactivas, por lo que no se genera el evento.

 **Sleep for a while**
Con esta acción puede congelar la escena por un cierto tiempo. Esto usualmente se usa al inicio de un nivel o cuando le das algún mensaje al jugador. Especificas el número de milisegundos a pausar. También indicas si la pantalla debiera ser dibujada primero para mostrar la situación más reciente.

 **Set a time line**
(Only available in advanced mode.) With this action you set the particular time line for an instance of an object. You indicate the time line and the starting position within the time line (0 is the beginning). You can also use this action to end a time line by choosing No Time Line as value.

 **Set the time line position**

(Only available in advanced mode.) With this action you can change the position in the current time line (either absolute or relative). This can be used to skip certain parts of the time line or to repeat certain parts. For example, if you want to make a looping time line, at the last moment, add this action to set the position back to 0. You can also use it to wait for something to happen. Just add the test action and, if not true, set the time line position relative to -1.

 **Display a message**
Con esta acción puedes mostrar un mensaje en una ventana de diálogo. Simplemente tecleas el mensaje. Si usas el símbolo # en el texto del mensaje será interpretado como un carácter de nueva línea. Si el mensaje empieza con una comilla o dobles comillas, se interpreta como una expresión. Ve abajo para tener más información sobre las expresiones. (Nota que esta acción no funciona cuando el juego es ejecutado en modo exclusivo, ve el Capítulo 26)

 **Show the game information**
Con esta acción puedes mostrar la ventana de información del juego. Ve el Capítulo 25 para tener más información acerca de cómo crear la información del juego. (Esta acción no funciona cuando el juego se ejecuta en modo exclusivo).

 **Restart the game**
Con esta acción puedes reiniciar el juego desde el principio.

 **End the game**
Con esta acción termina el juego.

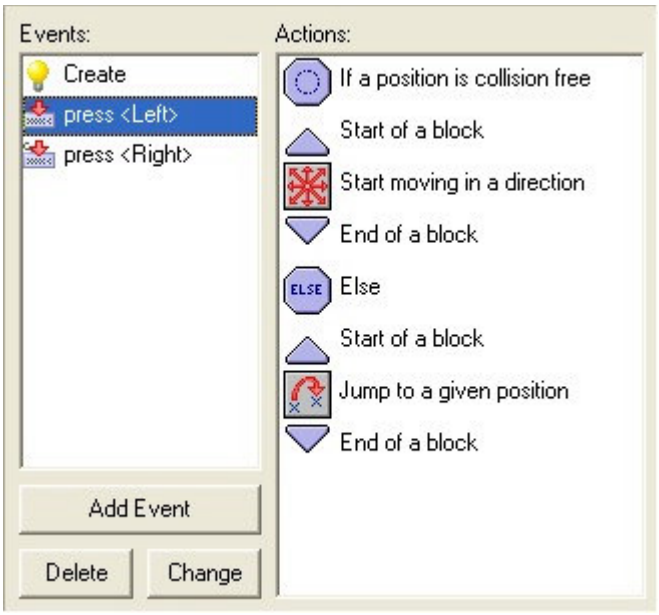
 **Save the game**
Con esta acción puedes guardar el estado actual del juego. Especificas el nombre del archivo a guardar (el archivo es creado en la carpeta de trabajo del juego). Después el juego puede ser cargado con la siguiente acción.

 **Load the game**
Carga el estado del juego desde un archivo. Aquí especificas el nombre del archivo. Asegúrate de que el juego guardado corresponde al mismo juego y que fue creado con la misma versión del *Game Maker*. De otra manera pudiera haber errores (para ser precisos, el juego se carga en el evento step actual. Por lo que aún se realizan algunas acciones después de esta aún son realizadas en el juego actual, ¡no en el cargado!

12.4 Control

Hay varias acciones con las que puedes controlar qué otras acciones se llevan a cabo. La mayoría de estas acciones hacen una pregunta, por ejemplo si es que una posición está vacía. Cuando la respuesta es yes (true) se ejecuta la siguiente acción, de otra forma se

salta esa acción. Si quierers que se ejecuten o salten varias acciones basándote en el resultado puedes colocarlas dentro de un bloque. También puede haber una parte else, la cual se ejecuta cuando la respuesta es no. Así, una pregunta normalmente tiene la siguiente estructura:



Aquí se pregunta si una posición para la instancia actual está libre de colisión. De ser así, la instancia comienza a moverse en una dirección dada. Si no, la instancia es movida a una posición dada.

Para todas las preguntas hay un campo marcado **NOT**. Si lo marcas, el resultado de la pregunta se invierte. Esto es, si el resultado era verdadero se vuelve falso, y si era falso, se vuelve verdadero. Esto te permite realizar ciertas acciones cuando una pregunta no es verdadera.

Para muchas preguntas puedes indicar si deben aplicar a todas las instancias de un objeto en particular. En este caso el resultado es verdadero solo si es verdadero para todas las instancias del objeto. Por ejemplo, puedes checar si para todas las pelotas la posición ligeramente a la derecha está libre de colisión.

Las siguientes preguntas y acciones relacionadas están disponibles. (Nota que todas ellas tienen un icono con forma y color de fondo diferentes de manera que puedan ser distinguidos más fácilmente de otras acciones).

If a position is collision free

Esta pregunta devuelve verdadero si la instancia actual, colocada en la posición indicada no provoca una collision con algún objeto. Puedes especificar si la posición es absoluta o relativa. Puedes también indicar si solo los objetos sólidos o todos los objetos debieran

ser tomados en cuenta. Esta acción es normalmente empleada para checar si la instancia se puede mover a una posición en particular.

 **If there is a collision at a position**
Esta es la inversa de la acción anterior. Devuelve verdadero si hay una colisión cuando la instancia actual es colocada en la posición indicada (de nuevo, ya sea sólo con objetos sólidos o con toda clase de objetos).

 **If there is an object at a position**
Esta pregunta devuelve verdadero si la instancia colocada en la posición indicada encuentra una instancia del objeto indicado.

 **If the number of instances is a value**
Especificas un objeto y un número. Si el número de instancias del objeto es igual al número la pregunta devuelve verdadero. De otra forma devuelve falso. Puedes también indicar que la revisión debiera ser si el número de instancias es menor o mayor que el valor. Esto se usa normalmente para checar si todas las instancias de un tipo en particular han desaparecido. Este es normalmente el momento de terminar el nivel o el juego.

 **If a dice lands on one**
Especificas el número de lados del dado. Entonces si el dado cae en uno, el resultado es verdadero y se realiza la acción. Esto se puede emplear para agregar cierta aleatoriedad en tu juego. Por ejemplo, en cada paso puedes generar con cierta probabilidad una bomba o un cambio de dirección. Entre mayor sea el número de lados del dado, la probabilidad es menor. Puedes emplear números reales. Por ejemplo, si configuras el número de lados a 1.5 la siguiente acción se realizará dos de tres veces. El emplear un número menor que 1 no tiene sentido.

 **If the user answers yes to a question**
Especificas una pregunta. Se muestra una ventana de diálogo al usuario con dos botones: Yes y No. El resultado es verdadero si el jugador contesta yes. Esta acción no puede ser empleada en modo exclusivo; la respuesta entonces sería siempre yes.

 **If an expression is true**
Puedes introducir una expresión. Si la expresión se evalúa como verdadera (esto es, un número mayor o igual a 0.5) se realizará la siguiente acción. Ve más abajo para más información sobre las expresiones.

 **If a mouse button is pressed**
Devuelve verdadero si el botón indicado del ratón es presionado. Un uso estándar está en el evento paso. Puedes checar si un botón del ratón está presionado, y si es así, por ejemplo moverte a esa posición (usa la acción jump to a point con los valores `mouse_x` and `mouse_y`).

 **If instance is aligned with grid**
Devuelve verdadero si la posición de la instancia se encuentra sobre una cuadrícula. Especificas el espaciado horizontal y vertical de la cuadrícula. Esto es muy útil cuando ciertas acciones, como el dar una vuelta, no son permitidas cuando se está alineado a una posición en la cuadrícula.

 **Else**
Detrás de esta acción sigue el else, que se ejecuta cuando el resultado de la pregunta es falso.

 **Start of block**
Indica el inicio de un bloque de acciones.

 **End of block**
Indica el final de un bloque de acciones.

 **Repeat next action**
Esta acción se emplea para repetir la siguiente acción (o bloque de acciones) cierto número de veces. Puedes simplemente indicar el número.

 **Exit the current event**
Cuando se encuentra esta acción, no se realiza ninguna otra en el evento. Normalmente se emplea después de una pregunta. Por ejemplo, cuando una posición está libre no se necesita hacer nada por lo que salimos del evento. En este ejemplo, las acciones siguientes son solo ejecutadas cuando hay una colisión.

12.5 Acciones de dibujo

Las acciones de dibujo sólo tienen efecto en el evento Draw. En otros eventos son simplemente ignoradas. Por favor recuerda que el dibujar otras cosas además de sprites y fondos es relativamente lento. Así que usa esto sólo cuando sea estrictamente necesario.

 **Draw a sprite image**
Indicas el sprite, la posición (ya sea absoluta o relativa a la posición actual de la instancia) y la subimagen del sprite. (Las subimágenes están numeradas de 0 hacia arriba). Si quieres dibujar la subimagen actual, usa el número –1.

 **Draw a background image**
Indicas la imagen de fondo, la posición (absoluta o relativa) y si la imagen debe o no repetirse en todo el cuarto.

 **Draw a rectangle**
Indicas las coordenadas de las dos esquinas opuestas del rectángulo; ya sean absolutas o relatives a la posición actual de la instancia.

 **Draw an ellipse**
Indicas las coordenadas de las dos esquinas opuestas del rectángulo alrededor de la elipse; ya sean absolutas o relatives a la posición actual de la instancia.

 **Draw a line**
Indicas las coordenadas de los dos extremos de la línea; ya sean absolutas o relatives a la posición actual de la instancial.

 **Draw a text**
Indicas el texto y la posición. Un símbolo # en el texto se interpreta como el inicio de una nueva línea. (Usa \# para mostrar el símbolo #). Por lo que puedes crear texto multilínea. Si el texto inicia con comilla simple o doble, es interpretado como una expresión. Por ejemplo, puedes usar

```
'X: ' + string(x)
```

para mostrar el valor de la coordenada x de la instancia. (La variable x guarda la coordenada x actual. La función string() convierte este número en una cadena. + combina las dos cadenas).

 **Set the colors**
Te permite seleccionar el color usado para rellenar los rectánguos y elipses, y el color empleado para las líneas alrededor de los rectángulos y elipses y para dibujar una línea.

 **Set a font for drawing text**
Puedes configurar la fuente que a partir de este momento se empleará para dibujar texto.

 **Change full screen mode**
Con esta acción puedes cambiar el modo de la pantalla de ventana a pantalla completa y viceversa. Puedes indicar si se cambia entre modos o si cambiar a modo de ventana o a modo a pantalla completa. (No funciona en modo exclusivo).

12.6 Acciones de score

En la mayoría de los juegos el jugador tendrá cierto score. También muchos juegos le dan al jugador cierto número de vidas. Finalmente, a menudo el jugador cuenta con cierto nivel de energía. Las siguientes acciones permiten manejar de manera sencilla el score, las vidas y la energía del jugador.

 **Set the score**
El *Game Maker* tiene un mecanismo interno de score. El score normalmente se muestra en el título de la ventana. Puedes usar esta acción para cambiar el score. Simplemente introduces un nuevo valor para éste. A menudo querrás agregar puntos al score. En este caso no olvides seleccionar la opción **Relative**.

 **If score has a value**
Con esta pregunta puedes checar si el score ha alcanzado un valor en particular. Puedes indicar el valor y si el score debe ser igual, menor o mayor a ese valor.

 **Draw the value of score**
Con esta acción puedes dibujar el valor del score en cierta posición de la pantalla. Proporcionas la posición y el título que se mostrará antes del score. El score se dibuja con la fuente actual. Esta acción sólo puede emplearse en el evento draw de algún objeto.

 **Clear the highscore table**
Esta acción 'limpia' la tabla de récords.

 **Display the highscore table**
Cada juego guarda los diez mejores scores. Esta acción muestra la lista de récords. Si el score actual está entre los mejores diez, el nuevo score se inserta y el jugador puede introducir su nombre. Puedes indicar la imagen de fondo a usar, si la ventana tendrá borde, el color para la nueva y las otras entradas, y la fuente a emplear. (¡Esta acción no funciona en modo exclusivo!)

 **Set the number of lives**
El *Game Maker* también cuenta con un sistema interno de vidas. Con esta acción puedes cambiar el número de vidas que quedan. Normalmente lo pones a un valor como 3 al inicio del juego y lo aumentas o disminuyes dependiendo de lo que ocurra. No olvides seleccionar la opción **Relative** si deseas agregar o restar al número de vidas. En el momento en que el número de vidas llegue a 0 (o menos que 0) se genera un evento "no more lives".

 **If lives is a value**
Con esta pregunta puedes checar si el número de vidas ha llegado a un cierto valor. Puedes indicar el valor y si el número de vidas debe ser igual, menor o mayor a ese valor.

 **Draw the number of lives**
Con esta acción puedes dibujar en pantalla en número de vidas. Das la posición y el título que se colocará antes del número de vidas. El número de vidas se dibuja con la fuente actual. Esta acción sólo puede ser usada en el evento draw de algún objeto.

 **Draw the lives as image**
En lugar de indicar el número de vidas con un número, es mejor usar pequeñas imágenes para indicar las vidas. Esta acción hace eso precisamente. Especificas la posición y la imagen, y en la posición indicada se dibuja el número de vidas como imágenes. Esta acción sólo puede emplearse en el evento draw de un objeto.

 **Set the health**
El *Game Maker* tiene un mecanismo interno de energía. Puedes emplear esta acción para cambiar la energía. Un valor de 100 se considera como energía completa y 0 es sin energía. Simplemente das un nuevo valor para la energía. A menuco querrás restar algo a la energía. En este caso no olvides seleccionar la opción **Relative**. Cuando la energía sea igual o menor que 0 se genera un evento "out of health".

 **If health is a value**
Con esta pregunta puedes checar si la energía ha alcanzado cierto valor. Indicas el valor y si la energía debe ser igual, menor o mayor que ese valor.

 **Draw the health bar**
Con esta acción puedes dibujar la energía en forma de una barra. Cuando la energía es 100 la barra es dibujada completa. Cuando es 0 la barra está vacía. Puedes indicar la posición y el tamaño de la barra de energía y el color para la barra y el fondo de la misma.

 **Set the window caption information**
Normalmente en el título de la ventana se muestra el nombre del cuarto y el score. Con esta acción puedes cambiar esto. Puedes indicar si se muestra o no el score, las vidas, la energía y los títulos para estas opciones.

12.7 Acciones relacionadas con código

Finalmente hay varias acciones que tienen que ver principalmente con código.

 **Execute a script**
(Sólo disponible en modo advanced.) Con esta accíón puedes ejecutar un script que hayas agregado al juego. Indicas el script y como máximo 5 argumentos para el mismo. Ve el capítulo 23 para más información sobre los scripts.

 **Execute a piece of code**
Cuando agregas esta acción, se muestra una ventana en la que puedes introducir unz pieza de código. Éste funciona en la misma forma que definir scripts (ve el Capítulo 23). La única diferencia es que puedes indicar para qué instancias se debe ejecutar el código. Usa esta acción para pequeñas piezas de código. Para piezas más largas se te recomienda usar scripts.

 **Set the value of a variable**
Hay muchas variables internas en el juego. Con esta acción puedes cambiar sus valores. También puedes crear tus propias variables y asignarles valores. Indicas el nombre de la variable y un nuevo valor. Cuando seleccionas la opción **Relative**, el valor se suma al valor actual de la variable. ¡Por favor nota que esto sólo puede usarse si la variable ya tiene un valor asignado!. Ve más adelante para más información sobre las variables.

 **If a variable has a value**
Con esta acción puedes checar cuál es el valor cierta variable. Si el valor de la variable es igual al número introducido, la pregunta devuelve true. En cualquier otro caso devuelve false. Puedes indicar si se checará si la el valor es menor o mayor que el dado. Ve abajo para más información sobre las variables. De hecho, también puedes usar esta acción para comparar dos expresiones.

 **Draw the value of a variable**
Con esta acción puedes dibujar el valor de una variable en cierta posición de la pantalla.

 **Call the inherited event**
(Sólo disponible en modo advanced.) Esta acción sólo es útil cuando el objeto tiene un padre (ve el Capítulo 19). Esta acción llama al evento correspondiente en el objeto padre.

 **Comment**
Usa esta acción para agregar una línea de comentarios a la lista de acciones. La línea se muestra en fuente cursive. No hace nada cuando se ejecuta el evento. El agregar comentarios te ayuda a recordar lo que estás haciendo en tus eventos.

12.8 Usando expresiones y variables

En muchas acciones necesitas introducir valores para algunos parámetros. En lugar de introducir un nombre, también puedes introducir una fórumla, p. ej. 32*12. Pero de hecho, puedes introducir expresiones much más complicadas. Por ejemplo, si quieres duplicar la velocidad horizontal, podrías establecerla a `2*hspeed`. Aquí `hspeed` es una variable que indica la velocidad horizontal actual de la instancia. Hay muchas otras variables que puedes emplear. Algunas de las más importantes son:

`x` la coordenada x de la instancia
`y` la coordenada y de la instancia
`hspeed` la velocidad horizontal (en pixeles por step)
`vspeed` la velocidad vertical (en pixeles por step)
`direction` la dirección de movimiento actual en grados (0-360)
`speed` la velocidad actual en esta dirección
`visible` si el objeto es visible (1) o invisible (0)
`image_scale` la cantidad de escala de la imagen (1 = no escalada)

image_single esta variable indica la subimagen del sprite que se debe mostrar; si la pones a –1 (valor por defecto) se hace un ciclo entres las imagines, con otro valor sólo la subimagen (empezando con el número 0) es mostrada todo el tiempo.
image_speed esta variable indica la velocidad con la que se muestran las subimágenes. El valor por defecto es 1. Si haces a este valor mayor que 1 algunas subimágenes serán evitadas para hacer más rápida la animación. Si lo haces menor que 1 la animación será más lenta.
score el valor actual del score
lives el número actual de vidas
health la energía actual (0-100)
mouse_x posición x del ratón
mouse_y posición y del ratón

Puedes cambiar la mayoría de estas variables usando la acción set variable. También puedes definir tus propias variables asignándoles un valor. (No uses relative, porque aún no existen). Luego puedes emplear estas variables en expresiones. Las variables que creas son locales para la instancia actual, esto es, cada objeto tiene su propia copia de ellas. Para crear una variable global, coloca la palabra global y un punto antes de ella.

También te puedes hacer referencias a los valores de las variables de otros objetos colocando el nombre del objeto y un punto antes de la variable. Así por ejemplo, si quieres que una pelota se mueva al lubar donde está una moneda puedes establecer la posición a (`moneda.x` , `moneda.y`). En el caso de un evento de collision puedes hacer referencia a la coordenada x del otro objeto mediante `other.x`. En expresiones condicionales puedes usar comparaciones como < (menor que), >, etc.

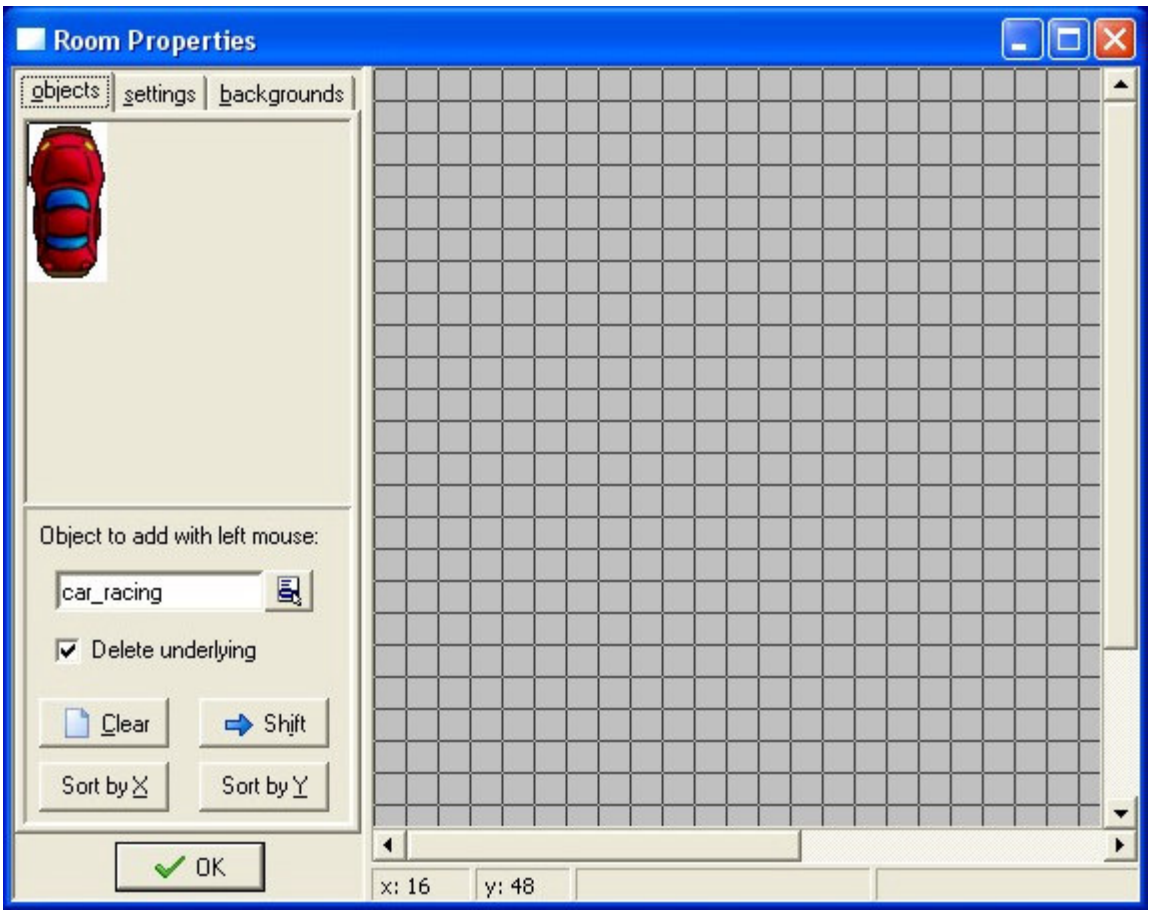
En tus expresiones puedes también emplear funciones. Por ejemplo, la función `random(10)` da un valor aleatorio real menor a 10. Así puedes por ejemplo establecer la velocidad o la dirección de movimiento a un valor aleatorio. Hay muchas más funciones. Para información más precisa sobre expresiones y funciones ve el Capítulo 28 y subsecuentes.

Capítulo 13 Creando cuartos

Ahora que has definido los objetos con su comportamiento en forma de eventos y acciones, es hora de crear los cuartos o niveles en los que se llevara a efecto el juego. Cualquier juego requerirá al menos un cuarto. En estos cuartos colocamos instancias de los objetos. Una vez que el juego inicia se muestra el primer cuarto y las instancias en él toman vida por las acciones en sus eventos de creación.

Hay un gran número de posibilidades al crear cuartos. Además de establecer ciertas propiedades y agregar las instancias de los objetos puedes agregar fondos, definir vistas, y agregar tiles. La mayoría de estas opciones se discuten más adelante en el Capítulo 20. en este capítulo solamente discutiremos las opciones básicas, el agregar instancias de objetos y la configuración de las imágenes de fondo.

Para crear un cuarto, selecciona **Add Room** en el menu **Add**. Se muestra la siguiente ventana:



A la izquierda versa tres fichas (cinco en modo advanced). La ficha **objects** es donde puedes agregar instancias de los objetos al cuarto. En la ficha **settings** puedes indicar

varias opciones para el cuarto. En la ficha **backgrounds** puedes configurar las imagines de fondo para el cuarto.

13.1 Agregando instancias

A la derecha en la ventana de diseño puedes ver el cuarto. Al principio está vacío, con un fondo gris.

Para agregar instancias al cuarto, primero selecciona la ficha **objects** si no está visible. A continuación selecciona el objeto que deseas agregar haciendo clic en el botón con icono de un menu (o haciendo clic en el area de imagen a la izquierda). La imagen del objeto aparece a la izquierda. (Nota que hay una cruz en la imagen. Esta indica cómo se alinearan las instancias con la cuadrícula). Ahora haz clic con el boto´n izquierdo del ratón en el area a la derecha. Aparece una instancia del objeto. Éste se alineará a la cuadrícula indicada. (Puedes cambiar la cuadrícula en las opciones; ve más abajo. Si mantienes presionada la tecla <Alt> mientras colocas la instancia no se alineará a la cuadrícula). Con el botón derecho del ratón puedes remover las instancias. De esta manera es como defines el contenido del cuarto. Si mantienes presionado el botón mientras lo arrastras sobre el cuarto, colocarás o eliminarás varias instancias.

Como te darás cuenta, si colocas una instancia sobre otra, la primera desaparece. Normalmente sera lo que desees, pero no siempre. Esto puede evitarse al deshabilitar la opción **Delete underlying** a la izquierda. Hay tres acciones que puedes realizar usando el botón derecho del ratón: cuando mantienes presionada la tecla <Ctrl> mientras haces clic sobre una instancia con el botón derecho, la instancia al fondo en esa posición será traída al frente. Mantener presionada la tecla <Alt> enviará al fondo la instancia que se encuentre al frente. Esto puede emplerase para cambiar las instancias sobrepuestas. Finalmente, al mantener presionada la tecla <Shift> mientras haces clic con el botón derecho eliminará todas las instancias en esa posición, no sólo la del frente.

Hay cuatro útiles botones en la ficha a la izquierda. Cuando presionas el botón **Clear** se borrarán todas las instancias del cuarto. Cuando presionas el botón **Shift** puedes mover a todas las instancias cierto número de pixeles. Usa valores negativos para moverlas hacia la izquierda o hacia arriba. Esto es útil cuando decidiste p. ej. agrandar el cuarto. (También puedes usar esto para colocar instancias fuera del cuarto, lo cual es útil en ciertas ocasiones). Finalmente hay dos botones para ordenar las instancias por la coordenada X o la Y. Esto es útil cuando se colocan instancias que se sobreponen parcialmente.

13.2 Configuración del cuarto

Cada cuarto tiene ciertas opciones que puedes cambiar haciendo clic en la ficha **settings**. Sólo trataremos las más importantes aquí.

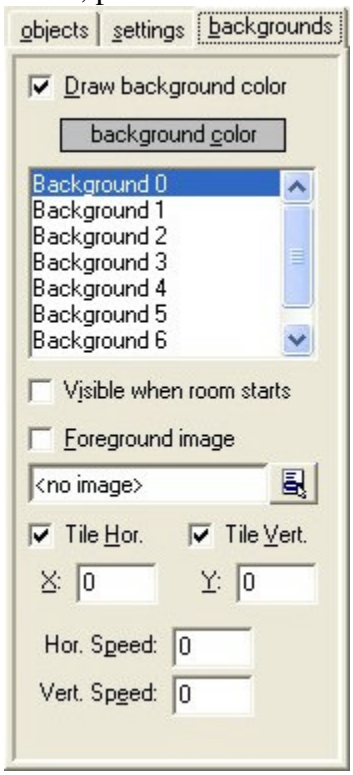
Cada cuarto tiene un nombre, major dale uno con sentido. También hay un título. Este título se muestra en la ventana cuando se ejecuta el juego. Puedes establecer el ancho y alto del cuarto (en pixeles). También puedes configurar la velocidad del juego. Este ese l

número de pasos por segundo. Entre más velocidad, más suave será el movimiento. Pero necesitarás una computadora más rápida para ejecutarlo.

En la parte inferior de la ficha **settings** puedes indicar el tamaño de las celdas de la cuadrícula usada para alinear los objetos. Al hacer clic en el botón **Show** puedes indicar si p. ej. Mostrar las líneas de la cuadrícula. (También puedes indicar aquí si se muestran los fondos, etc. A veces es útil ocultar ciertos aspectos del juego).

13.3 Configurando el fondo

Con la ficha **backgrounds** puedes establecer la imagen de fondo para el cuarto. De hecho, puedes indicar varios fondos. La ficha se ve como sigue:



En la parte superior versa el color de fondo. Puedes hacer clic sobre este para cambiarlo. El color de fondo sólo es útil si no usas una imagen de fondo que cubra todo el cuarto. De otra forma, mejor inhabilita la opción **Draw background color** ya que significaría una pérdida de tiempo.

Arriba versa una lista de 8 fondos. Puedes definir cada uno de ellos pero la mayoría de las veces sólo necesitaras uno o dos. Para definir un fondo, primero selecciona uno en la lista. A continuación marca la opción **Visible when room starts** o de otra forma no lo verás. El nombre del fondo se pondrá en negritas cuando esté definido. Ahora indica una imagen de fondo en el menu. Hay ciertas opciones que puedes cambiar. Primero que nada puedes indicar si la imagen de fondo se repetirá horizontal y/o verticalmente. Puedes también indicar la posición del fondo en el cuarto (esto también influirá el tiling). Finalmente puedes hacer que el fondo se desplace dándole una velocidad horizontal o vertical (pixeles por step).

Hay una opción más llamada **Foreground image**. Cuando la activas, el fondo se colocará en primer plano, y se dibujará al frente de todo en lugar de detrás de todo. Obviamente una imagen de este tipo debiera ser parcialmente transparente para que tenga alguna utilidad.

Capítulo 14 Distribuyendo tu juego

Con la información de los capítulos anteriores puedes crear tus juegos. Una vez que hayas creado un buen juego probablemente quieras darlo a otras personas para que lo jueguen. Puedes distribuir libremente y en la forma que desees los juegos que crees con el *Game Maker*. Incluso puedes venderlos. Lee el acuerdo de licencia adjunto para más información al respecto.

Básicamente hay tres diferentes formas en las que puedes distribuir tus juegos. La manera más fácil es simplemente dar el archivo *.gmd que contiene el juego. Sin embargo la otra persona debe tener el *Game Maker*. (No se te permite distribuir el *Game Maker* con tu juego peros ellos pueden bajarlo gratuitamente desde el sitio web). Además, la otra persona puede hacer cambios al juego.

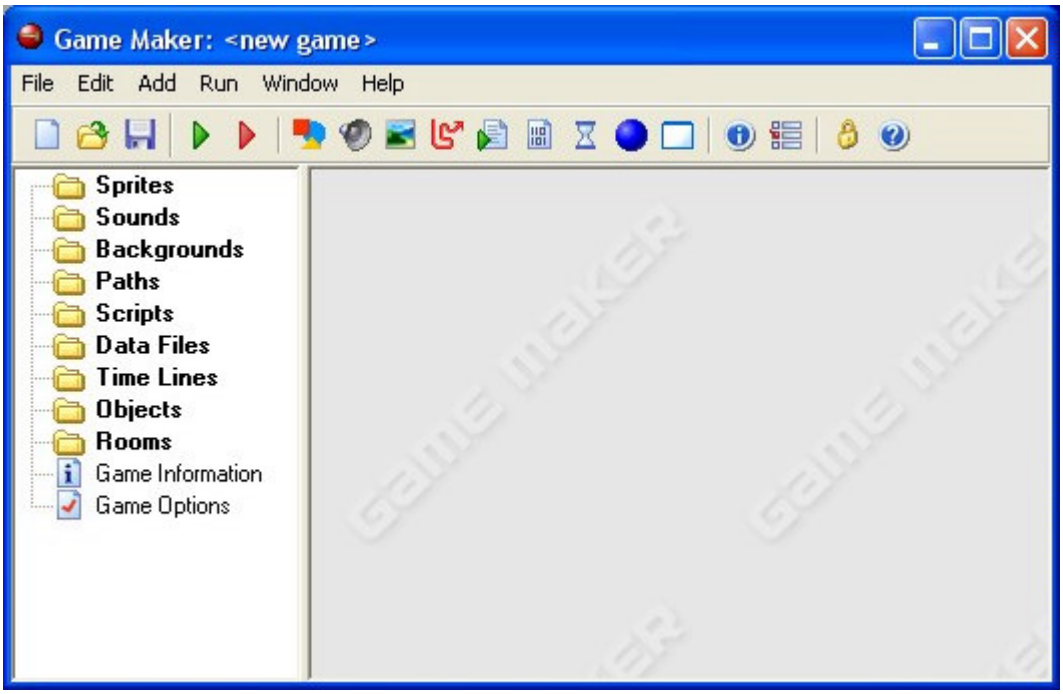
La segunda forma es crear un ejecutable del juego. Esto puede hacerse seleccionando la opción **Create Executable** en el menu **File**. Se te pedirá un nombre para el ejecutable que contendrá el juego. Indica un nombre, presiona **OK** y tendrás un juego ejecutable que puedes dar a quien tú quieras. Puedes configurar el icono para el juego ejecutable en la ventana Game Options. (Si tu juego usa algún otro archivo debieras copiarlos a la carpeta que contiene el ejecutable del juego). Ahora debieras distribuir este archivo a otras personas (tal vez quieras comprimirlo antes).

La tercera opción es crear un archivo de instalación. Hay varios programas freeware para la creación de instaladores disponibles en la red. Lo primero es crear una versión ejecutable del juego y luego emplear el instalador para crear una instalación. Cómo se haga esto depende del instalador que emplees.

Capítulo 15 Modo avanzado

Hasta ahora hemos considerado las opciones más simples de *Game Maker*. Pero hay muchas mas posibilidades. Para habilitarlas debes ejecutar *Game Maker* en Modo Avanzado (**Advanced Mode**).
Es fácil de cambiar. En el **Menú File**, click en el ítem **Advanced Mode**. (Para ver los cambios debes reiniciar *Game Maker* o al menos guardar tu juego y volver a cargarlo)

Cuando comience *Game Maker* en **Advanced Mode**, el siguiente formulario se mostrara:



Contiene todo lo que estaba en **Simple Mode**, pero hay un número de recursos adicionales, botones e ítems de menú. También, como veremos en los siguientes capítulos los diferentes recursos tienen opciones adicionales. Allí analizaremos los diferentes ítems del menú.

15.1 Menú File

En el **Menú File** puedes encontrar los siguientes comandos:

- **Import Scripts** (Importar Scripts): Se usa para importar útiles scripts desde archivos. Esto será analizado con más detalles en el capitulo 21.
- **Export Script** (Exportar Scripts): Se usa para guardar scripts en archivos para que puedan ser usados por otros. Otra vez, chequea el capitulo 21.
- **Merge Game** (Unir Scripts): Con este comando puedes unir todos los recursos (sprites, sonidos, objetos, cuartos, etc.) de otro juego en tu actual proyecto. Esto es bastante útil se deseas hacer algo que quieras volver a usar (Ej.: **Menú Systems**). (Fíjate que todos los recursos, instances y tiles tendrán nuevos nombres lo que puede causar problemas si los usas en scripts).Es tu responsabilidad hacer que sea

seguro que los recursos en los dos diferentes archivos tengan distintos nombres, de otro modo, podrían aparecer problemas.

- **Preferences** (Preferencias): Aquí tú puedes poner un número de preferencias sobre *Game Maker*. Estas serán recordadas entre diferentes llamadas de *Game Maker*. Las siguientes preferencias se podrán elegir:
    - **Show recently edited games in the file menú** (Mostrar los juegos recientemente editados en el **Menú File**): Si esta chequeada los ocho juegos mas recientemente editados serán mostrados debajo de **recent files** en el **Menú File**.
    - **Load last opened file on startup** (Cargar el último abierto en el comienzo): Si esta chequeada cuando comiences *Game Maker* el archivo abierto más recientemente será abierto automáticamente.
    - **Keep backup copies of files** (Crear copias de respaldo): Si esta chequeada el programa creara copias de respaldo de tu juego con las extensiones ba0-ba9. Puedes abrir esos juegos con *Game Maker*.
    - **Maximal number of backups** (Máximo numero de copias): Aquí indicas cuantas (1-9) diferentes copias de respaldo deben ser recordadas por el programa.
    - **Hide the designer and wait while the game is running** (Ocultar el diseñador y esperar mientras el juego se este ejecutando): Si esta chequeada, cuándo ejecutes un juego, la ventana designer desaparerecera y volverá cuando el juego sea terminado.
    - **Run games in secure mode** (Ejecutar el juego en modo seguro): Si esta chequeada, cualquier juego creado con *Game Maker* que rule en tu maquina, no permitirá ejecutar programas externos, o cambiar o eliminar archivos en un lugar diferente del que se encuentre el juego (Esto es una garantía contra Troyanos).Chequeándola puede hacer que ciertos juegos no se ejecuten correctamente.
    - **Show the origin and bounding box in the sprite image** (Mostrar el origen y la **bounding box** en el sprite): Si esta chequeada, en el formulario de propiedades del sprite, el origen y la **bounding box** del sprite será indicada.
    - **In object properties, show hints for actions** (En las propiedades del objeto, mostrar descripción para las acciones): Si esta chequeada, en el formulario de propiedades del objeto, cuando pases el puntero del mouse sobre alguna de las acciones, una descripción será mostrada.
    - **Remember room settings when closing the form** (Recordar la configuración del cuarto cuando se cierre el formulario): Si esta chequeada, un numero de opciones del cuarto; como **whether to show the grid**, **whether to delete underlying objects**, etc. serán recordadas cuando edites el cuarto mas adelante.
    - **External sound editors** (Editores externos de sonido): Puedes indicar cual editor externo quieres usar para los diferentes tipos de sonido. (Fíjate que *Game Maker* no tiene un editor interno de sonido, por lo que si no indicas ninguno no podrás editar los sonidos)

- **Scripts and code and colors** (Scripts y código en colores): Fíjate mas abajo sobre esta preferencia.
- **Image editor** (Editor de imágenes): Por defecto *Game Maker* tiene un editor interno de imágenes tú tienes un mejor programa editor de imágenes puedes indicarlo aquí para usar un programa diferente para edita imágenes.

15.2 Menú Edit

En el **Menú File** puedes encontrar los siguientes comandos:

- **Insert group** (Insertar grupo): Los recursos pueden ser agrupados juntos. Esto es bastante útil si tú haces un juego grande. Por ejemplo, puedes poner todos los sonidos relativos a un objeto juntos en un grupo, o puedes agrupar todos los objetos que usas en un nivel en particular. Este comando crea un nuevo grupo en el tipo de recurso seleccionado actualmente. Te preguntaras por el nombre. Los grupos pueden contener otros grupos dentro de ellos, etc. Como indica bajo puedes arrastrar recursos hacia los grupos.
- **Find Resource** (Buscar recurso): Con este comando pones el nombre del recurso y abre el formulario de propiedades correspondiente.
- **Show Object Information** (Mostrar información del objeto): Usando este comando puedes obtener una vista general de todos los objetos en el juego.

15.3 Menú Add

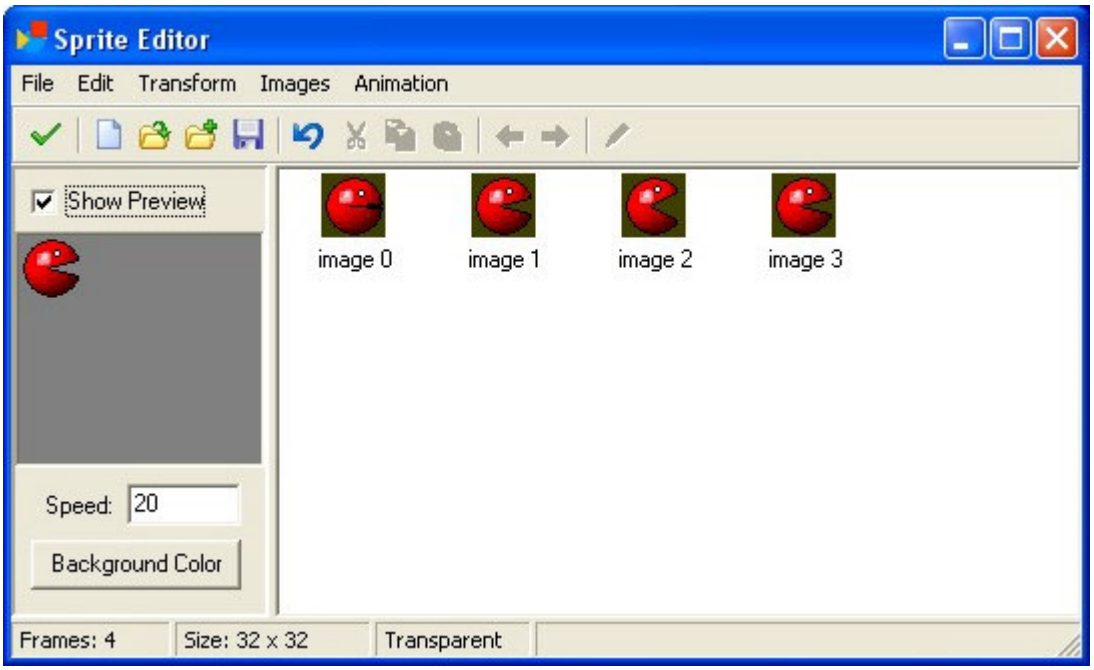
Con este menú puedes añadir recursos adicionales. Fijate que para cada uno de ellos hay un botón en la barra de herramientas y un método abreviado del teclado.

Capítulo 16 Más acerca de los sprites

Hasta ahora hemos cargado nuestros sprites desde archivos. Sin embargo, también es posible crearlos y modificarlos con *Game Maker*. Para hacer esto, abre la ventana de propiedades del sprite haciendo doble click en uno de tus sprites (o creando uno nuevo).Ahora presiona el botón con la etiqueta **Edit Sprite**. Un nuevo formulario aparecerá mostrando todas las sub-imágenes que componen el sprite.

16.1 Editando tus sprites

El formulario de edición de sprites luce así:



A la derecha veras las imágenes que componen el sprite. Fíjate que en *Game Maker* todas las sub-imágenes del sprite deben tener el mismo tamaño. A la izquierda se ve una vista previa de la animación. (Si no ves la animación chequea el botón **Show Preview**. Abajo de la animación podrás cambiar la velocidad de esta y el color de fondo. De esta manera, tendrás una idea sobre como se vera la animación en el juego. (Fíjate que esta velocidad es solo para la vista previa, la velocidad de la animación durante el juego dependerá de la velocidad del cuarto.

El editor de sprites contiene muchos comandos para crear y cambiar el sprite. Estos se darán a través de los menús. (Para algunos hay botones en la barra de herramientas.) Algunos comandos trabajan en imágenes individuales. Esto requiere que primero selecciones una sub-imagen con el mouse.

16.1.1 File menu

El **Menú File** contiene un número de comandos relacionados con cargar y salvar sprites.

- **New**. (Nuevo): Crear uno nuevo, un sprite vacio. Debes indicar el tamaño del sprite. (Recuerda que todas las imágenes del sprite deben tener el mismo tamaño).
- **Create from file**. (Crear desde archivo): Crear el sprite desde un archivo. Muchos tipos de archivos pueden ser usados. Crean una imagen que consiste de una imagen simple, excepto los Gif animados que se separaran en sub-imagenes. Por favor fíjate que el color de la transparencia es tomado del píxel de la esquina izquierda-fondo, no de la transparencia del archivo gif.
- **Add from file** (Añadir desde archivo): Añade una imagen (o imágenes) desde un archivo al actual sprite. Si la imagen no tiene el mismo tamaño puedes elegir el lugar donde colocarlo o reducirlo o achicarlo.
- **Save as Gif** (Guardar como gif): Salva el sprite como un gif animado.
- **Save as strip** (Guardar como Strip): Salva el sprite como mapa de bits (*bmp) con una imagen al lado de la otra con su respectivo orden.
- **Create from strip** (Crear desde Strip): Te permite crear un sprite desde un strip. Chequea mas abajo para más información.
- **Add from strip** (Añadir desde Strip): Usa este para añadir imágenes desde un strip. Chequea abajo.
- **Close saving changes** (Cerrar guardando cambios): Cierra el formulario y salva los cambios hechos en el sprite. Si no quieres guardar los cambios, clickea en el botón de **Cerrar** la ventana.

16.1.2 Menú Edit

El **Menú Edit** contiene un número de comandos que trabajan con la selección actual del sprite. Puedes cortar hacia el portapapeles, pegar una imagen desde el portapapeles, limpiar el sprite actual, borrarlo y mover sprites hacia la derecha u izquierda en la secuencia. Finalmente, hay un comando para editar imágenes individualmente usando el editor interno (Chequea abajo)

16.1.3 Menú Transform

En el **Menú Transform** puedes realizar un número de transformaciones de las imágenes.

- **Mirror horizontal**. (Espejo horizontal): Voltea la imagen horizontalmente.
- **Flip Vertical** (Espejo vertical): Voltea la imagen verticalmente.
- **Shift** (Mover): Aquí tú puedes mover las imágenes en un indicado monto horizontalmente o verticalmente.
- **Rotate** (Rotar): Puedes rotar las imágenes 90 grados, 180 grados, o hacia un monto arbitrario. En el último de los casos puedes elegir la calidad. Experimenta para encontrar los mejores efectos.
- **Resize Canvas** (Redimensionar el contenedor de imágenes): Aquí tú puedes cambiar el tamaño del fondo del contenedor de imágenes. También puedes indicar donde se ubicara la imagen vieja en el nuevo contenedor.
- **Stretch** (Reducir o agrandar): Aquí puedes reducir o agrandar la imagen a un nuevo tamaño. Puedes indicar el factor de la escala y la calidad.

- **Scale** (Escalar): Este comando escala la imagen (pero no el tamaño de la imagen!). Puedes indicar el factor de la escala, la calidad y la posición de la imagen actual en las escaladas.

16.1.4 Menú Images

En el **Menú Images** puedes realizar una numero de operaciones con las imágenes.

- **Cycle left**. (Ciclo izquierdo): Hace un ciclo de animación hacia la una posición en la izquierda. Esto efectivamente comienza la animación en un punto diferente.
- **Cycle Right** (Ciclo derecho): Hace un ciclo de animación hacia la una posición en la derecha.
- **Black and White** (Blanco y negro): Convierte un sprite en blanco y negro (no afecta el color de transparencia!).
- **Colorize** (Colorizar): Aquí puedes cambiar el color de las imágenes. Utiliza el deslizador para elegir diferentes colores.
- **Intensity** (Intensidad): Cambia la intensidad proveyendo valores para el color de saturación y la intensidad de la imagen.
- **Fade** (Desvanecer): Debes especificar un color y un monto. Los colores de las imágenes se desvanecerán.
- **Transparency** (Transparencia): Aquí puedes indicar un nivel del umbral de transparencia. Esto hará un número de píxeles transparentes.
- **Blur** (Borronear): Borroneando las imágenes los colores se mezclaran un poco, haciéndolos mas vagos. Mientras mas alto sea el valor, habrá mayor borroneo de la imagen.
- **Crop** (Recortar): Esto hace la imagen lo mas pequeña posible. Esto es muy útil, ya que, a mayor imagen, mayor memoria de video usa *Game Maker*. Tienes que dejar un pequeño borde alrededor de la imagen para evitar problemas de transparencia.

Experimenta con estos comandos para obtener el sprite que tú quieres.

16.1.5 Menú Animation

Con el **Menú Animation** podrás crear nuevas animaciones partiendo de la animación actual. Hay muchas opciones y debes experimentar un poco con ellas para crear los efectos que quieres. Tampoco no olvides que siempre puedes guardar las animaciones y después añadirlas en la animación actual. También siempre puedes añadir imágenes vacías y eliminar las que desees. Haremos un pequeño recorrido a través de las diferentes posibilidades.

- **Set Length**. (Cambiar longitud): Aquí puedes cambiar la longitud de la imagen. La animación se repetirá para crear la cantidad de frames que indiques.(Normalmente deberás buscar un múltiplo de la cantidad de frames que utilizas.)

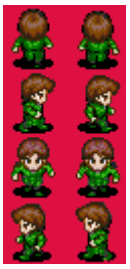
- **Stretch** (Aumentar o reducir): Este comando también cambia la longitud de la animación. Pero ahora, los frames serán duplicados o borrados hasta obtener el número correcto. Asi que, si aumentas el número de frames la animación será más lenta y si lo reduces será más rápida.
- **Reverse** (Reversa): Como imaginas, este comando le da un sentido inverso a la animación. Esto significa que se reproducirá de manera inversa.
- **Add Reverse** (Añadir reversa): Esta vez la animación inversa será añadida, duplicando el número de frames. Es muy útil para hacer un objeto vaya de derecha a izquierda, cambiar el color y reestablecerlo, etc. A veces deberás borras algunos frames para que la animación sea correcta.
- **Translation Sequence** (Secuencia de translación): Crea una animación en la cual la imagen rota o gira. Puedes elegir en el sentido del reloj o en contra. Especifica el numero de frames y la cantidad de grados del ángulo (360º es el giro completo). (Quizás necesites redimensionar el fondo para asegurarte que la imagen este completa durante la rotación).
- **Colorize** (Colorizar): Crea una animación que vuelve la imagen a un color en particular.
- **Fade to color** (Desvanecer hacia un color): Crea una animación que decolora la imagen hacia un color en particular.
- **Disappear** (Desaparecer): Hace que una imagen desaparezca usando el umbral de transparencia.
- **Shrink** (Encoger hasta desaparecer): Encoge la image hacia desaparecer. Debes indicar la dirección.
- **Grow** (Expandir): Expande la imagen desde el sprite vacío.
- **Flatten** (Achatar): Achata la imagen hacia desaparecer en una dirección dada.
- **Raise** (Engrosar): Engrosa la imagen en una dirección dada.
- **Overlay** (Mezclar): Mezcla la animación con otra animación o un archivo de imagen.
- **Morph** (Transformar): Transforma la animación hacia otra o una imagen de un archivo. Fíjate que al transformar trabajaras mejor si las dos animaciones cubren la misma área en la imagen. De otra manera, ciertos píxeles desaparecerán y otros aparecerán.

Los dos últimos comandos son particularmente muy poderosos. Por ejemplo, para explotar un objeto, añade un numero de copias y luego un numero de frames vacíos. Entonces mezclalo con una animación de una explosión. (asegurate que el número de imágenes concuerde). Alternativamente trasfórmalo en explosión. Con algo de práctica podrás hacer sprites grandiosos.

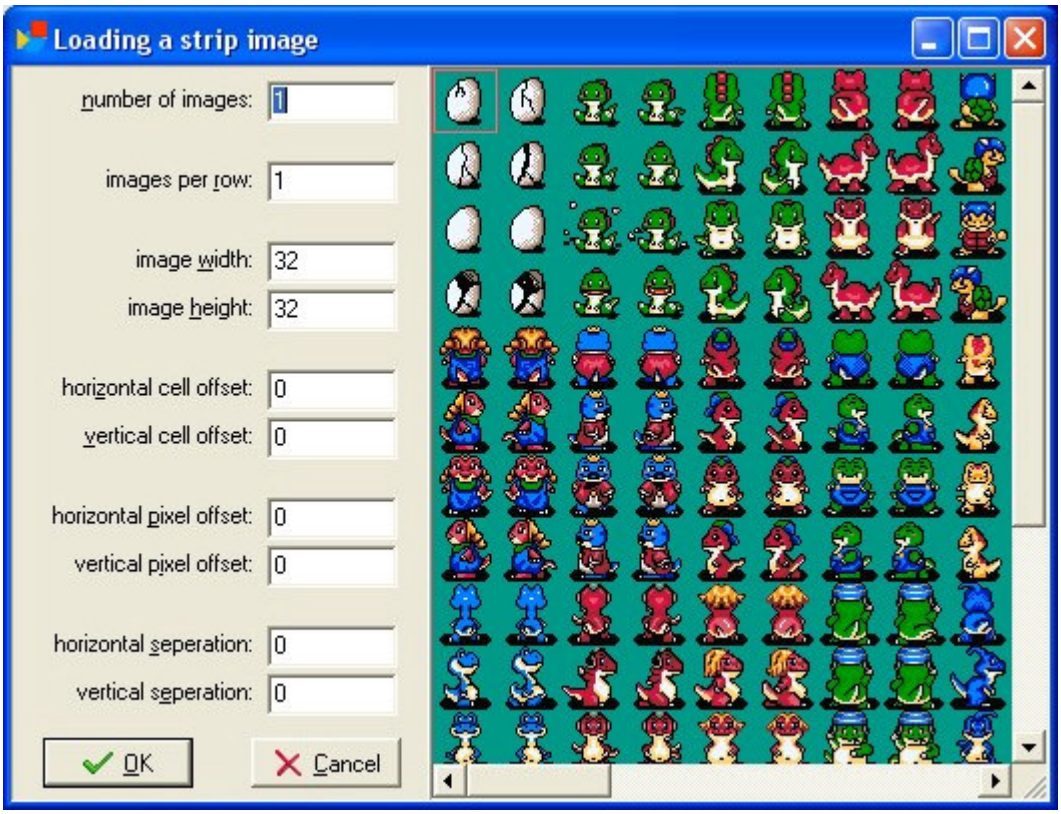
16.1.6 Strips

Como indicamos arriba, normalmente los sprites son almacenados en archivos gif o en strips. Un strip es un bitmap grande que almacena todas las imágenes en un correcto orden. El único problema es que el tamaño de las imágenes individuales no es almacenado en la imagen. Muchos strips se encuentran disponibles en la **Web** en un solo

archivo. Por ejemplo, en la siguiente pieza de strip se encuentran cuatro animaciones diferentes.



Para seleccionar sprites individuales de estos archivos, puedes seleccionar **create from strip** o **Add from strip** en el **Menú File**. Después de indicar el strip apropiado veras el siguiente formulario:



A la derecha puedes ver una parte del strip que has seleccionado. A la izquierda puedes indicar un numero de parámetros que especifican las imágenes en las que estas interesado. Fíjate que uno o más rectángulos en la imagen indican las imágenes que seleccionaste. Los siguientes parámetros puedes ser especificados:

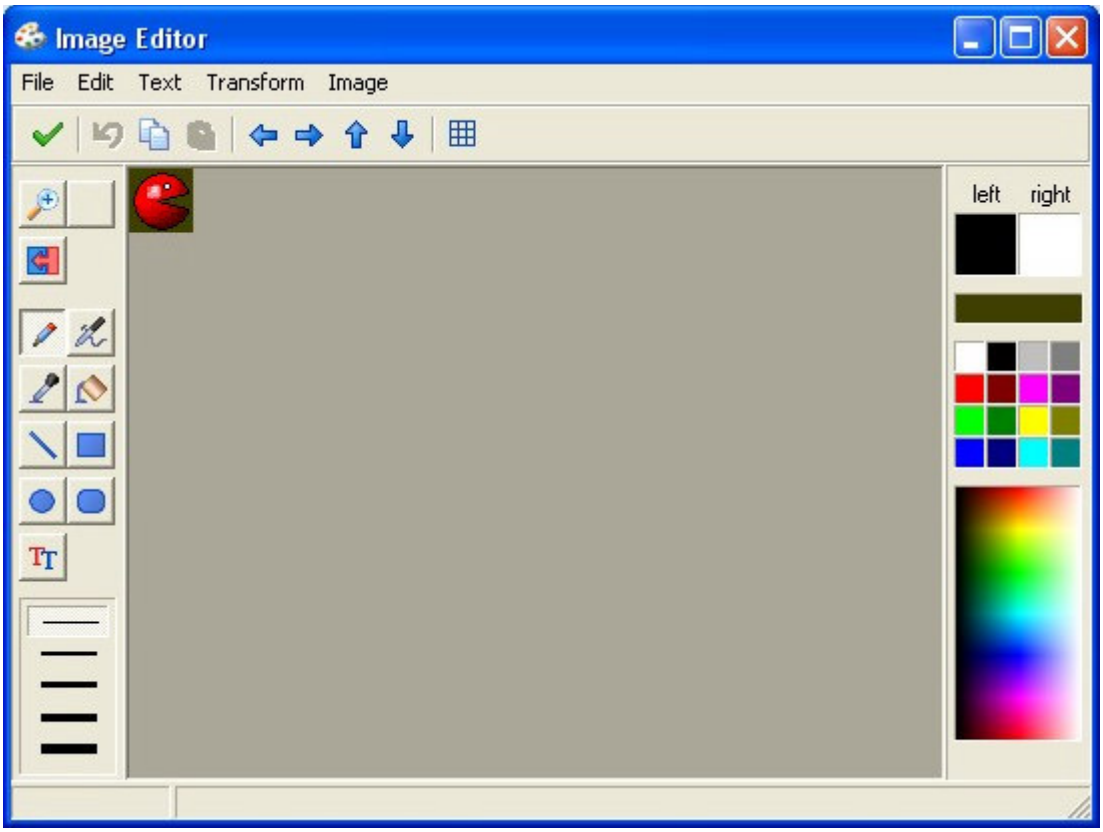
- **Number of images**. (Numero de imágenes): Es el número de imágenes que deseas tomar desde el strip.

- **Images per row** (Imágenes por fila): Cuántas imágenes se toman en cada fila. Por ejemplo, determinando 1 estará seleccionando una secuencia vertical de imágenes.
- **Image width** (Ancho de imagen): El ancho de cada imagen individual.
- **Image height** (Alto de imagen): El alto de cada imagen individual.
- **Horizontal cell offset** (Offset horizontal de celda): Si no deseas seleccionar la imagen superior izquierda, debes indicar cuántas imágenes se saltean horizontalmente.
- **Vertical cell offset** (Offset vertical de celda): Aquí puedes indicar cuántas imágenes salta verticalmente.
- **Horizontal píxel offset** (Offset horizontal de píxel): Algunas veces existe un espacio adicional en el ángulo superior izquierdo. Aquí puedes indicar esta cantidad (en píxeles)
- **Vertical píxel offset** (Offset vertical de píxel): Cantidad vertical de espacio extra.
- **Horizontal separation** (Separación horizontal): En algunos strips existen líneas vacías entre las imágenes. Aquí indicarás la cantidad de píxeles que se saltan entre las imágenes.
- **Vertical separation** (Separación vertical): Cantidad de píxeles verticales que se saltan entra imágenes.

Una vez que selecciones el conjunto de imágenes correcto, presiona OK para crear tu sprite. Recuerda que solo tienes permitido usar imágenes de otro si tienes el permiso o si son freeware.

16.2 Editando sub-imágenes individuales

También puedes editar individualmente las sub-imágenes. Para esto selecciona una sub-imagen y elige **Edit image** del **Menú Image**. Esto abrirá el pequeño editor de imágenes interno de *Game Maker*. Por favor, recuerda que se trata de un programa limitado para tratar imágenes pequeñas y no para crear nuevas. Por eso, es mejor usar un programa de edición mas completo y usar las opciones de cortar o copiar y pegar para insertar la imagen dentro de *Game Maker*.



El formulario muestra la imagen en el medio y un número de botones con operaciones básicas de dibujo a la izquierda. Aquí tú puedes hacer zoom, ampliando o reduciendo, dibujar píxeles, líneas, rectángulos, textos, etc. Fíjate que el color depende de lo que selecciones con el botón derecho o izquierdo del mouse. Para algunas herramientas de dibujo puedes seleccionar opciones (como el tamaño de la línea o si desees el borde visible o no). Existe un botón que permite cambiar el color de algunos píxeles por otro. Es muy útil para cambiar el fondo para ser usado como transparencia. En la barra de herramientas, algunos botones especiales permiten mover los píxeles en una dirección particular. También puedes ocultar o mostrar una grilla cuando la imagen se amplía (Trabaja solamente con un factor de al menos 4).

A la derecha de el formulario puedes seleccionar el color que va a ser usado (uno con el botón derecho del mouse y otro con el izquierdo). Hay cuatro formas de cambiar el color. Primero que todo puedes clickear con el botón del mouse (derecho o izquierdo) en uno de los 16 colores basicos. Fijate que hay una caja especial de color que contiene el color del píxel superior izquierdo de la imagen que es usado como color de transparencia si el sprite es transparente. Puedes usar este color si deseas que una parte del sprite sea transparente. La segunda manera es hacer un click en la imagen con cambio de color. Aquí puedes elegir entre muchos mas colores. Puedes mantener algunos de los botones del mouse oprimido para ver el color que estas eligiendo. Tercero, puedes clickear con el botón izquierdo del mouse en las cajas indicando el color izquierdo o derecho. Una caja de dialogo de colores aparecerá para elegir el color. Finalmente, puedes usar la

herramienta gotero y hacer click sobre la posición en la imagen para copiar el color que allí existe.

En el menú puedes encontrar los mismos comandos de transformación y cambio de imágenes que están disponibles en el editor de sprites. (Cuando un sprite tiene múltiples imágenes, los comandos que cambian el tamaño, como stretcht, no están disponibles).
También puedes guardar la imagen como bitmap. Otros comandos adicionales se encuentran el **Menú Image**:

- **Clear** (Limpiar)**:** Limpia la imagen con el color izquierdo (Que automáticamente convierte la imagen en transparente).
- **Gradient fill**. (Cambio gradual de color): Con este comando puedes rellenar la imagen con un cambio gradual de color (No muy útil para hacer sprites, pero puede ser usado para backgrounds que usen el mismo programa de dibujo).

Fíjate que no hay un mecanismo para seleccionar partes de una imagen. También algunas rutinas elegantes de dibujo no están disponibles. Para esto deberías usar un programa de edición de dibujo más avanzado (o el Paint que viene con Windows). La forma más fácil de hacer esto es copiar la imagen en el portapapeles. Ahora en tu programa de edición de dibujo, usa pegar para obtener la imagen. Luego puedes volver a pegar la imagen en *Game Maker* y así obtenerla actualizada.

16.3 Configuración avanzada de sprites

En **Advanced Mode**, en el formulario de propiedades del sprite hay un número de opciones avanzadas que aquí trataremos.

En primer lugar hay opciones relacionadas con el control de colisión. Si dos instancias se encuentran un evento de colisión es generado. Son controladas de la siguiente manera. Cada sprite tiene una caja que lo rodea. Esta caja contiene las partes no-transparente de todas las sub-imágenes. Cuando las cajas se superponen se chequean los píxeles de las sub-imagines para ver si se superponen. Esta operación consume bastantes recursos en cuanto a memoria se refiere. Así que, si no estas interesado en una colisión precisa para cierto sprite, desactiva la casilla **Precise collision checking**. En este caso solo se chequea la caja que rodea al sprite. También puedes cambiar las propiedades de dicha caja. Aunque es difícilmente requerido, pero a veces querrás hacer una caja más pequeña, con lo cual las colisiones que se generan en los bordes de la caja no serán controladas.

Los sprites pueden ser almacenados en dos lugares: en la memoria de video y en la memoria Standard. La memoria de video se encuentra en la tarjeta grafica y generalmente es más rápida. Así que si tienes muchas instancias del sprite preferirás almacenarla ahí. Pero el monto de la memoria de video es limitado, dependiendo de la tarjeta grafica que el jugador tenga. Así que te recomendamos no almacenar los sprites más grandes en la memoria de video.

Algunos sprites quizás los uses en uno o dos niveles de tu juego. Es un poco incomodo mantener estos sprites en la memoria de video todo el tiempo. En este caso puedes activar

la casilla **Load only on use**. El sprite ahora es cargado en el momento en que se lo necesite. Al terminar el cuarto es descartado y se libera memoria. Para juegos grandes con muchos sprites es importante manejar cuidadosamente cuales sprites son cargados y cuales almacenados en la memoria de video. (También puedes cargar y descartar sprites desde piezas de código).

Finalmente puedes indicar el origen del sprite. Este es el punto en que el sprite corresponde a una posición. Cuando designes una instancia en una posición particular, el origen del sprite se ubicara allí. Predefinidamente esta en el ángulo superior izquierdo de sprite pero a veces es más conveniente usar el centro u otro punto importante. También puedes elegir un punto afuera del sprite, o marcar el origen clickeando en el sprite (cuando el origen es mostrado en la imagen).

Capítulo 17 Más sobre los sonidos y música

Cuando añades un recurso de sonido a tu juego hay un número de aspectos que puedes indicar. Están solo visibles en **Advanced Mode**.

Todos los sonidos que desees utilizar pueden ser cargados como **Load only on use**. Esto es predefinido para archivos midi pero no para archivos wav. Si chequeas esta casilla el sonido no se cargara en la memoria cuando comience el juego. Solo se cargara cuando se necesite. Aunque hace un poco mas lento el juego, pero salva una gran cantidad de memoria y hace que el juego se cargue mas rápido. También, al finalizar el cuarto, el sonido es descartado y se libera memoria. Solo si es requerido otra vez se cargara de nuevo. No uses esto para efectos de sonido cortos, pero es recomendado para músicas de fondo o para sonidos que se usan eventualmente.

Para archivos wav puedes indicar el número de búferes. Este número indica el número de veces que un sonido puede ejecutarse simultáneamente. Por ejemplo, cuando tienes algún sonido de explosión y un número de explosiones deben escucharse simultáneamente, querrás incrementar este numero para que varias explosiones se escuchen simultáneamente. Ten cuidado, ya que los búferes múltiples consumen (dependiendo de la tarjeta de sonido) mas memoria.

También puedes indicar los sonidos que se prepararan para ser efectos de sonido. Esos efectos, como cambiar el canal de audición y cambiar el volumen, podrán solo ser usados desde el código. Estos requieren más recursos.

Game Maker no tiene un editor interno de sonidos. Pero en **Preferences** puedes indicar los editores externos que quieras para editar tus sonidos. Si los indicas, puedes presionar en el botón **Edit Sound** para editar los sonidos que estés usando. (La ventana de *Game Maker* se ocultara mientras edites tus sonidos y se reestablecerá cuando cuando cierres el editor de sonidos).

Además de archivos wav y midis, actualmente hay un tipo diferente de archivos de sonidos: los mp3. Son archivos wav comprimidos. Si bien, no los ves cuando eliges un archivo de sonido puedes usarlos en *Game Maker*. Selecciona mostrar todos los archivos en la parte inferior de la caja de diálogo y podrás cargarlos. Se cuidadoso, ya que hay un numero de desventajas. Primero, necesitan ser descomprimidos y esta operación consume tiempo y recursos haciendo más lento el juego. El hecho de que el tamaño del archivo sea más chico no significa que use menos memoria. Segundo, no todas las maquinas lo soportan. Entonces tu juego no podría ejecutarse en cualquier maquina. Preferentemente no los uses, es mejor convertirlos a wav. Si todavía quieres usarlos usalos solamente como música de fondo.

Capítulo 18 Más acerca de los fondos

Además de importarlos desde archivos existentes, también tienes la posibilidad de crear tus propios fondos. Para este fín, presiona el botón **Edit Background (Editar Fondo)**. Se abrirá un programa de dibujo que viene incluído en el que puedes crear o editar tus fondos. Es importante remarcar que éste no es un programa sofisticado de diseño. Para realizar tareas de diseño más avanzadas y disponer de herramientas de dibujo más sofisticadas utiliza algún programa especializado en ello. El programa incluído se describe más ampliamente en la Sección **16.2**. Hay una opción que es particularmente útil, en el menú **Image (Imagen)** encontrarás un comando llamado **Gradient Fill (Relleno Degradado)**; Con esta opción se pueden crear muy buenos fondos.

En el modo avanzado, la ventana de propiedades de fondo, contiene algunas opciones avanzadas.

Por defecto, los fondos son almacenados en la memoria de video. Esto no implica mayores complicaciones si éstos son pequeños pero cuando se utilizan fondos de dimensiones más grandes será mejor utilizar la memoria normal a cambio. Esto reducirá ligeramente la velocidad de carga del mismo, pero se debe considerar que la memoria de video es limitada. Para no utilizar ésta, la casilla **Use video memory (Usar memoria de video)** no deberá estár seleccionada.

También por defecto, los fondos se cargan en la memoria cuando se necesitan y se descargan de nuevo al salir de la habitación. Esto ahorra grandes cantidades de memoria pero hará ligeramente más lentos los inicios de las habitaciones y pueden provocar un pequeño salto cuando se cambia el fondo a la mitad de una de ellas. Para evitar que esto suceda, no seleccione la casilla **Load only on use (Cargar sólo al usar)**.

Capítulo 19 Más acerca de los objetos

Cuando se añade un objeto nuevo en modo avanzado, se pueden configurar algunas opciones más avanzadas.

19.1 Depth (Profundidad)

Antes que nada, se podrá fijar la profundidad de las instancias del objeto en cuestión. Cuando éstas se dibujan en pantalla, se muestran en orden de acuerdo a su profundidad. Las instancias con mayor profundidad se dibujarán primero y aquellas con menor profundidad, al final. Cuando varias instancias tienen la misma profundiad, se dibujan en el orden en que fueron creadas. Si se quiere asegurar que un objeto se encuentra encima de los demas déle una profundidad negativa, si se quiere que siempre se encuentre siempre debajo de los otros, déle una profundidad positiva muy grande. También se puede cambiar la profundidad de una instancia en particular durante la ejecución del juego utilizando la variable `depth`.

19.2 Persistent objects (Objetos persistentes)

En segundo término, se puede hacer persistente a un objeto. Un objeto persistente seguirá existiendo aún si se pasa de una habitación a la otra. Solamente desaparecerá cuando es destruída expresamente. De este modo, solo es necesario colocar una instancia del objeto en la primera habitación y continuará apareciendo en todas las habitaciones. Esto es muy útil cuando, por ejemplo, se tiene un personaje principal que se mueve de habitación en habitación. La utilización de objetos persistentes es un mecanismo poderoso pero también puede producir fácilmente errores.

19.3 Parents (Padres)

Todos los objetos pueden tener un objeto padre. Cuando un objeto tiene a otro como su padre, hereda el comportamiento de éste. Dicho de otra manera, el objeto es una especie de variante del objeto padre. Por ejemplo, si se tienen cuatro pelotas distintas, nombradas ball1, ball2, ball3 y ball4, y todas se comportan (interactúan) de la misma forma pero tienen un sprite distinto, se puede hacer al objeto ball1 padre de las otras tres; De modo que sólo es necesario especificar eventos y acciones para un objeto: ball1. Los otros heredarán los eventos y actuarán exactamente de la misma forma. Además, cuando se ejecutan acciones a instancias del objeto padre también se aplicarán a los hijos. De forma que si se destruyen todas las instancias del objeto ball1 también se destruirán las instancias de los objetos ball2, ball3, y ball4. Esta opción ahorra mucho trabajo.

Con frecuencia varios objetos deben comportarse de manera casi idéntica pero con algunas diferencias pequeñas. Por ejemplo, un monstruo podría moverse arriba y abajo y el otro hacia la izquierda y la derecha. Para el resto ambos tendrían el mismo comportamiento. En este caso casi todos los eventos deberían tener las mismas acciones excepto uno o dos que tendrían que ser distintos. En este caso podemos hacer a un objeto el padre del otro. Pero también debemos definir algunos eventos para el objeto hijo. Estos eventos se sobrepondrán a los eventos del padre. De forma tal que mientras un evento del objeto hijo contenga acciones, éstas se ejecutarán en lugar de las del mismo evento del

objeto padre. Si además se requiere ejecutar el evento del padre, se puede ' llamar' al ' evento heredado' usando la acción apropiada.

De hecho, es una buena práctica en tales casos crear un objeto base, por ejemplo un objeto ball0. Este objeto contiene todo el comportamiento por defecto pero nunca se utiliza en el juego. Todos los objetos ' reales' tienen a este objeto base como padre.

Los objetos padre pueden así mismo tener padres, y así sucesivamente. (Obviamente no está permitido crear círculos.) De ésta forma se puede crear una jerarquía de objetos. Esto es extremadamente útil oara mantener tu juego estructurado y es ampliamente recomendado aprender a utilizar éste mecanismo.

También hay otra utilidad de los objetos padre. Permite heredar el comportamiento en colisiones para otros objetos. Permítame explicar ésto con un ejemplo. Supongamos que tenemos cuatro objetos ' suelo' distintos. Cuando una pelota golpea el suelo ésta debe cambiar su dirección. Este cambio debe especificarse en el evento colisión de la pelota con el suelo. Debido a que hay 4 distintos suelos necesitamos escribir código en 4 eventos colisión distintos de la pelota. En cambio, cuando se tiene un objeto base y éste es padre de los 4 suelos reales, sólo se debe especificar el evento colisión con éste objeto base. Las demás colisiones ejecutarán el mismo evento. Esto, igualmente ahorra mucho trabajo.

Tal como se indica, dondequiera que se utiliza un objeto, se involucrarán a sus descendientes. Esto sucede cuando, en una acción, se indica que ésta debe aplicarse a instancias de un objeto determinado. Lo mismo sucede cuando se utiliza la sentencia `with()` en código (ver abajo). Y funciona cuando se llaman a funciones como `instance_position`, `instance_number`, etc. Finalmente, sucede cuando se hace referencia a variables en otros objetos. En el ejemplo anterior, cuando se fija la variable `ball1.speed` a 10 este cambio es también aplicado a los objetos ball2, ball3 y ball4.

19.4 Masks (Máscaras)

Cuando dos instancias hacen colisión ocurre un evento colisión. Para decidir en qué circunstancias se intersectan dos instancias, se utilizan los sprites de ambas. Esto es suficiente en la mayoría de los casos, pero en ocaciones se desea basar las colisiones en formas distintas. Por ejemplo, si se está creando un juego isométrico, los objetos tienen típicamente una altura (para darles una apariencia en 3D). Pero para las colisiones sólo se requiere usar la parte que se supone entra en contacto con el suelo de acuerdo al sprite. Este efecto puede lograrse creando un sprite separado que se utiliza como máscara de colisión para el objeto.

19.5 Información

El botón **Show Information (Mostrar Información)** brinda una visión completa de toda la información para el objeto que también puede imprimirse. Esto es útil en particular cuando se pierde una visión integral de todas las acciones y eventos del objeto.

Capítulo 20 Más acerca de las habitaciones

Las habitaciones en *Game Maker* tienen muchas opciones. En **¡Error! No se encuentra el origen de la referencia.** solamente tratamos las más importantes. En este capítulo discutiremos las demás opciones disponibles.

20.1 Configuraciones avanzadas

Hay dos aspectos en la pestaña **settings (configuraciones)** que no hemos discutido aún. La primer opción que aparece es una casilla nombrada **Persistent (persistente)**. Normalmente, cuando se abandona una habitación y posteriormente se regresa al mismo, la configuración de la habitación vuelve a sus valores iniciales. Esto está muy bien si se tienen varios niveles en un juego (N.T. : En juegos como ' Tetris') pero no es lo que normalmente se desea en un RPG, por ejemplo. En este caso, la habitación debe aparecer en el mismo estado en que se dejo la última vez que se visitó. Al seleccionar la casilla **Persistent** se logrará precisamente eso. El estado de las habitaciones será recordado y cuando se vuelva a ellas con posterioridad en el juego, éstas estarán tal y como estaban al salir de ellas. Solamente cuando el juego sea reiniciado, el estado de las habitaciones se reestablecerá. De hecho, hay una excepción a ésto, si algunos objetos están configurados como persistentes (ver **0**), las instancias de este objeto no permanecerán en la habitación sino se moverán a la habitación siguiente.

Después, hay un botón llamado **Creation code (Código de creación)**. Aquí podrás escribir código en GML (ver más adelante) que será ejecutado al momento de crear la habitación. Esto es útil para, por ejemplo, asignar valores a variables, crear algunas instancias, etc. Es muy útil comprender que es exáctamente lo que sucede cuando te mueves a una habitación en particular.

- Primero, en la habitación actual (si existiera) todas las instancias tendrán un evento salir-habitación. Despues, los objetos no persistentes se remueven (¡sin generar un evento destrucción!).
- Posteriormente, se agregan a la nueva habitación las instancias persistentes de la habitación anterior.
- Se crean todas las instancias nuevas y se ejecutan sus eventos creación (en caso de quela habitación no sea persistente o nohaya sido visitada antes en el juego).
- Cuando la habitación es la primera del juego, todas las instancias tendrán un evento inicio-juego.
- Ahora el código de creación de la habitación es ejecutado.
- Finalmente, todas las instancias tendrán un evento inicio-habitación.

De modo que, por ejemplo, los eventos inicio-habitación pueden emplear variables configuradas en el evento creación de la habitación y en el código de creación se puede referir a las instancias en la habitación (tanto las nuevas como las persistentes).

Hay una opción más. Si se configura la preferencia correcta, al dar click con el botón derecho del ratón en una instancias aparecerá un menú contextual. Aquí podrás usar los comandos usuales de suprimir o mover en el orden de profundidad, y también podrás indicar el código de creación de la instancia. Este código se ejecutará cuando la

habitación sea iniciada, justo antes de que el evento creación de la instancia sea ejecutado. Esto es muy útil para, por ejemplo, configurar ciertos parametros que son específicos a la instancia.

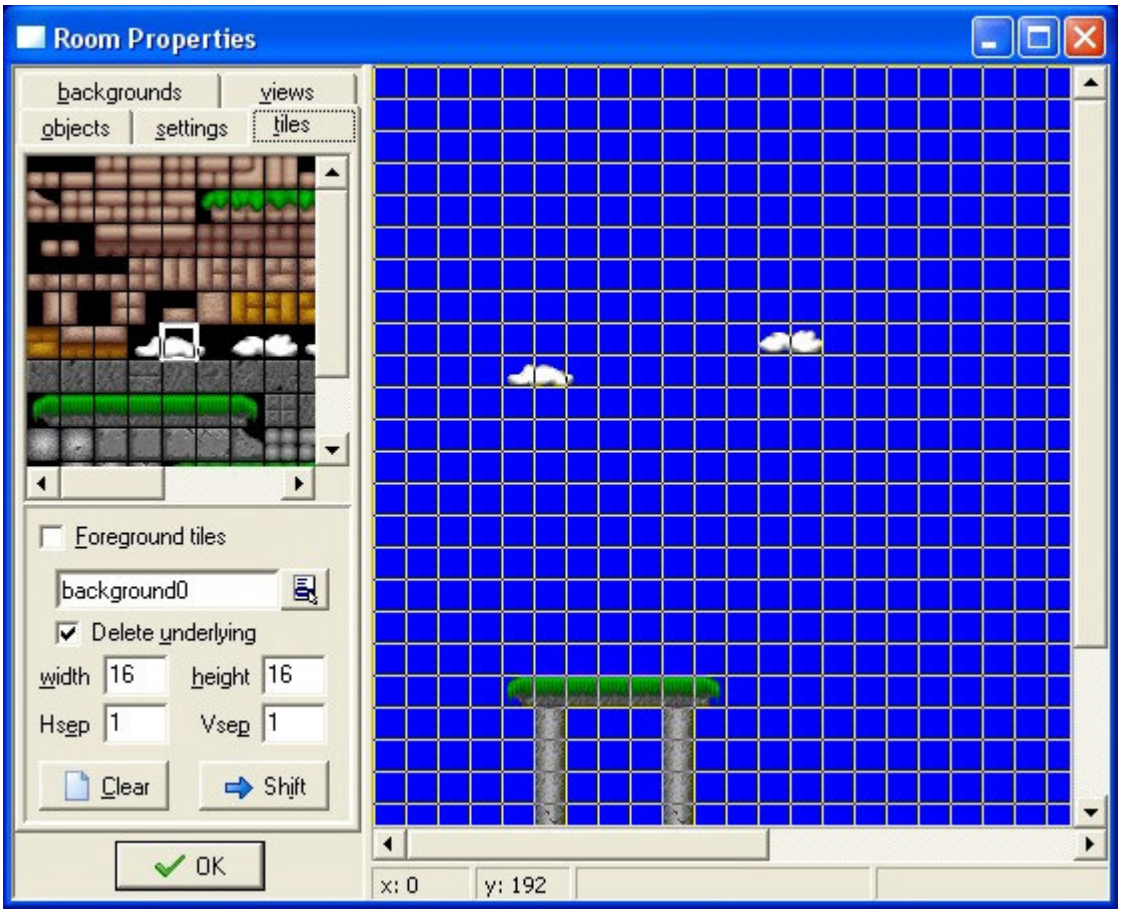
20.2 Agregando mosaicos

También se pueden crear fondos llamados ' de mosaicos' . El objeto de ésto es el siguiente: En muchos juegos querrás tener fondos atractivos. Por ejemplo, en una juego de laberintos, los muros del laberinto deberán encajar, y en un juego de plataforma querrás ver árboles, plataformas, obstáculos, etc. En *Game Maker* puedes lograr esto definiendo muchos objetos diferentes y armar las habitaciones con estos objetos. El problema, sin embargo, es que esto toma mucho trabajo, utiliza grandes cantidades de recursos, y provoca que los juegos se ejecuten lentamente debido a la gran cantidad de objetos distintos utilizados. Por ejemplo, para crear muros con buena apariencia en juegos de laberintos necesitas, por lo menos, 15 objetos con distinta forma.

La solución estándar, utilizada en muchos juegos, es que los muros y otros objetos estáticos son dibujados en el fondo. ¿Pero, te preguntarás, como sabe la computadora que un objeto toca un muro si éste está dibujado en la imagen de fondo? El truco se logra como sigue: Solamente se crea un objeto ' muro' en tu juego. Éste deberá tener el tamaño correcto pero no tendrá que verse bien. Al momento de crear la habitación, Coloca este objeto en todos los lugares en que halla un muro. Y haz a este objeto invisible. De forma que al momento de ejecutar el juego no se verán los objetos ' muro' . Únicamente se verá la imagen de fondo. Pero los objetos ' muro' sólidos aún estarán ahí y lo objetos reaccionarán a ellos.

Puedes emplear esta técnica para cualquier objeto que no cambie su forma ni posición, y no requiera ser animado. Para juegos de plataforma, problemente necesitarás sólo un objeto para el suelo y otro para los muros, pero tus fondos podrán tener parte en que parezca que se camina sobre pasto, sobre ramas de árboles, etc.

Para agregar mosaicos a tu habitación primero necesitarás una imagen de fondo que contenga los mosaicos. Algunas de éstas se proveen con *Game Maker*. Si quieres que tus mosaicos sean parcialmente transparentes, asegúrate que seleccionaste la opción transparente al importar la imagen de fondo. Ahora, cuando estés definiendo la habitación haz click en la etiqueta **tiles (mosaicos)**. Aparecerá una ventana como la siguiente (de hecho, en ésta ya se agregaron algunos mosaicos a la habitación).



En la esquina superior izquierda está el set de mosaicos utilizado. Para seleccionar un ser, haz click en el botón de menú debajo de la parte que muestra la imagen de los mosaicos y selecciona la imagen de fondo apropiada. Debajo de éste botón, podrás cambiar varias opciones. Puedes indicar el largo (width) y alto (height) de cada mosaico y la separación entre los mosaicos (normalmente 0 o 1).

Ahora, podrás añadir mosaicos seleccionando el mosaico deseado en la imagen de la esquina superior izquierda, y después posteriormente haciendo click en la posición apropiada en la habitación a la derecha. Esto funcions en la misma forma exactamente en que se añaden instancias de objetos. Si la opción **Delete underlying (Eliminar mosaico inferior)** está seleccionada, al colocar un mosaico en un lugar donde ya exista uno se reemplazará el existente por el nuevo. Con el botón derecho del ratón se pueden eliminar mosaicos. También hay botones para eliminar **(Clear)** todos los mosaicos y para trasladar **(Shift)** todos los mosaicos en una dirección.

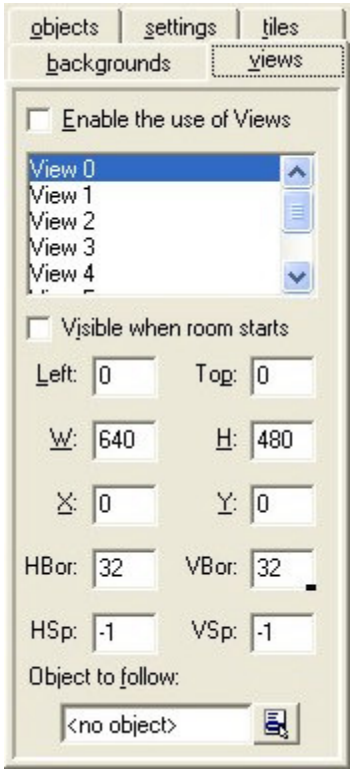
Hay una casilla llamada **Foreground tiles (Mosaicos en primer plano)**. Si seleccionas esta casilla, los mosaicos se dibujarán en frente de los objetos, en lugar de debajo de ellos. Esto se puede utilizar en muchas formas. Cuando se selecciona esta casilla también se eliminan solo los mosaicos en primer plano.

Utilizar mosaicos es una opción muy poderosa que debería ser empleada siempre que sea posible. Es mucho más rápida que utilizar objetos y las imágenes de mosaicos se almacenan sólo una vez. De modo que puedes utilizar habitaciones muy grandes llenas de mosaicos con muy poco consumo de memoria.

20.3 Vistas

Finalmente, hay una solapa llamada **views (vistas)**. Esta opción brinda un mecanismo para dibujar partes diferentes de una habitación en distintos lugares de la pantalla. Existen muchos usos para las vistas. Primero que nada, en distintos juegos querrás mostrar sólo una parte de la habitación en todo momento. Por ejemplo, en la mayoría de los juegos de plataformas, la vista sigue los movimientos del personaje principal. En juegos de dos jugadores, frecuentemente necesitarás dividir la pantalla y mostrar en una parte a uno de los personajes y en otra al personaje que controla el otro jugador. Un tercer uso es común en los juegos en que parte de la habitación deberá desplazarse con el personaje principal y ontra parte de la pantalla está fija (por ejemplo una barra de estado). Esto se puede lograr fácilmente en *Game Maker*.

Cuando hace click en la solapa **views (vistas)** la siguiente información aparecerá:



En la parte superior hay una casilla llamada **Enable the use of Views (Permitir el uso de Vistas)**. Debes seleccionar esta casilla para utilizar las vistas. Debajo verás la lista de las ocho vistas que puedes definir como máximo. Debajo de la lista puedes proporcionar la información de las vista. Primero que nada, debes indicar si la vista debe ser visible cuando la habitación comienza. Asegúrate de que por lo menos una vista es visible. Las vistas visibles se muestran con negritas. Después podrás indicar el área de la habitación que debe mostrarse en la vista. Debes especificar las coordenadas izquierda (left) y superior (top), y el largo (width) y alto (height) de la vista. Debajo, indicas la posición de la vista en la pantalla.

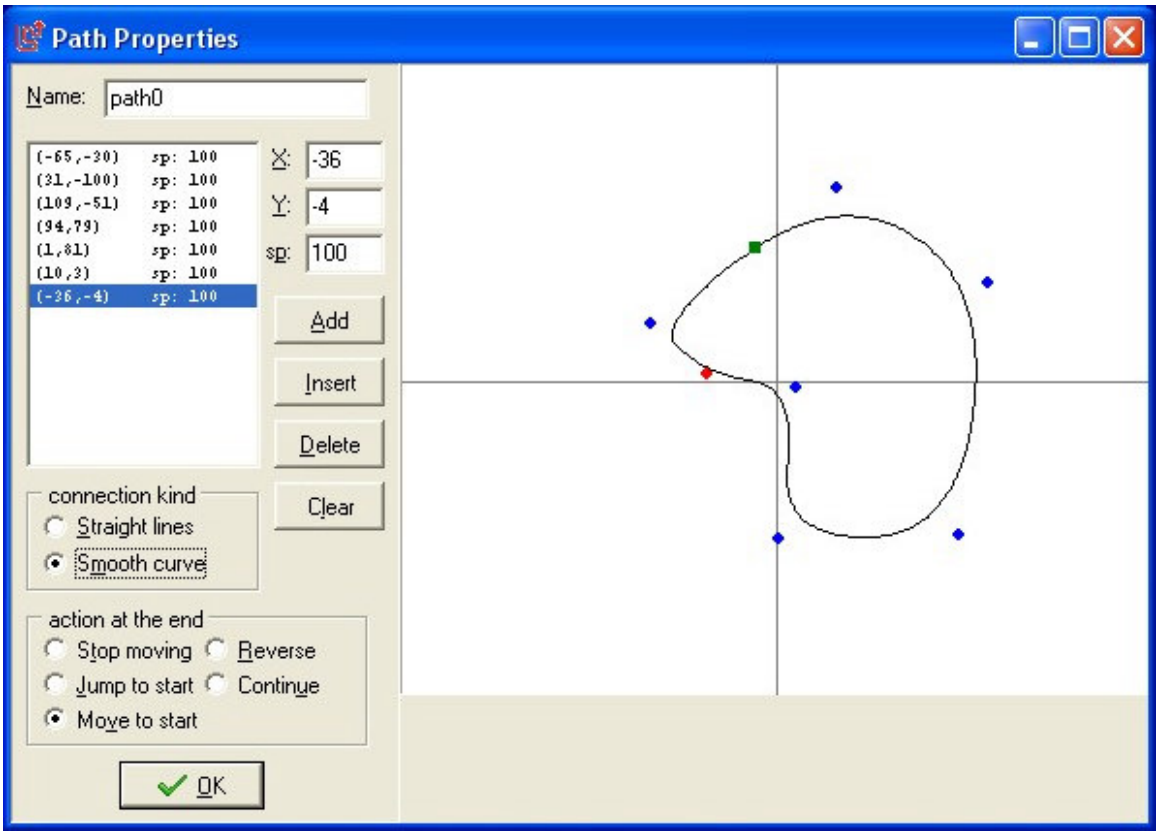
Como se indico anteriormente, frecuentemente se quiere que la vista siga a un determinado objeto. Puedes indicar al objeto a seguir en la última opción disponible. Si hay múltiples instancias de ese objeto, sólo la primera de ellas es seguida por la vista. (Utilizando código también se puede indicar qué instancia seguir.) Normalmente el personaje debe ser capaz de caminar libremente dentro de un determinado espacio sin que la vista cambie. Sólo cuando el personaje se acerca a los límites de la vista, ésta debe cambiar. Puedes especificar el tamaño del borde que debe permanecer visible alrededor del objeto. Finalmente, puedes restringir la velocidad con que la vista cambia. Esto puede significar que el personaje puede caminar fuera de la vista, pero da un movimiento mucho más suave. Usa –1 si deseas que la vista cambie instantáneamente.

Capítulo 21 Paths

En la mayoría de los juegos avanzados regularmente buscas permitir a las instancias seguir ciertos senderos. Incluso puedes indicarlo usando alarmas o código, pero eso es complicado. Los recursos Paths es un mecanismo que lo hace fácil. La idea es bastante simple. Defines un sendero dibujándolo. Enseguida puedes situar una acción en Creation event del objeto para que este siga un sendero en particular. Este capítulo explicará esto a detalle. La implementación actual es bastante limitada. Espera más posibilidades en versiones futuras (compatible con la versión actual).

21.1 Definiendo paths

Para añadir un path a tu juego, escoge **Add Path** en el menú **Add**. Aparecerá la siguiente ventana (en este ejemplo ya añadimos un pequeño path).



Arriba a la izquierda de la ventana puedes especificar el nombre del path. Debajo encontrarás los puntos que influyen el path. Cada punto tiene una posición y velocidad (indicado con sp). La posición no es absoluta. Como se indica debajo, la instancia siempre empieza en la primera posición del path, y lo sigue a partir de ahí. La velocidad es interpretada de la siguiente manera. Un valor de 100 significa la velocidad original de la instancia. Un valor menor reduce la velocidad, un valor más alto la incrementa (esto indica el porcentaje de la velocidad real). La velocidad es intercalada entre puntos, por lo que la velocidad cambia gradualmente.

Para añadir un punto presiona el botón **Add**. Ahora puedes indicar la posición actual y la velocidad. En cualquier momento en que selecciones un punto en la lista, puedes cambiar sus valores. Presiona **Insert** para insertar un nuevo punto después del actual, y **Delete** para borrar el punto actual. Finalmente, puedes usar **Clear** para borrar el path completamente.

A la derecha de la ventana verás el path actual. También puedes cambiar el path usando el ratón. Da un clic sobre la imagen para agregar un punto. Da clic en un punto existente y arrástralo para cambiar su posición. Cuando mantienes <Shift> mientras das un clic, insertas un punto. Finalmente, puedes usar el botón derecho para borrar puntos. (Nota que de esta manera no puedes cambiar la velocidad.)

Puedes influir la forma del path de dos maneras. Primero puedes usar un tipo de unión. Puedes escoger líneas rectas o curvas. En segundo lugar, puedes indicar que pasa cuando se llega al último punto. Existen varias opciones. La más común es continuar moviéndose hacia el primer punto, cerrando el path (move to start). También puede dejar de moverse (stop moving), brincar al primer punto (jump to start), o regresarse por el mismo path (reverse). La opción final (continue) reinicia el path desde la posición actual. De esta manera el path va a "avanzar". Sólo se muestran las primeras cinco repeticiones, pero el path continúa después de esto.

21.2 Asignando paths a los objetos

Para asignar un path a una instancia de un objeto, puedes asignar la acción path en algunos eventos, por ejemplo en el evento "create". En esta acción debes especificar el path de un menú desplegable. También hay otros dos valores que debes proporcionar. Primero la velocidad a la que el path será ejecutado (speed). Recuerda que cuando defines el path tu especificas la velocidad real relativo a la velocidad indicada. Después puedes indicar en dónde debe iniciar el path. Un valor de 0 indica el inicio (que es lo más común). Un valor de 1 indica el final del path, esto es, que en el momento en que el path es ejecutado es la segunda vez. Por ejemplo cuando el cuando el path es inverso (reverse), un valor de 0.5, es el instante en que se regresa.

Cuando usas scrips o piezas de código tienes más control sobre la manera en que el path es ejecutado. Existen 4 variables que lo influencian. La variable path_index indica el inicio del path. La variable path_position indica la posición actual de le path. (Entre 0 y 1 como se indicó anteriormente). Esto cambia mientras la instancia siga el path. La velociadad es controlada por la variable speed en path properties. Nota que la variable direction está fijada automáticamente en cada paso hacia la dirección correcta a lo largo del path. Entonces puedes usar esta variable para por ejemplo escoger la subimagen correcta. La variable path_scale puede ser usada para escalar el path. Un valor de 1 es la talla real. Un valor más grande indica que el path será más grande, uno menor lo disminuye. La variable path_orientation indica la orientación en que el path es ejecutado (en grados contrario a las manecillas del reloj). Esto permite que puedas ejecutar paths en diferentes orientaciones (Ej. Moviéndose arriba y abajo en vez de izquierda y derecha.)

Tal vez te preguntes que pasa cuando la instancia coaliciona con otra instancia mientras este sigue un path. Primero el evento coalición es ejecutado. Si la otra instancia es sólida la instancia se detendrá, como debería (Asumiendo que existe un evento de coalición definido.) Sin embargo la variable path_position continuará siguiendo el path. En algún momento la instancia podría moverse otra vez en una dirección diferente, si una posición semejante es alcanzada en el path.

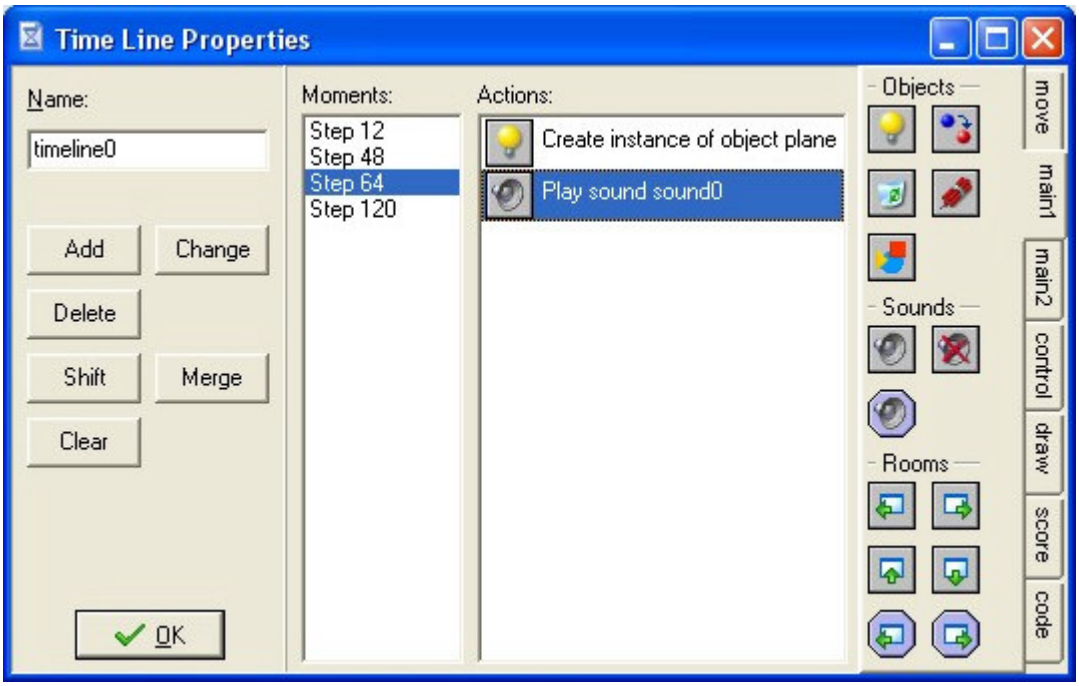
21.3 El evento path

Como se describió antes, puedes indicar que sucede cuando la instancia llega al fin del path. En ente momento ocurre el evento **End of path**. Puedes encontrarlo en el evento **Other**. Aquí puedes añadir acciones. Por ejemplo, tal vez quieras destruir la instancia, o que empiece un nuevo path (diferente).

Capítulo 22 Time Lines

En muchos juegos ciertas cosas deben pasar en ciertos lapsos de tiempo. Puedes intentas hacer esto usando el evento alarm pero cuando las cosas se vuelven muy complicadas esto no podría funcionar. Mediante el recurso 'time line" lo puedes hacer. En una time line tu especificas que acciones deben suceder en ciertos lapsos de tiempo. Puedes utilizar todas las acciones que son disponibles para los diferentes eventos. Ya que hayas creado una time line puedes asignarla a una instancia de un objeto. Esta instancia ejecutará las acciones en los lapsos de tiempo indicados. Permíteme explicarlo con un ejemplo. Supongamos que quieres crear un guardia. Este guardia debe moverse 20 pasos a la izquierda, después 10 arriba, 20 a la derecha, 10 hacia abajo y después detenerse. Para lograr esto creas una time line donde empiezas con un movimiento a la izquierda. En 20 pasos pones un movimiento hacia arriba. En el paso 30 un movimiento a la derecha, en el paso 50 un movimiento hacia abajo y en el 60 detienes el movimiento. Ahora puedes asignar esta línea de tiempo al guardia y el guardia hará exactamente lo que planeaste. También puedes usar una línea de tiempo para controlar tu juego de manera global. Crea un objeto invisible, crea una time line que en ciertos momentos cree enemigos, y asígnala a el objeto. Si empiezas a trabajar con esto encontrarás que es un concepto muy poderoso.

Para crear una time line, selecciona **Add time line** del menú **Add**. Aparecerá la siguiente ventana:



Se parece un poco a la ventana de propiedades del objeto. A la izquierda puedes poner el nombre y además hay botones para añadir y modificar "moments" en la time line. Enseguida está la lista de los "moments". Esta lista especifica los lapsos de tiempo en steps en que las acciones asignadas van a pasar. Y enseguida está la lista de acciones para

el moment seleccionado y finalmente a la derecha está la lista total de acciones disponibles.

Para añadir un moment presiona el botón **Add**. Indicando un lapso de tiempo (esto es el número de pasos desde que la time line comienza). Ahora puedes arrastrar acciones a la lista de manera semejante como los eventos de los objetos. También hay botones para borrar el moment seleccionado y para borrar toda la time line.

Finalmente existen dos botones especiales. Con el botón **Merge** puedes unir varios moments en un rango deseado en uno solo. Con el botón **Shift** puedes cambiar todos los moments cierta cantidad de tiempo avanzándolos o retrocediéndolos la cantidad deseada. Asegúrate de que no creaste moments con un step negativo, porque nunca se ejecutarán.

Existen dos acciones relativas a las time lines:

 **Set a time line**
Con esta acción asignas una time line en particular a una instancia de un objeto. Aquí indicas la time line y la posición de inicio de la time line (0 es al principio). También puedes usar esta acción para finalizar una time line escogiendo No Time Line como valor.

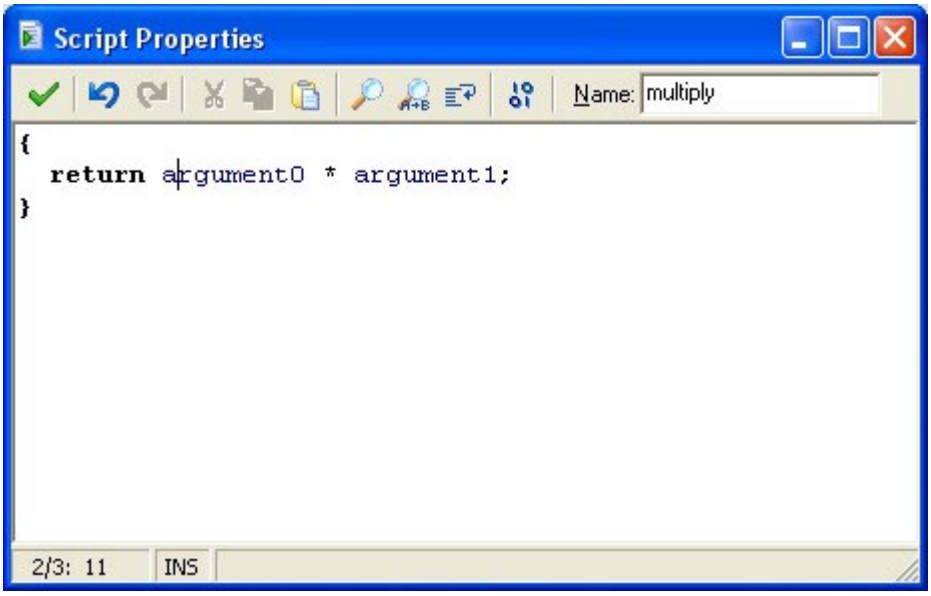
 **Set the time line position**
Con esta acción puedes cambiar la posición de la time line actual (puede ser absolute o relative). Esto puede ser usado para saltar ciertas partes de la time line o para repetir ciertas partes. Por ejemplo, si quieres crear una time line que se repita una y otra vez, añade esta acción en el último moment para que la posición regrese a 0. También puedes usarlo para esperar a que pase algo. Sólo añade la acción de prueba (usa un if) y, si no es cierta, asignas la posición de la time line un valor relative de -1.

Capítulo 23 Scripts

Game Maker posee un lenguaje de programación interno. Una vez que te familiarices con Game Maker y quieras usarlo a sus máximos alcances, es prudente aprender a usar este lenguaje. Ve el capítulo 28 para una descripción más completa. Existen dos maneras para usar el lenguaje. Primero puedes crear scripts. Estos son piezas de código a los que les darás un nombre. Son mostrados en el árbol de recursos y pueden ser salvados a un archivo y cargarlos desde un archivo. Pueden ser usados para formar una librería que extienda las posibilidades de Game Maker. Alternamente, puedes añadir una acción de código a algún evento y escribir una pieza de código ahí. Añadir el código funciona exactamente de la misma manera que añadir scripts excepto por dos cosas. Las piezas de código no tienen un nombre y no usan argumentos. Además ya tienen indicados a que objeto deben aplicar el código. Por lo demás escribes el código de la misma manera como en los scripts. En este capítulo nos vamos a concentrar a fondo en los scripts.

Como se señaló antes, un script es una pieza de código en el lenguaje de programación interno que realiza una tarea específica. Un script puede usar un número de argumentos. Para ejecutar un código desde algún evento, puedes usar la acción script. En esta opción especificas el script que desees ejecutar, junto con un máximo de cinco argumentos. (También puedes ejecutar scripts desde una pieza de código de la misma manera en que llamas una función. En este caso puedes usar 16 argumentos como máximo.) Cuando un script retorna un valor, puedes usarlo también como una función cuando provee valores en otras acciones.

Para añadir un script a tu juego, selecciona **Add script** de el menú **Add**. Aparecerá la siguiente ventana (en este ejemplo ya añadimos un pequeño script que calcula el producto de dos argumentos).



(Realmente, este es el editor de scripts interno. En las preferencias puedes indicar si deseas usar un editor externo.) Arriba a la derecha puedes indicar el nombre del script. Tienes un pequeño editor donde puedes escribir el script. El editor tiene varias propiedades útiles la mayoría mediante botones (presiona el botón derecho del ratón para algunos comandos adicionales):

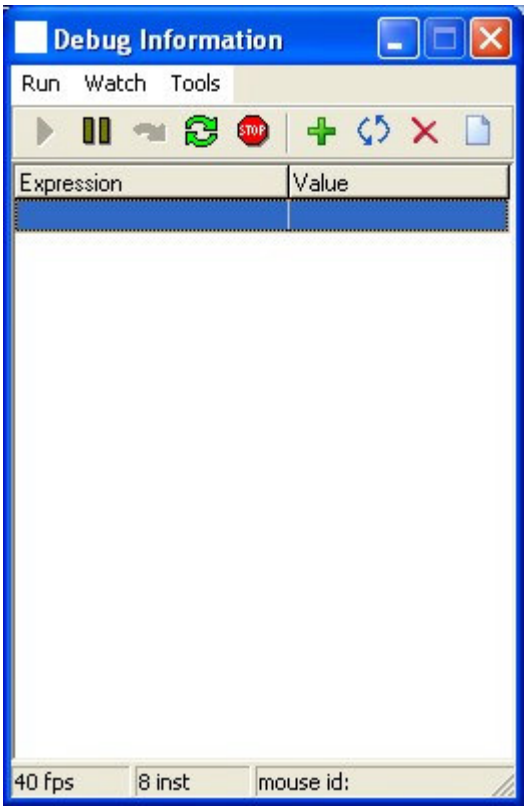
- Múltiple deshacer y rehacer ya sea por tecla presionada o en grupos (puede ser cambiado en las preferencias)
- Sangría automática inteligente que se alinea con la línea previa (puede ser habilitado en las preferencias)
- Tabulador inteligente que hasta el primer carácter (no espacio) en la línea previa (puede ser habilitado en las preferencias)
- Usa <Ctrl.+I> para aumentar la sangría en las líneas seleccionadas y <Shift><Ctrl>+I para reducirla en las líneas seleccionadas.
- Cortar y pegar
- Buscar y reemplazar
- Usa <Ctrl> + arriba, abajo, re-pag o av-pag para desplazarte sin cambiar la posición del cursor.
- Usa F4 para abrir el script o el recurso cuyo nombre esté en la posición del cursor (no funciona en la acción de código; sólo en los scripts)
- Guardar y Cargar el script como un archivo de texto.

También hay un botón con el cual puedes probar si el script está correcto. No todos los aspectos son probados, solo la sintaxis de tu script, junto con la existencia de alguna función usada.

Como habrás notado, partes del texto del script tienen diferentes colores. El editor reconoce la existencia de objetos, variables internas y funciones, etc. El color del código ayuda bastante a evitar errores. En particular, notas inmediatamente si escribiste mal algún nombre o si usaste una keyword (variable con una función definida como var, global, other, self, etc.) como variable. Sin embargo colorear el código es algo lento. En las preferencias del menú file puedes activarlo o desactivarlo. Además puedes cambiar el color para los diferentes componentes de los programas. (Si algo sale mal con el coloreado del código, presiona F12 dos veces, para desactivarlo y volver a activarlo.) También puedes cambiar la fuente usada en los scripts y en las piezas de código.

Los scripts son extremadamente útiles para extender las posibilidades del *Game Maker*. Aunque esto requiere que diseñes tus scripts con cuidado. Los scripts pueden ser guardados en librerías que pueden ser añadidos a tu juego. Para importar una librería, usa el item **Import scripts** de el menú file. Para guardar tu scripts como una librería usa **Export scripts**. Las librerías de scripts son simples archivos de texto (Aunque tienen la extención .gml). Preferentemente no los edites directamente porque poseen una estructura especial. Algunas librerías con scripts útiles son incluidas. (Para evitar trabajo innecesario cuando cargas el juego, después de importar una librería, es mejor borrar las librerías que no utilices.)

Cuando se crean scripts es muy fácil cometer errores. Siempre prueba los scripts usando el botón apropiado. Cuando ocurre un error durante la ejecución del script este es reportado, junto con una indicación del tipo de error y el lugar. Si necesitas verificar las cosas más cuidadosamente, puedes correr el juego en debug mode (F6). En este caso aparece una ventana en la cual puedes monitorear mucha de la información de tu juego.

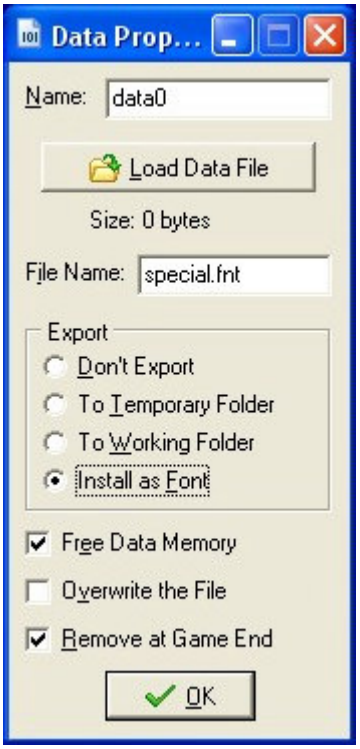


En el menú **Run** puedes pausar el juego, correrlo paso a paso e incluso reiniciarlo. En el menú **Watch** puedes ver el valor de ciertas expresiones. Usa **Add** para escribir algunas expresiones cuyos valores se mostrarán en cada paso del juego. De esta manera puedes ver si el juego está haciendo las cosas de la manera adecuada. Puedes ver varias expresiones. Puedes salvarlos para usarlos después. (Ej. Para hacer correcciones al juego). En el menú **Tools** se encuentran comandos para ver más información. Puedes ver una lista de todas las instancias en el juego, puedes ver todas las variables globales (usa el nombre del objeto o el id de la instancia.) Además puedes ver mensajes los cuales puedes enviar desde tu código usando la función show_debug_message(str). Finalmente puedes dar al juego comandos y cambiar la velocidad del juego. Si haces juegos complicados deberías aprender a usar las opciones del debug mode.

Capítulo 24 Data files

En la mayoría de los juegos avanzados necesitas a menudo usar archivos adicionales, por ejemplo archivos que describan ciertas propiedades, backgrounds, videos, archivos dll (explicados más adelante) o tus propias fuentes. Puedes distribuir estos archivos con tu juego, pero es más agradable agregarlos dentro del juego. Para esto puedes usar el recurso data file. Los data file simplemente guardan el contenido de un archivo. Cuando comienza el juego este archivo es guardado en el disco y pueden ser usados en el juego.

Para añadir un data file selecciona **Add data file** de el menú **Add**. Aparecerá la siguiente ventana:



Como siempre puedes darle un nombre. Presiona el botón **Load data file** para cargar el archivo dentro del recurso. Puedes escoger **File name** (nombre del archivo) el cual será usado para guardar el recurso (no puedes usar un ruta en el file name). A continuación puedes indicar que se hará con el data file cuando comience el juego:

- **Don't Export**. No lo exporta. (Hay una función para exportarlo más tarde.)
- **To Temporary Folder**. El archivo es guardado en el directorio temporal para el juego. Esto es muy útil para por ejemplo cargar backgrounds, etc.
- **To Working Folder**. El archivo es guardado en el fólder donde el juego es ejecutado. Muchas rutinas buscan por archivos ahí, pero debes tener cuidado pues el jugador podría correr el juego desde una unidad de sólo lectura.
- **Install as Font**. Si tu archivo es un archivo de fuente puedes seleccionar esta opción. El archivo es guardado en la carpeta temporal y enseguida instalado como una fuente que puede ser usada en el juego.

Finalmente hay unas cuantas opciones:

- **Free Data Memory**. Si lo habilitas la memoria usada para el data file es liberada después de exportarla. Esto salva memoria pero significa que el recurso no puede ser usado en funciones.
- **Overwrite the File**. Si lo habilitas el archivo es sobrescribido si ya existe.
- **Remove at Game End**. Si está habilitado el archivo es removido al finalizar el juego. (Nota que los archivos en el directorio temporal siempre son removidos al final del juego.)

Si ocurre un error durante la exportación el juego seguirá corriendo. Pero el archivo o fuente no estará disponible.

Los Data Files pueden usar mucha memoria, pero cargarlos y exportarlos en muy rápido.

Capítulo 25 Información del juego

Un buen juego provee al jugador con cierta información sobre como jugar el juego. Esta información es desplegada cuando el jugador presiona la tecla <F1> durante el juego. Para crear la información del juego, da doble clic en **Game Information** en el árbol de recursos a la izquierda de la pantalla. Se abre un pequeño editor de texto en donde puedes editar la información del juego. Puedes usar diferentes fuentes, diferentes colores, y estilos. También puedes poner un color de fondo.

Una opción interesante en el menú **Format** es **Mimic Main Form.** Cuando habilitas esta opción la ventana de la ayuda se muestra exactamente en la misma posición y tamaño que el juego. Esto da como resultado que se vea como si el texto apareciera en la ventana del juego. Escogiendo el color de fondo apropiado provee un efecto visual agradable. (Podrías indicar en el fondo de la ayuda que el jugador debe presionar Escape para continuar jugando.)

Un buen consejo es hacer la información corta pero precisa. Por supuesto puedes agregar tu nombre porque tú creaste el juego. Todos los juegos de ejemplo proveen un archivo de información acerca del juego y de cómo fue creado.

Si quieres hacer una ayuda más imaginativa, usa por ejemplo Word. Y entonces seleccionas la parte que desees y usa copiar y pegar para moverlo desde Word al editor de **Game Information**.

A good game provides the player with some information on how to play the game. This information is displayed when the player presses the <F1> key during game play. To create the game information, double click **Game Information** in the resource tree at the left of the screen. A little build-in editor is opened where you can edit the game information. You can use different fonts, different colors, and styles. Also you can set the background color.

One interesting option in the **Format** menu is to **Mimic Main Form**. When you check this option the help form is displayed exactly at the position and the size of the game form. As a result it looks like the text appears in the game window. Choosing the correct background color now provides a nice visual effect. (You might want to indicate at the bottom of the help file that the user must press Escape to continue playing.)

A good advice is to make the information short but precise. Of course you should add your name because you created the game. All example games provided have an information file about the game and how it was created.

If you want to make a bit more fancy help, use e.g. Word. Then select the part you want and use copy and paste to move it from Word to the game information editor.

Capítulo 26 Opciones del juego

Existen un número de opciones que puedes cambiar para tu juego. Pueden ser encontradas haciendo doble clic en **Game Options** en el árbol de recursos a la izquierda de la pantalla. Estas están subdivididas en varias pestañas.
There are a number of options you can change for your game. They can be found by double clicking on **Game Options** in the resource tree at the left of the screen. They are subdivided in a number of tabbed pages.

26.1 Opciones para los gráficos

En esta pestaña puedes poner un número de opciones que están relacionadas con la apariencia gráfica del juego. Normalmente es útil comprobar los efectos de esas opciones porque pueden tener un efecto significativo en la forma en que el juego luce. Recuerda que diferentes usuarios tienen máquinas diferentes. Entonces asegúrate que las opciones funcionen en las máquinas de otra gente.

Start in fullscreen mode
Cuando lo habilitas el juego corre en pantalla completa; de lo contrario corre en una ventana.

Scale percentage in windowed mode
Aquí puedes indicar que la imagen en el modo ventana debe ser escalada. 100 es no hacer una escala. Típicamente lo usas cuando tus sorites y cuantos son muy pequeños. Crear escalas es muy lento pero la mayoría de las tarjetas de video actuales pueden hacerlo sin problemas. Es mejor no usar valores menores de 100 pues hacer escalas para reducir en tamaño normalmente es muy lento.

Scale percentage in fullscreen mode
Aquí indicas que la imagen en pantalla completa debe ser escalada. 100 es sin escala. Un valor de 0 indica que la escala debe ser la máxima posible. Usar escalas es muy lento pero la mayoría de las tarjetas de video actuales pueden hacerlo sin problemas. Es mejor no usar valores menores de 100 pues hacer escalas para reducir en tamaño normalmente es muy lento.

Only scale when there is hardware support
Si lo habilitas solo se usan las escalas si el hardware lo soporta. Desafortunadamente aunque algunas tarjetas de video indican que el hardware soporta escalas, aunque en realidad no lo puedan hacer.

Don’t draw a border in windowed mode
Si lo habilitas en el modo ventana no tendrá un borde ni la barra de título.

Don’t show the buttons in the window caption
Cuando está habilitado, en el modo ventana va a mostrar los botones para cerrar o para minimizar.

Wait for vertical blank before drawing
La pantalla de tu computadora se refresca cierto número de veces por segundo (normalmente entre 50 y 100). Después de refrescar la pantalla hay algo llamado "vacío vertical" donde nada pasa en la pantalla. Si dibujas la pantalla continuamente, parte de una imagen y parte de la siguiente podría verse en la pantalla, lo cual le da un efecto visual pobre. Si esperas por el vacío vertical antes de dibujar el siguiente cuadro, este problema desaparece. La desventaja es que el programa debe esperar por el vacío vertical lo que puede hacer un poco más lento el juego.

Display the cursor
Aquí indicas si quieres que el puntero del ratón vaya a ser visible. Desactivarlo es normalmente más rápido y agradable. (Puedes hacer fácilmente tu propio cursor con el Game Maker.)

Display the caption in fullscreen mode
Si lo habilitas, en el modo pantalla completa una pequeña caja blanca es dibujada arriba a la izquierda, mostrando el "caption" del cuarto, el "score" y el "number of lives". Puedes deshabilitarlo aquí. Es más agradable si tú dibujas estas cosas en un lugar apropiado en tus cuartos.

Freeze the game when the form looses focus
Cuando lo habilitas, si el jugador pone alguna otra ventana sobre el juego (por ejemplo otro programa) el juego se congela hasta que la ventana del juego es enfocada otra vez.

26.2 Resolución
En esta pestaña puedes indicar la resolución en que correrá el juego.

Set the resolution of the screen
La pantalla tiene una resolución particular. Tres aspectos juegan un papel aquí: el número de píxeles de la pantalla (horizontales y verticales), el número de bits usados para representar los colores, y la frecuencia en que la pantalla es actualizada. Normalmente Game Maker no cambia estas opciones, esto es, que usa la configuración de la máquina donde el juego se ejecuta. Esto nos lleva a gráficos pobre. Por ejemplo, si tus cuartos son pequeños y un usuario tiene una resolución más grande, el juego se va a mostrar en una pantalla muy pequeña. Puedes seleccionar esto usando "full screen mode" (pantalla completa) y escalando la imagen, pero esto puede hacer lento el juego. La mejor manera de resolver esto es decir al Game Maker que cambie la resolución de la pantalla cuando corra el juego. Lo cambiará de regreso después del juego. Para hacerlo habilita esta opción. Un número de opciones adicionales aparecerán. Primero puedes indicar la intensidad del color (16 o 32 bits; 16 bits es mejor; en Windows '98 los juegos siempre corren con una paleta de colores de 16 bits, incluso si especificas 32 bits). Después puedes indicar el tamaño de la pantalla (320x240, 640x480, 800x600, 1024x768, 1280x1024, o 1600x1200). También aquí hay algunas advertencias. Si usas una resolución pequeña (Ej. 320x240) Windows cambiará el tamaño de todas la ventanas cuando corra el juego. Esto puede causar problemas con otras aplicaciones que están corriendo (incluyendo Game Maker). (Usa "exclusive Mode" para evitarlo.) Cuando usas

las dos más grandes tienes que tener cuidado porque no todas las computadoras lo soportan. También puedes indicar que no cambie la resolución de la pantalla. Finalmente puedes indicar la frecuencia (60, 70, 85, 100; si el que especificas es demasiado alto, la frecuencia predeterminada es usada; también puedes especificar que usa la frecuencia predeterminada). También la siguiente opción aparece:

Use exclusive graphics mode

En modo exclusivo, el juego tiene el control total sobe la pantalla. Ninguna otra aplicación la puede usar más. Esto hace los gráficos un poco más rápidos y permite algunos efectos especiales (como la configuración gamma). Si quieres asegurarte que la computadora del jugador esté y se mantenga en la resolución de pantalla correcta, mejor usa "exclusive mode" (modo exclusivo). Además no se pueden mostrar advertencias. En modo exclusivo no se puede mostrar ninguna otra ventana. Esto significa que no puedes usar acciones que muestren un mensaje, hagan una pregunta, muestre los puntajes más altos, o la información del juego. Además ningún error puede ser reportado. En general, cuando algo falla en modo exclusivo el juego termina, y a veces esto no ayuda y el jugador no tiene otra opción que reiniciar el juego. Debes asegurarte que tu juego funcione correctamente. No puedes correr el juego en "debug mode" cuando se usa el modo exclusivo.

26.3 Opciones de teclas

Let <Esc> end the game

Cuando lo habilitas, al presionar la tecla Escape termina el juego. Los juegos más avanzados normalmente no quieren que esto pase porque talvez quieran hacer algunos procesos (como salvar) antes de terminar el juego. En este caso, deshabilita esta opción y provee tu propia acción para la tecla escape. (Dar un clic sobre la cruz de la ventana también genera un evento de la tecla escape.)

Let <F1> show the game information

Cuando está habilitado, presionando la tecla F1 muestra la información del juego (pero no en el modo exclusivo).

Let <F4> switch between screen modes

Si lo habilitas, la tecla F4 cambiará entre pantalla completa y el modo ventana (no en el modo exclusivo.)

Let <F5> and <F6> load and save the game

Cuando está habilitado, el jugador puede usar F5 para guardar la situación actual del juego, y F6 para cargar el último juego guardado.

26.4 Opciones de carga

Aquí puedes indicar lo que debe suceder cuando cargas el juego. Primero puedes especificar una imagen propia de cargado. Después puedes indicar si se mostrará una barra del proceso de cargado en la parte baja de la imagen. Tienes tres opciones. Que no se muestre una barra de progreso, que se muestre la barra que está por defecto o puedes especificar dos imágenes: El fondo de la barra de cargado (back image) y la imagen en

primer plano (front image). Estas se ajustarán para obtener el tamaño adecuado. (En este caso debes de especificar ambas imágenes, no sólo una.)

Después, puedes indicar un icono que será usado en el juego ejecutable. Sólo puedes usar un icono de 32x32 pixeles. Si intentas seleccionar otro tipo de icono obtendrás una advertencia.

Finalmente puedes cambiar el id único del juego. Este id se usa para registrar los puntajes más altos y los juegos salvados. Si tú liberas una nueva versión de tu juego y no quieres usar los puntajes anteriores, debes cambiar este número.

26.5 Opciones de errores

Aquí especificas algunas opciones relativas a la manera en que los errores son reportados.

Display error messages
Si lo habilitas, los mensajes de error se muestran al jugador (excepto en "exclusive mode.") En la versión final de tu juego deberías deshabilitar esta opción.

Write error messages to file game_errors.log
Si lo seleccionas, todos los mensajes de errores son escritos en un archivo llamado game_errors.log en la carpeta del juego.

Abort on all error messages
Normalmente ciertos errores son fatales, mientras otros pueden ser ignorados. Cuando habilitas esta opción todos los errores son considerados fatales y el juego es terminado. En la versión final del juego deberías habilitar esta opción.

Treat uninitialized variables as 0
Un error común es utilizar una variable antes de que tenga un valor asignado. Algunas veces esto es difícil de evitar. Cuando habilitas esta opción, las variables sin inicializar no reportarán un error pero se les asignará el valor de 0. Aunque debes tener cuidado. Esto significará que ya no serás informado de errores de escritura.

26.6 Opciones de información

Aquí indicas el nombre del autor del juego. La versión del juego, y alguna información acerca del juego. Además se mantiene la fecha de la última modificación. Esto es útil si estás trabajando con varias personas en un juego o haciendo una nueva versión. Esta información no es accesible mientras se ejecute el juego.

Capítulo 27 Consideraciones sobre la velocidad

Si estás haciendo juegos complicados probablemente quieras hacer que corran lo más rápido posible. Aparte de que Game Maker hace que el juego corra lo más rápido posible, mucho depende en como diseñes el juego. Además es muy fácil crear juegos que utilicen mucha memoria. En este capítulo te daré algunas sugerencias sobre cómo hacer tus juegos más rápidos y de tamaño pequeño.

Antes de todo, mira cuidadosamente a los sprites y backgroundos que utilizas. Los sprites animados necesitan mucha memoria y dibujar muchos sprites toma mucho tiempo. Debes hacer tus sprites lo más pequeños que puedas. Decide cuidadosamente cuales sprites serán almacenados en la memoria de video y cuales cargar sólo en uso. Lo mismo sucede con los backgrounds. En general deben cargarse solo en uso y, en particular cuando las imágenes son muy grandes, no debes de almacenarlas en la memoria de video. Si tienes un background que cubra el fondo, asegúrate de deshabilitar el uso de un color de fondo (background color) en las opciones de tus cuartos.

Si utilizas el modo pantalla completa o el modo exclusivo, asegúrate de que el tamaño del cuarto (o ventana) no sea mayor que el tamaño de la pantalla. La mayoría de las tarjetas de video pueden eficientemente aumentar a escala las imágenes, pero reducir a escala es lento. Además, dibuja la menor cantidad de cosas que no sean sprites. Esto es lento. Si las necesitas, prefiere dibujar una después de otra. Finalmente, si es posible, desactiva el cursor. Esto hace lentos los gráficos.

También ten cuidado de no utilizar muchas vistas. Por cada vista el cuarto se vuelve a dibujar.

Aparte de los gráficos, otros aspectos influyen a la velocidad. Asegúrate de tener la menor cantidad de instancias (objetos) como sea posible. En particular, destruye los objetos que ya no sean requeridos (Ej. cuando abandonan el cuarto). Evita mucho código en el evento step o en el evento drawing de los objetos. Frecuentemente las cosas no necesitan ser verificadas en cada paso. La interpretación del código es razonablemente rápida, pero esto es interpretado. Además, algunas funciones y acciones toman mucho tiempo; en particular cuando tiene que verificar todos los objetos (como las acciones de colisión).

Piensa acerca de donde utilizar los eventos de colisión. Normalmente tienes dos opciones. Los objetos que no tienen eventos de colisión son tratados mucho más rápidos, entonces utilízalos en los objetos que tengan pocas instancias.

Ten cuidado con el uso de archivos de sonido muy grandes. Necesitan mucha memoria y además se comprimen muy mal. Deberías verificar tus sonidos y ver si puedes detenerlos.

Finalmente, si quieres crear un juego que muchas personas puedan jugar, asegúrate de probar el juego en máquinas viejas.

Capítulo 28 El Lenguaje Game Maker (GML)

Como habrás leído antes, el *Game Maker* contiene un lenguaje de programación interno. Este lenguaje te da mucha más flexibilidad y control que las acciones estándar. Nos referiremos a este lenguaje como el GML (de Game Maker Language). Hay tres diferentes lugares en los que puedes escribir programas con este lenguaje. El primero, cuando defines scripts. Un script es un programa en GML. Segundo, cuando agregas una acción de código a un evento. En una acción de código debes escribir un programa en GML. Finalmente, en cualquier momento que necesites especificar algún valor en una acción, puedes también emplear una expresión en GML. Una expresión, como veremos más adelante no es un programa completo, sino una pieza de código que devuelve un resultado.

En este capítulo describiré la estructura básica de los programas en GML. Cuando desees usar programas en GML, se debe tener cuidado con ciertos aspectos. Primero que nada, para todos tus recursos (sprites, objetos, sonidos, etc.) debes emplear nombres que inicien con una letra y que sólo consistan de letras, números y el guión bajo '_'. De otra forma no podrás referirte a ellas desde el programa. También ten cuidado de no nombrar a tus recursos self, other, global o all porque estas son palabras que tienen un significado especial dentro del lenguaje. Tampoco debes usar ninguna de las palabras reservadas indicadas a continuación.

28.1 Un programa

Un programa consiste en un bloque. Un bloque consiste de una o más sentencias, delimitadas por los símbolos '{' y '}'. Las sentencias deben separarse con el símbolo ';'. Así, la estructura global de todo programa es:

```
{
  <sentencia>;
  <sentencia>;
  …
}
```

Una sentencia puede ser también un bloque de ellas. Hay varios tipos de sentencias, de las cuales hablaremos a continuación.

28.2 Variables

Como en cualquier lenguaje de programación, el GML contiene variables. Las variables pueden almacenar valores reales o cadenas. Las variables no necesitan ser declaradas. Hay un gran número de variables internas. Algunas son generales, como `mouse_x` y `mouse_y`, las cuales indican la posición actual del cursor, mientras otras son locales para la instancia del objeto para el cual se ejecuta el programa, como `x` y `y` que indican la posición actual de la instancia. Una variable tiene un nombre que debe iniciar con una letra, y puede contener sólo letras, números, y el símbolo '_'. (La longitud máxima es de 64 caracteres). Cuando haces uso de una nueva variable, ésta es local para la instancia actual y no es conocida en los programas de otras instancias (aún del mismo objeto).

aunque existe la posibilidad de hacer referencia a variables de otras instancias; ve a continuación.

28.3 Asignación

Una asignación pasa el valor de una expresión a una variable. Una asignación tiene la siguiente forma:

```
<variable> = <expresión>;
```

Además de asignar un valor a una variable, también se le puede sumar usando +=, restar usando -=, multiplicarla usando *=, o dividirla usando /=. (Esto solo funciona para variables que contengan valores reales y expresiones, no para las cadenas).

28.4 Expresiones

Las expresiones pueden ser números reales (p. ej. 3.4), cadenas entre comillas simples o dobles (p. ej. `'hola'` o `"hola"`) u otras más complicadas. Para las expresiones, existen los siguientes operadores binarios (en orden de prioridad):

- `&&`, `||`: funciones Booleanas (`&&` para la función and, `||` para la función)
- <, <=, ==, !=, >, >=: comparaciones, el resultado es true (1) o false (0)
- +, −: adición, sustracción
- *, /, `div`, `mod`: multiplicación, división, división entera y módulo.

También, tenemos los siguientes operadores unarios:

- !: not, convierte un valor verdadero en falso y uno falso en verdadero
- −: cambio de signo

Como valores se pueden emplear números, variables o funciones que devuelvan algún valor. Las sub-expresiones se pueden colocar entre paréntesis. Todos los operadores funcionan para valores reales. Las comparaciones también funcionan para las cadenas y el + concatena cadenas.

Ejemplo
Aquí tenemos un ejemplo con algunas asignaciones.

```
{
  x = 23;
  str = 'hola mundo';
  y += 5;
  x *= y;
  x = 23*((2+4) / sin(y));
  str = 'hola' + " mundo";
  b = (x < 5) && !(x==2 || x==4);
}
```

28.5 Más sobre variables

Puedes crear nuevas variables al asignándoles un valor (no es necesario declararlas antes). Si simplemente usas un nombre de variable, la variable será almacenada sólo para la instancia actual. Por lo que no esperes encontrarla cuando manejes otro objeto (u otra instancia del mismo objeto). También se puede cambiar y leer variables de otros objetos colocando el nombre del objeto con un punto antes del nombre de la variable.

Para crear variables globales, cuyo valor pueda ser visto por todas las instancias, coloca antes la palabra `global` y un `punto`. Por ejemplo puedes escribir:

```
{
  if (global.hacer)
  {
    // hacer algo
    global.hacer = false;
  }
}
```

28.6 Direccionando variables en otras instancias

Como se dijo antes, puedes alterar variables en la instancia actual usando sentencias como

```
x = 3;
```

Pero en ciertos casos querrás acceder a variables en otra instancia. Por ejemplo, para detener el movimiento de todas las pelotas, o para mover al personaje principal a cierta posición, o, en el caso de una colisión, cambiar el sprite de la otra instancia involucrada. Esto puede lograrse antecediendo el nombre del objeto y un punto al nombre de la variable. Así por ejemplo, puedes escribir

```
pelota.speed = 0;
```

Esto cambiará la velocidad de todas las instancias del objeto pelota. Hay ciertos "objetos" especiales.

- `self`: La instancia actual para la que estamos ejecutando la acción
- `other`: La otra instancia involucrada en un evento de colisión
- `all`: Todas las instancias
- `noone`: Ninguna instancia (tal vez te parezca raro pero puede ser útil como veremos más adelante)
- `global`: No es precisamente una instancia, sino un contenedor que almacena variables globales

Así, por ejemplo, puedes usar las siguientes sentencias:

```
other.sprite_index = sprite5;
all.speed = 0;
global.mensaje = 'Buen resultado';
global.x = pelota.x;
```

Ahora tal vez te estés preguntando lo que la última tarea realiza cuando hay más de una pelota. Bien, se toma la primera y su valor x es asignado al valor global.

Pero qué tal si deseas establecer la velocidad de una pelota en particular, en lugar de la de todas ellas. Esto es un poco más difícil. Cada instancia tiene un id único. Cuando colocas instancias en un cuarto en el diseñador, este id se muestra cuando colocas el ratón sobre la instancia. Estos números son mayors o iguales a 100000. Puedes emplear estos números como la parte a la izquierda del punto. Pero ten cuidado. El punto sera interpretado como el punto decimal en el número. Para evitarlo, colocalo entre paréntesis. Así por ejemplo, asumiendo que el ide de la pelota es 100032, puedes escribir:stos números But what if you want to set the speed of one particular ball, rather than all balls. This is slightly more difficult. Each instance has a unique id. When you put instances in a room in the designer, this instance id is shown when you rest the mouse on the instance. These are numbers larger than or equal to 100000. Such a number you can also use as the left-hand side of the dot. But be careful. The dot will get interpreted as the decimal dot in the number. To avoid this, put brackets around it. So for example, assuming the id of the ball is 100032, you can write:

```
(100032).speed = 0;
```

Cuando creas una instancia en el programa, la llamada devuelve su id. Una pieza de programa válido es

```
{
    nnn = instance_create(100,100,pelota);
    nnn.speed = 8;
}
```

Esto crea una pelota y establece su velocidad. Nota que hemos asignado el id de la instancia a una variable y usamos esta variable como indicación antes del punto. Esto es completamente válido. Déjame explicarlo un poco major. Un punto es de hecho, un operador. Toma un valor como el operador de la izquierda y una variable (dirección) como el operador de la derecha, y devuelve la dirección de esta variable en particular para el objeto o instancia indicados. Todos los nombres de objetos, y los objetos especiales nombrados antes representan valores y pueden ser tratados como con cualquier otro valor. Por ejemplo, el siguiente programa es correcto: This creates a ball and sets its speed. Note that we assigned the instance id to a variable and used this variable as indication in front of the dot. This is completely valid. Let me try to make this more precise. A dot is actually an operator. It takes a value as left operand and a variable (address) as right operand, and returns the address of this particular variable in the indicated object or instance. All the object names, and the special objects indicated above simply represent values and these can be dealt with like any value. For example, the following program is valid:

```
{
    obj[0] = pelota;
    obj[1] = bandera;
    obj[0].alarm[4] = 12;
```

```
    obj[1].id.x = 12;
}
```

La última sentencia debiera interpretarse como sigue. Tomamos el id de la primera bandera. Para la instancia con ese id establecemos a 12 su coordenada x.

Los nombres de objetos, objetos especiales y los id de las instancias pueden también emplearse en otras funciones.

28.7 Arrays

Puedes emplear arrays de una o dos dimensiones en el GML. Simplemente coloca el índice entre corchetes cuadrados para un array unidimensional, y los dos índices con una coma entre ellos para los arrays bidimensionales. En el momento en que emplees un índice el array es generado. Cada array inicia en el índice 0. Por lo que debes tener cuidado al usar índices muy grandes ya que se ocupará memoria para un array grande. Nunca emplees indices negativos. El sistema coloca un límite de 32000 para cada indice y 1000000 para el tamaño total. Por ejemplo, puedes escribir lo siguiente:

```
{
  a[0] = 1;
  i = 1;
  while (i < 10) { a[i] = 2*a[i-1]; i += 1;}
  b[4,6] = 32;
}
```

28.8 Sentencia if

Una sentencia if tiene la forma

```
if (<expresión>) <sentencia>
```

o

```
if (<expresión>) <sentencia> else <sentencia>
```

La sentencia también puede ser un bloque. La expresión se evaluará. Si el valor (entre paréntesis) es <=0 (false) se ejecuta la sentencia después del else, de otra forma (true) se ejecuta la otra sentencia. Es un buen hábito colocar siempre corchetes a las sentencias en la sentencia if. Por lo que mejor usa

```
if (<expresión>)
{
  <sentencia>
}
else
{
  <sentencia>
}
```

Ejemplo
El siguiente programa mueve el objeto hacia el centro de la pantalla.

```
{
  if (x<200) {x += 4} else {x -= 4};
}
```

28.9 Sentencia repeat

Una sentencia repeat tiene la forma

```
repeat (<expresión>) <sentencia>
```

La sentencia se repite el número de veces indicado por el valor redondeado de la expresión.

Ejemplo
El siguiente programa crea cinco pelotas en posiciones aleatorias.

```
{
  repeat (5) instance_create(random(400),random(400),pelota);
}
```

28.10 Sentencia while

Una sentencia while tiene la forma

```
while (<expresión>) <statement>
```

Mientras la expresión sea verdadera, la sentencia (que puede también ser un bloque) es ejecutada. Ten cuidado con tus ciclos while. Puedes fácilmente hacer que se repitan eternamente, en cuyo caso el juego se bloqueará y ya no responderá a los comandos del usuario.

Ejemplo
El siguiente programa intenta colocar al objeto actual en una posición libre (esto es casi lo mismo que la acción para mover un objeto a una posición aleatoria).

```
{
  while (!place_free(x,y))
  {
    x = random(room_width);
    y = random(room_height);
  }
}

}
```

28.11 Sentencia do

Una sentencia do tiene la forma

```
do <sentencia> until (<expresión>)
```

La sentencia (que puede también ser un bloque) es ejecutada hasta que la expresión sea verdadera. La sentencia se ejecuta por lo menos una vez. Ten cuidado con los ciclos do. Puedes fácilmente crear uno que se repita indefinidamente, en cuyo caso el juego se bloqueará y ya no responderá a los eventos generados por el usuario.

Ejemplo
El siguiente programa intenta colocar el objeto actual en una posición libre (es casi lo mismo que la acción para mover un objeto a una posición aleatoria).

```
{
  do
  {
    x = random(room_width);
    y = random(room_height);
  }
  until (place_free(x,y))
}
```

28.12 Sentencia for

Una sentencia for tiene la forma

```
for (<sentencia1> ; <expresión> ; <sentencia2>) <sentencia3>
```

Funciona de la manera siguiente. Primero se ejecuta la sentencia1. entonces se evalúa la expresión. Si es verdadera, se ejecuta la sentencia3; entonces la sentencia2 y luego se evalúa nuevamente la expresión. Esto continúa hasta que la expresión sea falsa.

Puede sonar complicado. Debes interpretarlo de la manera siguiente. La primera sentencia inicializa el ciclo for. La expresión prueba si el ciclo debiera terminar. La sentencia2 es la sentencia de paso hacia la evaluación del siguiente ciclo.

El uso más común es para llevar un contador hasta cierto valor.

Ejemplo
El siguiente programa inicializa un array lista de longitud 10 con los valores 1-10.

```
{
  for (i=0; i<9; i+=1) lista[i] = i+1;
}
```

28.13 Sentencia switch

En ciertas situaciones querrás llevar a cabo alguna acción dependiendo de un valor en particular. Puedes lograrlo empleando varias sentencias if pero es más sencillo si empleas la sentencia switch. Una sentencia switch tiene la siguiente forma:

```
switch (<expresión>)
{
```

```
    case <expresión1>: <sentencia1>; … ; break;
    case <expresión2>: <sentencia2>; … ; break;
    …
    default: <sentencia>; …
}
```

Funciona como sigue. Primero se ejecuta la expresión. Después se compara con los resultados de las diferentes expresiones delante de las sentencias case. La ejecución continúa después de la sentencia case con el valor correcto, hasta que se encuentre una sentencia break. Si no se encuentra una sentencia case con el valor correcto, la ejecución continúa después de la sentencia default. (No es necesaria la sentencia default. Nota que se pueden colocar múltiples sentencias case para la misma sentencia. También, no es necesaria la sentencia break. Si no existe una sentencia break, la ejecución simplemente continúa con el código para la siguiente sentencia case.This works as follows. First the expression is executed. Next it is compared with the results of the different expressions after the case statements. The execution continues after the first case statement with the correct value, until a break statement is encountered. If no case statement has the right value, execution is continued after the default statement. (No default statement is required. Note that multiple case statements can be placed for the same statement. Also, the break is not required. If there is no break statement the execution simply continues with the code for the next case statement.

Ejemplo
El siguiente programa lleva a cabo una acción de acuerdo a la tecla que sea presionada.

```
switch (keyboard_key)
{
  case vk_left:
  case vk_numpad4:
    x -= 4; break;
  case vk_right:
  case vk_numpad6:
    x += 4; break;
}
```

28.14 Sentencia break

La sentencia break tiene la forma

```
break
```

Si se emplea en un ciclo for, while, repeat, en una sentencia switch o with, finaliza el ciclo o sentencia. Si es empleada fuera de una de estas sentencias finaliza el programa (no el juego).

28.15 Sentencia continue

La sentencia continue tiene la forma

```
continue
```

Si se emplea dentro de un ciclo for, while, repeat o con una sentencia with, continua con el siguiente valor del ciclo for o de la sentencia with.

28.16 Sentencia exit

La sentencia exit tiene la forma

```
exit
```

Simplemente termina la ejecución del programa/script actual. (¡No termina la ejecución del juego! Para ello necesitas la función `game_end();` ver más abajo)

28.17 Funciones

Una función tiene la siguiente estructura: nombre de la función, seguido por uno o varios argumentos entre paréntesis, separados por comas (también puede no incluir ningún argumento).

```
<función>(<arg1>,<arg2>,…)
```

Hay dos tipos de funciones. En primer lugar, tenemos una gran cantidad de funciones internas, para controlar todos los aspectos del juego. Después, cualquier scipt que definas en el juego puede ser empleado como una función.

Nota que para una función sin argumentos aún se necesitan los paréntesis. Algunas funciones devuelven valores y pueden ser empleadas en expresiones. Otras simplemente ejecutan órdenes.

28.18 Scripts

Cuando creas un script, querrás tener acceso a los argumentos enviados a él (ya sea cuando uses una acción script, o cuando llames al script como una función desde un programa u otro, o inclusive desde el mismo script). Estos argumentos se almacenan en las variables `argument0`, `argument1`, …, `argument9`. Por lo que puede haber como máximo 10 argumentos. (Nota: cuando se llama un script desde una acción, sólo se pueden especificar los primeros 3 argumentos).

Los scripts también pueden devolver un valor, por lo que pueden ser empleados en expresiones. Para ello debes emplear la declaración return:

```
return <expresión>
```

¡La ejecución del script termina en la declaración return! statement!

Ejemplo
Aquí está la definición de un pequeño scrpt que calcula el cuadrado del argumento:

```
{
```

```
  return (argument0*argument0);
}
```

Para llamar un script desde una pieza de código, solo hazlo como cuando se hacen las llamadas a funciones. Esto es, el nombre del script con sus argumentos entre paréntesis.

28.19 Construcciones with

Como se indicó antes, es posible leer y cambiar el valor de las variables en otras instancias. Pero en ciertos casos querrás hacer mucho más con esas otras instancias. Por ejemplo, imagina que deseas mover todas las pelotas 8 pixeles hacia abajo. Pudieras pensar que eso se logra con el siguiente código

```
pelota.y = pelota.y + 8;
```

Pero no es cierto. El valor a la derecha de la asignación obtiene la coordenada y de la primera pelota y le suma 8. Entonces este nuevo valor se toma como la coordenada y para todas las pelotas. Por lo que el resultado es que todas las pelotas tienen la misma coordenada y. La sentencia

```
pelota.y += 8;
```

tendrá exactamente el mismo efecto porque es simplemente una abreviatura de la primera declaración. Entonces, ¿cómo logramos esto? Para ello existe la declaración with. Su forma general es

```
with (<expresión>) <statement>
```

<expresión> indica una o más instancias. Para esto puedes emplear el id de la instancia, o el nombre de un objeto (para indicar todas las instancias de este objeto) o uno de los objetos especiales (all, self, other, noone). <declaración> se ejecuta para cada una de las instancias indicadas, como si la instancia fuera la instancia (self) actual. Así, para mover todas las pelotas 8 pixeles hacia abajo, puedes escribir

```
with (pelota) y += 8;
```

Si deseas ejecutar múltiples declaraciones, colócalas entre corchetes. Por ejemplo, para mover todas las pelotas a una posición aleatoria, puedes usar

```
with (pelota)
{
    x = random(room_width);
    y = random(room_height);
}
```

Nota que, dentro de las declaraciones, la instancia indicada se ha vuelto la instancia self. En tonces, la instancia self original ahora es la instancia other. Así, por ejemplo, para mover todas las pelotas a la posición de la instancia actual, puedes usar

```
with (pelota)
{
    x = other.x;
    y = other.y;
}
```

El uso de la declaración with es muy poderoso. A continuación te muestro unos cuantos ejemplos más. Para destruir todas las pelotas usas

```
with (pelota) instance_destroy();
```

Si una bomba explota y quieres destruir todas las instancias cercanas a ella puedes usar

```
with (all)
{
  if (distance_to_object(other) < 50) instance_destroy();
}
```

28.20 Comentarios

Puedes agregar comentarios a tus programas. Todo en una línea después de // no se ejecuta. Internamente se emplea para indicar el fin de una línea.

28.21 Funciones y variables en GML

El GML contiene un gran número de funciones y variables internas. Con ellas puedes controlar cualquier parte del juego. Para todas las acciones existe una función correspondiente por lo que de hecho no necesitas emplear ninguna acción si prefieres emplear código. Pero hay muchas más funciones y variables que controlan aspectos del juego que no se pueden acceder sólo empleando acciones. Por lo que si deseas crear juegos más avanzados se te recomienda leer los siguientes capítulos para tener un panorama general de todo lo que es posible lograr. Por favor nota que estas variables y funciones pueden también emplearse cuando se envían valores para las acciones. Por lo que aún si no planeas emplear código o escribir algún script, aún obtendrás beneficios de esta información.

En los capítulos siguientes se emplean las siguientes convenciones. Los nombres de variables marcados con un * son sólo de lectura, es decir, no se puede cambiar su valor. Los nombres de variables con [0..n] después de ellos son arrays. Se da el intervalo posible de sus índices.

Capítulo 29 Haciendo cálculos

El *Game Maker* contiene un gran número de funciones para llevar a cabo diversos cálculos. Aquí está una lista completa.

29.1 Constantes

Se tienen las siguientes constantes:

`true` Igual a 1.
`false` Igual a 0.
`pi` Igual a 3.1415…

29.2 Funciones de valores reales

Se tienen las siguientes funciones que manejan números reales.

`random(x)` Devuelve un número real entre 0 y x. El número siempre es menor que x.
`abs(x)` Devuelve el valor absoluto de x.
`sign(x)` Devuelve el signo de x (-1 or 1).
`round(x)` Devuelve x redondeado al entero más cercano.
`floor(x)` Devuelve x redondeado al entero inmediato inferior.
`ceil(x)` Devuelve x redondeado al entero inmediato superior.
`frac(x)` Devuelve la parte fraccional de x, es decir, la parte después del punto decimal.
`sqrt(x)` Devuelve la raíz cuadrada de x. x no debe ser negativo.
`sqr(x)` Devuelve x*x.
`power(x,n)` Devuelve x elevado a la potencia n.
`exp(x)` Devuelve e elevado a la potencia x.
`ln(x)` Devuelve el logaritmo natural de x.
`log2(x)` Devuelve el logaritmo base 2 de x.
`log10(x)` Devuelve el logaritmo base 10 de x.
`logn(n,x)` Devuelve el logaritmo base n de x.
`sin(x)` Devuelve el seno de x (x en radianes).
`cos(x)` Devuelve el coseno de x (x en radianes).
`tan(x)` Devuelve la tangente de x (x en radianes).
`arcsin(x)` Devuelve el seno inverso de x.
`arccos(x)` Devuelve el coseno inverso de x.
`arctan(x)` Devuelve la tangente inversa de x.
`arctan2(y,x)` Calcula arctan(Y/X), y devuelve un ángulo en el cuadrante correcto.
`degtorad(x)` Convierte grados a radianes.
`radtodeg(x)` Convierte radianes a grados.
`min(x,y)` Devuelve el mínimo de x y y.
`max(x,y)` Devuelve el máximo entre x y y.

min3(x,y,z) Devuelve el mínimo de x, y y z.
max3(x,y,z) Devuelve el máximo de x, y y z.
mean(x,y) Devuelve el promedio de x y y.
point_distance(x1,y1,x2,y2) Devuelve la distancia entre el punto (x1,y1) y el punto (x2,y2).
point_direction(x1,y1,x2,y2) Devuelve la dirección del punto (x1,y1) hacia el punto (x2,y1) en grados.
is_real(x) Devuelve 1 si x es un valor real (comparado contra un string).
is_string(x) Devuelve 1 si x es un string (comparado contra un valor real).

29.3 Funciones para el manejo de strings

Las siguients funciones manejan caracteres y cadenzas (strings).

chr(val) Devuelve una cadena que contiene el caracter con valor de código ascii val.
ord(str) Devuelve el código ascii del primer caracter en str.
real(str) Convierte str en un número real. str puede contener un signo menos, un punto decimal e inclusive una parte exponencial.
string(val) Convierte un valor real en una cadena empleando un formato estándar (no hay parte decimal si es un entero, y dos decimales para los demás casos).
string_format(val,tot,dec) Convierte val en una cadena empleando tu propio formato: tot indica el número total de caracteres y dec indica el número de decimales.
string_length(str) Devuelve el número de caracteres en la cadena.
string_pos(substr,str) Devuelve la posición de substr en str (0 = substr no aparece en str).
string_copy(str,index,count) Devuelve una subcadena de str, empezando en la posición index, y con longitud count.
string_char_at(str,index) Devuelve el caracter en posición index de la cadena str.
string_delete(str,index,count) Devuelve una copia de str con la parte removida que empieza en la posición index con longitud count.
string_insert(substr,str,index) Devuleve una copia de str con substr insertado en la posición index.
string_replace(str,substr,newstr) Devuelve una copia de str con la primer ocurrencia de substr reemplazada por newstr.
string_replace_all(str,substr,newstr) Devuelve una copia de str con todas las ocurrencias de substr reemplazadas por newstr.
string_count(substr,str) Devuelve el número de ocurrencias de substr en str.
string_lower(str) Devuelve una copia de str en minúsculas
string_upper(str) Devuelve una copia en mayúsculas de str.
string_repeat(str,count) Devuelve una cadena que consiste de count copias de str.

`string_letters(str)` Devuelve una cadena que solo contiene las letras en str.
`string_digits(str)` Devulve una cadena que solo contiene los dígitos en str.
`string_lettersdigits(str)` Devuelve una cadena que contiene las letras y dígitos en str.

Las siguientes funciones acceden al portapapeles para almacenar texto.

`clipboard_has_text()` Devuelve si hay texto en el portapapeles.
`clipboard_get_text()` Devuelve el texto actual en el portapapeles.
`clipboard_set_text(str)` Coloca la cadena str en el portapapeles.

Capítulo 30 GML: Game play

Hay una gran cantidad de variables y funciones que puedes emplear para definir el game play. Estas en particular influyen en el movimiento y creación de instancias, el timing, y el manejo de los eventos.

30.1 Moviéndose

Obviamente, un aspecto importante de los juegos es el movimiento de las instancias de los objetos. Cada instancia tiene dos variables internas x y y que indican la posición de la instancia. (Para ser precisos, indican el lugar donde se encuentra el punto de origen del sprite). La posición (0,0) es la esquina superior izquierda del cuarto. Puedes cambiar la posición de la instancia al cambiar los valores de sus variables x y y. Es lo que debes hacer si deseas movimientos más complicados. Este código normalmente se coloca en el evento sep del objeto.

Si el objeto se mueve con velocidad y dirección constants, hay una manera más fácil de lograrlo. Cada instancia tiene una velocidad horizontal (`hspeed`) y vertical (`vspeed`). Ambas se indican en pixeles por paso (step). Una velocidad horizontal positiva indica movimiento a la derecha, una velocidad horizontal negativa indica movimiento a la izquierda. La velocidad vertical positiva es movimiento hacia abajo y la negativa indica movimiento hacia arriba. Por lo que sólo debes establecer estos valores una vez (por ejemplo en el evento de creación) para dar al objeto un movimiento constante.

Hay otra manera muy diferente de especificar el movimiento, usando dirección (en grados 0-359), y velocidad (no debe ser negativa). Puedes configurar y leer estas variables para especificar un movimiento arbitrario. (Internamente se convierte a valores de `hspeed` y `vspeed`). También tenemos la fricción y la gravedad, y la dirección de la gravedad. Finalmente, tenemos la función `motion_add(dir,speed)` para agregar movimiento al actual.

Para concluir, cada instancia tiene las siguientes variables y funciones referents a su posición y movimiento:

`x` Su posición x.
`y` Su posición y.
`xprevious` Su posición x anterior.
`yprevious` Su posición y previa.
`xstart` Su posición x inicial en el cuarto.
`ystart` Su posición y inicial en el cuarto.
`hspeed` Componente horizontal de la velocidad.
`vspeed` Componente vertical de la velocidad.
`direction` Su dirección actual (0-360, contra las manecillas del reloj, 0 = a la derecha).
`speed` Su velocidad actual (pixels por step).
`friction` Fricción actual (pixels por step).

gravity Cantidad actual de gravedad (pixels por paso).
gravity_direction Dirección de la gravedad (270 es hacia abajo).
motion_set(dir,speed) Establece el movimiento a la velocidad speed y la dirección dir.
motion_add(dir,speed) Agrega el movimiento al movimiento actual (como una suma vectorial).
path_index Indice del path actual que la instancia sigue. –1 indica que no hay path.
path_position Posición en el path actual. 0 es el inicio del path. 1 es el fin del path.
path_orientation Orientación (contra las manecillas del reloj) en el que se ejecuta el path. 0 es la orientación normal del path.
path_scale Escala del path. Increméntala para hacer al path más grande. 1 es el valor por defecto.

Se tiene una gran cantidad de funciones disponibles que te ayudarán a la definición de movimientos:

place_free(x,y) Devuelve si la instancia colocada en la posición (x, y) está libre de colisión. Normalmente se emplea para revisar antes de mover la instancia a la nueva posición.
place_empty(x,y) Devuelve si la instancia colocada en la posición (x, y) no se encuentra con nadie. Esta función también tma en cuenta las instancias no sólidas.
place_meeting(x,y,obj) Devuelve si la instancia colocada en la posición (x,y) se encuentra con un el objeto obj. obj puede ser un objeto en cuyo caso la función devuelve verdadero si se encuentra con una instancia de ese objeto. También puede ser el id de una instancia, o la palabra especial `other`.
place_snapped(hsnap,vsnap) Devuelve si la instancia está alineada con los valores de snap hsnap y vsnap.

move_random(hsnap,vsnap) Mueve la instancia a una posición libre, y la alinea con los valores hsnap y vsnap, alineada, al igual que la acción correspondiente.
move_snap(hsnap,vsnap) Alinea la instancia, como la acción correspondiente.
move_towards_point(x,y,sp) Mueve la instancia con velocidad sp hacia el punto (x,y).
move_bounce_solid(adv) Rebotar contra objetos sólidos, como la acción correspondiente. adv indica si se emplea rebote avanzado, que toma en cuenta las paredes inclinadas.
move_bounce_all(adv) Rebotar contra todas las insntancias, en lugar de solo con las sólidas.
move_contact_solid(dir,maxdist) Mover la instancia en la dirección dir hasta que haya contacto con un objeto sólido. Si no hay collision en la posición actual, la instancia es colocada justo antes de donde ocurre una colisión. Si ya hay

una collision en la posición actual, la instancia no se mueve. Puedes especificar la distancia máxima a mover la instancia maxdist (emplea un número negativo para indicar distancia arbitraria).

move_contact_all(dir,maxdist) Igual que la función anterior pero esta vez se detiene hasta que haya contacto con cualquier objeto, no solo sólidos.

move_outside_solid(dir,maxdist) Mueve la instancia en la direción dir hata que no esté al alcance de un objeto sólido. Si no hay collision en la posición actual, no se mueve la instancia. Puedes especificar la distancia máxima a mover (usa un valor negativo para indicar una distancia arbitraria).

move_outside_all(dir,maxdist) Igual que la anterior pero se mueve hasta estar fuera de alcance de cualquier objeto, no solo objetos sólidos.

distance_to_point(x,y) Devuelve la distancia de la caja límite de la instancia actual hacia el punto (x,y).

distance_to_object(obj) Devuelve la distancia de la instancia actual a la instancia más cercana del objeto obj.

position_empty(x,y) Indica si no hay nada en la posición (x,y).

position_meeting(x,y,obj) Indica si en la posición (x,y) hay una instancia obj. obj puede ser un objeto, una id de una instancia, o las palabras clave `self`, `other` o `all`.

30.2 Instancias

En el juego, las unidades básicas son las instancias de los diferentes objetos. Durante el juego puedes cambiar varios aspectos de estas instancias. También puedes crear nuevas instancias y destruir otras. Además de las variables relacionadas con el movimiento mostradas arriba y las variables relacionadas con el dibujo de las instancias que se tratarán más adelante, cada instancia cuenta con las siguientes variables:

object_index* Índice del objeto del cual es instancia. Esta variable no puede ser cambiada.

id* El identificador único para la instancia (>=100000). (Nota: al definir cuartos siempre se muestra el id de la instancia bajo el cursor).

mask_index Índice del sprite empleado como mascara para las colisiones. Da un valor de –1 para que que sea igual a sprite_index.

solid Indica si la instancia es sólida. Puede cambiarse durante el juego.

persistent Indica si la instancia es persistente y reaparecerá al moverse a un Nuevo cuarto. Normalmente desactivas la persistencia en ciertos momentos. (Por ejemplo si vuelves al cuarto anterior).

Sólo hay un problema al manejar instancias, no es fácil identificar instancias individuales. No cuentan con un nombre. Cuando sólo hay una instancia de un objeto en particular puedes usar el nombre del objeto pero en otros casos requerirás obtener el id de la instancia. Este es un identificador único para la instancia, puedes emplearlo en las sentencias **with** y como identificador de un objeto (el uso de la construcción . se describe

en la sección 28.6). Afortunadamente hay varias variables y rutinas que te ayudan a obtejer los id de las instancias.

instance_count* El número de instancias que existen actualmente en el cuarto.
instance_id[0..n-1]* El id de la instancia en particular. n es el número de la instancia.

Permíteme darte un ejemplo. Considera que cada unidad en tu juego tiene un poder en particular y quieres encontrar la más poderosa, pudieras usar el siguiente trozo de código:

```
{
  maxid = -1;
  maxpower = 0;
  for (i=0; i<instance_count; i+=1)
  {
    iii = instance_id[i];
    if (iii.object_index == unit)
    {
      if (iii.power > maxpower)
        {maxid = iii; maxpower = iii.power;}
    }
  }
}
```

Después del ciclo, maxid contendrá el id de la unidad con mayor poder. (No destruyas instancias durante un ciclo de este tipo porque se quitarán automáticamente del array y como resultado empezarás a saltarte instancias).

instance_find(obj,n) Devuelve el id de la instancia (n+1)ésima de tipo obj. obj puede ser un objeto o la palabra clave all. Si obj no existe, se devuelve el objeto especial `noone`.
instance_exists(obj) Indica si la instancia de tipo obj existe. obj puede ser un objeto, un id de una instancia, o la palabra clave all.
instance_number(obj) Devuelve el número de instancias de tipo obj. obj pude ser un objeto o la palabra clave all.
instance_position(x,y,obj) Devuelve el id de la instancia de tipo obj en la posición (x,y). cuando varias instancias están en esa posición se devuelve la primera. obj puede ser un objeto o la palabra clave all. Si no existe, se devuelve el objeto especial `noone`.
instance_nearest(x,y,obj) Devuelve el id de la instancia de tipo obj más cercana al punto (x,y). obj puede ser un objeto o la palabra clave all.
instance_furthest(x,y,obj) Devuelve el id de la instancia de tipo obj más alejada del punto (x,y). obj puede ser un objeto o la palabra clave all.
instance_place(x,y,obj) Devuelve el id de la instancia de tipo obj encontrado cuando la instancia actual es colocada en la posición (x,y). obj puede ser un objeto o la palabra clave all. Si no existe, se devuelve el objeto especial `noone`.

Las siguientes funciones pueden emplearse para crear y destruir instancias.

instance_create(x,y,obj) Crea una instancia de obj en la posición (x,y). La función devuelve el id de la nueva instancia.
instance_copy(performevent) Crea una copia de la instancia actual. El argumento indica si el evento de creación se debe ejecutar para la copia. La función devuelve el id de la nueva copia.
instance_destroy() Destruye la instancia actual.
instance_change(obj,perf) Camba la instancia en obj. perf indica si se deben ejecutar los eventos de destrucción y creación.
position_destroy(x,y) Destruye todas las instancias cuyo sprite contenga la posición (x,y).
position_change(x,y,obj,perf) Cambia todas las instancias en (x,y) en obj. perf indica si se deben ejecutar los eventos de destrucción y creación.

30.3 Timing

Los buenos juegos requirieron de cuidado especial de los tiempos en que las cosas se llevaban a cabo (timing). Afortunadamente el *Game Maker* se ocupa de la mayor parte del timing por ti. Se asegura de que las cosas ocurran con un ritmo constante. Este ritmo es definido al definir los cuartos. Pero puedes cambiarlo usando la variable global `room_speed`. Así por ejemplo, puedes incrementar lentamente la velocidad del juego, haciéndolo más difícil, agregando una muy pequeña cantidad (como 0.001) a `room_speed` en cada step. Si tu máquina es lenta la velocidad del juego pudiera no alcanzarse. Esto puede checarse usando la varible `fps` que monitorea constantemente el número actual de cuadros por segundo. Finalmente, para un timing avanzado puedes usar la variable `current_time` que te da el número de milisegundos desde que la computadora fue iniciada. Aquí está la colección completa de variables disponibles (sólo la primera puede ser cambiada):

room_speed Velocidad del juego en el cuarto actual (en steps por segundo).
fps* Número de cuadros que son dibujados por segundo.
current_time* Número de milisegundos que han pasado desde que el sistema fue iniciado.
current_year* El año actual.
current_month* El mes actual.
current_day* El día actual.
current_weekday* El día actual de la semana (1=domingo, …, 7=sábado).
current_hour* La hora actual.
current_minute* El minuto actual.
current_second* El segundo actual.

Algunas veces querrás detener el juego por un corto periodo. Para esto, usa la función sleep.

sleep(numb) Pausa el juego por numb milisegundos.

Como debes saber, cada instancia tiene 8 diferentes alarmas que puedes configurar. Para cambiar los valores (u obtener los valores) de las diferentes alarmas usa la siguiente variable:

> **alarm[0..7]** Valor de la alarma indicada. (Nota: ¡las alarmas solo se actualizan cuando el evento de alarma para el objeto contiene acciones!)

Hemos visto que para los problemas de un timing complejo puedes usar el recurso de las líneas de tiempo (time lines). Cada instancia puede tener un recurso time line asociado con ella. Las siguientes variables están relacionadas con esto:

> **timeline_index** Índice de la time line asociada con la instancia. Puedes establecerlo a una time line en particular para usarla. Ponlo en –1 para pdejar de usar la time line para la instancia.
> **timeline_position** Posición actual dentro de la time line. Puedes cambiarla para saltar o repetir ciertas partes.
> **timeline_speed** Normalmente, en cada step la posición en la time line se incrementa en 1. Puedes cambiar esta cantidad configurando esta variable a un valor diferente. Puedes usar números reales, p. ej. 0.5. si el valor es mayor que uno, varios momentos pueden ocurrir dentro del mismo tiempo del step. Se realizarán en el orden correcto, por lo que no se saltará ninguna acción.

30.4 Cuartos y score

Los juegos funcionan en cuartos. Cada cuarto tiene un índice que se indica por el nombre del cuarto. El cuarto actual es almacenado en la variable `room`. No puedes asumir que los cuartos están numerados en un orden consecutivo. Por lo que nunca sumes o restes un número de la variable `room`. En lugar de ello usa las funciones y variables indicadas abajo. Por lo que una típica pieza de código que usarás sería:

```
{
  if (room != room_last)
  {
    room_goto_next();
  }
  else
  {
    game_end();
  }
}
```

Las siguientes variables y funciones se relacionan con los cuartos (rooms).

> **room** Índice del cuarto actual; puede cambiarse para ir a un cuarto diferente, pero major usa las rutinas listadas abajo.
> **room_first*** Índice del primer cuarto en el juego.
> **room_last*** Índice del ultimo cuarto en el juego.
> **room_goto(numb)** Ir al cuarto con indice numb.

room_goto_previous() Ir al cuarto anterior.
room_goto_next() Ir al siguiente cuarto.
room_restart() Reiniciar el cuarto actual.
room_previous(numb) Devuelve el índice del cuarto anterior aa numb (-1 = ninguno) pero no va a él.
room_next(numb) Devuelve el índice del cuarto posterior a numb (-1 = ninguno).
game_end() Finaliza el juego.
game_restart() Reinicia el juego.

Los cuartos tienen varias propiedades adicionales:

room_width* Ancho del cuarto en pixeles.
room_height* Alto del cuarto en pixeles.
room_caption Título de la ventana del cuarto.
room_persistent Indica si el cuarto es persistente.

Muchos juegos ofrecen al jugador la posibilidad de guardar el juego y cargar un juego guardado. En el *Game Maker* esto ocurre automáticamente cuando el jugador presiona <F5> para guardar y <F6> para cargar. También puedes guardar y cargar juegos desde una pieza de código (nota que la carga sólo se lleva a cabo al final del step actual).

game_save(string) Guarda el juego al archivo con nombre string.
game_load(string) Carga el juego del archivo con nombre string.

Otro aspecto importante de muchos juegos es el score, la energía, y el número de vidas. El *Game Maker* mantiene el score en la variable global `score` y el número de vidas en la variable global `lives`. Puedes cambiar el score simplemente cambiado el valor de esta variable. Lo mismo aplica para la energía y las vidas. Si la variable `lives` es mayor que 0 y se vuelve menor o igual a 0 se ejecuta el evento no-more-lives para todas las instancias. Si no quieres mostrar el score y las vidas en el título, pon la variable `show_score`, `etc.`, a falso. También puedes cambiar el título. Para juegos más complicados major muestra el score tú mismo.

score El marcador actual.
lives El número de vidas.
health La energía actual (0-100).
show_score Indica si se muestra el marcador en el título de la ventana.
show_lives Indica si se muestra el número de vidas en el título de la ventana.
show_health Indica si se muestra la energía en el título de la ventana.
caption_score El título empleado para el marcador.
caption_lives El título empleado para el número de vidas.
caption_health El título para la energía.

También hay un mecanismo interno para manejar una lista de los mejores marcadores. Puede contener hasta diez nombres. Para más información, ve el Capítulo 34.

30.5 Generando eventos

Como sabes, el *Game Maker* es completamente manejado por eventos. Todas las acciones ocurren como resultado de eventos. Hay una gran cantidad de eventos diferentes. Los eventos de creación y destrucción ocurren cuando una instancia es creada o destruída. En cada step, el sistema maneja primero los eventos de alarma. Después los eventos de teclado y ratón, y luego el siguiente evento step. Después de esto las instancias son colocadas en su nueva posición después de lo cual se maneja el evento de colisión. Finalmente el evento draw se usa para dibujar las instancias (nota que cuando empleas múltiples vistas el evento draw es llamado varias veces en cada step). También puedes aplicar un evento a la instancia actual desde una pieza de código. Se tienen las siguientes funciones:

`event_perform(type,numb)` Realiza el evento numb del tipo type para la instancia actual. Se pueden emplear los siguientes tipos de eventos:

```
ev_create
ev_destroy
ev_step
ev_alarm
ev_keyboard
ev_mouse
ev_collision
ev_other
ev_draw
ev_keypress
ev_keyrelease
```

Cuando hay varios eventos del tipo dado, numb puede usarse para especificar el evento preciso. Para el evento de alarma numb puede tener un valor de 0 a 7. Para el evento de teclado puedes usar el código de tecla para la tecla. Para los eventos de ratón puedes usar las siguientes constantes:

```
ev_left_button
ev_right_button
ev_middle_button
ev_no_button
ev_left_press
ev_right_press
ev_middle_press
ev_left_release
ev_right_release
ev_middle_release
ev_mouse_enter
ev_mouse_leave
ev_joystick1_left
ev_joystick1_right
ev_joystick1_up
ev_joystick1_down
ev_joystick1_button1
```

ev_joystick1_button2
ev_joystick1_button3
ev_joystick1_button4
ev_joystick1_button5
ev_joystick1_button6
ev_joystick1_button7
ev_joystick1_button8
ev_joystick2_left
ev_joystick2_right
ev_joystick2_up
ev_joystick2_down
ev_joystick2_button1
ev_joystick2_button2
ev_joystick2_button3
ev_joystick2_button4
ev_joystick2_button5
ev_joystick2_button6
ev_joystick2_button7
ev_joystick2_button8

Para el evento de collision proporcionas el índice del otro objeto. Finalmente, para el evento other puedes usar las siguientes constantes:

ev_outside
ev_boundary
ev_game_start
ev_game_end
ev_room_start
ev_room_end
ev_no_more_lives
ev_no_more_health
ev_animation_end
ev_end_of_path
ev_user0
ev_user1
ev_user2
ev_user3
ev_user4
ev_user5
ev_user6
ev_user7

Para el evento step puedes dar el índice usando las siguientes constantes:

ev_step_normal
ev_step_begin
ev_step_end

event_perform_object(obj,type,numb) Esta función trabaja igual que la anterior pero esta vez puedes especificar eventos en otro objeto. Nota que las acciones en estos eventos se aplican a la instancia actual, no a las instancias del objeto dado.

event_user(numb) En los eventos other también puedes definir 8 eventos definidos por el usuario. Estos son ejecutados solo si llamas esta función. numb debe tener valores de 0 a 7.

`event_inherited()` Ejecuta el evento heredado. Esto solo funciona si la instancia tiene un objeto padre.

Puedes obtener información sobre el evento actualmente ejecutado usando las siguientes variables de solo lectura:

`event_type*` El tipo del evento que se está ejecutando.

`event_number*` El número del evento que se está ejecutando.

`event_object*` El índice del objeto para el cual se está ejecutando el evento actual.

`event_action*` El índice de la acción que está siendo ejecutada (0 es la primera en el evento, etc.)

30.6 Variables misceláneas y funciones

Aquí están algunas variables y funciones que se refieren a los errores.

`error_occurred` Indica si ha ocurrido un error

`error_last` Cadena que indica el ultimo mensaje de error

`show_debug_message(str)` Muestra la cadena str en modo debug

Capítulo 31 GML: Interacción con el usuario

No hay juego sin interacción con el usuario. La manera estándar de interactuar con el usuario en el *Game Maker* es colocando acciones en los eventos del ratón o del teclado. Pero en ocasiones se necesita más control. Desde una pieza de código puedes checar la posición del ratón o si alguno de sus botones es presionado. Normalmente esto se checa en el evento step de algún objeto controlador y llevas a cabo las acciones adecuadas. Tenemos las siguientes variables y funciones:

mouse_x* coordenada-X del ratón. No puede cambiarse.
mouse_y* coordenada-Y del ratón. No puede cambiarse.
mouse_button Botón del ratón presionado actualmente. como valores puedes emplear mb_none (ningún botón), mb_any (cualquier botón), mb_left (botón izquierdo), mb_middle (botón central) o mb_right (botón derecho).
keyboard_lastkey Código de la última tecla presionada. Lee más abajo las constantes de los códigos de las teclas. Puedes cambiarlo, p. ej. ponerlo a 0 si tú lo manipulaste.
keyboard_key Código de tecla de la tecla presionada actualmente (ve a continuación; 0 si no hay tecla presionada).
keyboard_lastchar Último caracter introducido (como string).
keyboard_string Cadena de caracteres que contiene los últimos 80 caracteres introducidos. Esta cadena solo contendrá los caracteres imprimibles en pantalla. También responde a la tecla de retroceso borrando el ultimo caracter.

En ocasiones es útil mapear una tecla a otra. Por ejemplo pudieras permitir al jugador emplear tanto las teclas del cursor como las del teclado numérico. En lugar de duplicar las acciones puedes mapear el teclado numérico a las teclas del cursor. También pudieras implementar un mecanismo en el que el jugador pueda seleccionar las teclas a usar. Para este fin, contamos con las siguientes funciones

keyboard_set_map(key1,key2) Mapea la tecla con el código de tecla key1 a la tecla key2.
keyboard_get_map(key) Devuelve el mapeado actual de la tecla key.
keyboard_unset_map() Reestablece todas las teclas a su mapa original.

Para checar si una tecla o botón del ratón en particular han sido presionados puedes emplear las siguientes funciones. Esto es útil particularmente cuando se presionan varias teclas simultáneamente.

keyboard_check(key) Indica si la tecla con el código key ha sido presionada.
keyboard_check_direct(key) Indica si la tecla con el código key es presionada checando el hardware directamente. El resultado es independiente de la aplicación enfocada. Esta función permite algunos chequeos más. En este caso puedes emplear los códigos vk_lshift, vk_lcontrol, vk_lalt, vk_rshift, vk_rcontrol y vk_ralt para checar si se presiona la tecla shift, control o alt, ya sea izquierda o derecha. (¡No funciona en Windows 95!).

mouse_check_button(numb) Indica si se presiona el botón del ratón numb (como valores de numb emplea mb_none, mb_left, mb_middle, o mb_right).

Las siguientes rutinas pueden emplease para manipular el estado del teclado:

keyboard_get_numlock() Indica si BloqNum está activada.
keyboard_set_numlock(on) Activa (on=true) o desactiva (on=false) BloqNum. (No funciona en Windows 95).
keyboard_key_press(key) Simula el presionar de la tecla con el código key.
keyboard_key_release(key) Simula la liberación de la tecla con el código key.

Tenemos las siguientes constants para los códigos de las teclas:

vk_nokey código que representa que ninguna tecla está presionada
vk_anykey código que representa que alguna tecla ha sido presionada
vk_left código para la tecla de la flecha izquierda
vk_right código para la tecla de la flecha derecha
vk_up código para la tecla de la fecha hacia arriba
vk_down código para la tecla de la flecha hacia abajo
vk_enter tecla enter (o intro)
vk_escape tecla escape
vk_space tecla espacio
vk_shift tecla shift
vk_control tecla control
vk_alt tecla alt
vk_backspace tecla de retroceso
vk_tab tecla tab
vk_home tecla inicio
vk_end tecla fin
vk_delete tecla supr
vk_insert tecla insert
vk_pageup tecla AvPag
vk_pagedown tecla RePag
vk_pause tecla Pausa/Inter
vk_printscreen tecla ImpPnt/PetSis
vk_f1 … **vk_f12** códigos para las teclas de función desde F1 a F12
vk_numpad0 … **vk_numpad9** teclas numéricas en el teclado numérico
vk_multiply tecla de multiplicación en el teclado numérico
vk_divide tecla dedivisión en el teclado numérico
vk_add tecla de suma en el teclado numérico
vk_subtract tecla de resta en el teclado numérico
vk_decimal tecla del punto decimal en el teclado numérido

Para las teclas de las letras usa por ejemplo `ord('A')`. (Las letras mayúsculas). Las siguientes constants solo pueden ser empleadas en `keyboard_check_direct`:

vk_lshift tecla shift izquierda
vk_lcontrol tecla control izquierda
vk_lalt tecla alt izquierda
vk_rshift tecla shift derecha
vk_rcontrol tecla control derecha
vk_ralt tecla alt derecha

¡No funcionan en versions anteriores de Windows!

Por ejemplo, asumiendo que tienes un objeto que el usuario puede controlar con las teclas del cursor puedes colocar el siguiente código en el evento step del objeto:

```
{
  if (keyboard_check(vk_left))  x -= 4;
  if (keyboard_check(vk_right)) x += 4;
  if (keyboard_check(vk_up))    y -= 4;
  if (keyboard_check(vk_down))  y += 4;
}
```

Por supuesto que esto es mucho más fácil si simplemente lo ponemos en los eventos del teclado correspondientes.

Hay tres funciones adicionales relacionadas con la interacción con el usuario.

keyboard_clear(key) 'Limpia' el estado de la tecla key. Esto significa que no generará eventos de teclado hasta que se vuelva a presionar.
mouse_clear(button) 'Limpia' el estado del botón del ratón button. Esto significa qeu no generará eventos del ratón hasta que el jugador lo suelte y lo vuelva a presionar.
io_clear() 'Limpia' todos los estados del teclado y del ratón.
io_handle() Maneja la entrada y salida por parte del usuario, actualizando los estados del teclado y del ratón.
keyboard_wait() Espera hasta que el usuario presione una tecla del teclado.

31.1 Soporte para joystick

Tenemos algunos eventos asociados con los joysticks (mandos de control, controles, palancas de mando, palancas de juego, etc.) Pero para tener control total sobre los joysticks hay un grupo de funciones para tratarlos. El *Game Maker* soporta hasta dos joystick. Por lo que todas estas funciones reciben el id del joystick como argumento.

joystick_exists(id) Indica si el joystick id (1 o 2) existe.
joystick_name(id) Devuelve el nombre del joystick.
joystick_axes(id) Devuelve el número de ejes del joystick.
joystick_buttons(id) Devuelve el número de botones del joystick.

joystick_has_pov(id) Indica si el joystick tiene capacidades point-of-view.
joystick_direction(id) Devuelve el código (vk_numpad1 a vk_numpad9) correspondiente a la dirección del joystick id (1 o 2).
joystick_check_button(id,numb) Indica si el botón del joystick id es presionado (numb está en el intervalo 1-32).
joystick_xpos(id) Devuelve la posición (-1 a 1) del eje-x del joystick id.
joystick_ypos(id) Devuelve la posición y del joystick id.
joystick_zpos(id) Devuelve la posición z del joystick id (si es que cuenta con eje z).
joystick_rpos(id) Devuelve la posición del timón del joystick id (del cuarto eje).
joystick_upos(id) Devuelve la posición u del joystick id (del quinto eje).
joystick_vpos(id) Devuelve la posición v del joystick id (del sexto eje).
joystick_pov(id) Devuelve la posición del point-of-view del joystick id. Este es un ángulo entre 0 y 360 grados. 0 es adelante, 90 a la derecha, 180 atrás y 270 a la izquierda. Cuando no se especifica la dirección del point-of-view se devuelve –1..

Capítulo 32 GML: Gráficos del juego

Una parte importante de un juego son los gráficos. Game Maker normalmente se encarga de esto y en juegos simples no hay de que preocuparse. Pero algunas veces quieres más control. Para algunos aspectos hay acciones pero con código puedes controlar más aspectos. Este capítulo describe todas las variables para esto y da más información acerca de lo que realmente está sucediendo.

32.1 Ventana y cursor

Originalmente, el juego corre dentro de una ventana centrada. EL jugador puede cambiar esto presionando la tecla F4 a menos que esta opción no esté disponible. También puedes hacer esto desde el programa usando la siguiente variable:

> **`full_screen`** Esta variable es verdadera cuando se está en el modo de pantalla completa. Puedes cambiar el modo poniendo esta variable en verdadero o falso.

Nota que en el modo de pantalla completa el título y la puntuación se muestran en la pantalla. (Esto se puede evitar en las opciones del juego.) En el modo de pantalla completa la imagen está centrada o escalada. Puedes controlar esto usando la siguiente variable:

> **`scale_window`** Esta variable indica le porcentaje de escala en el modo de ventana. 100 indica sin escala.
> **`scale_full`** Esta variable indica el porcentaje de escala en el modo de pantalla completa. 100 indica sin escala. 0 indica la máxima escala posible.

El modo escalado puede ser lento en máquinas con un procesador o tarjeta de gráficos lentos. Los juegos corren por defecto con el cursor visible. Para la mayoría de los juegos no quieres esto. Para quitar el cursor usa la variable:

> **`show_cursor`** Cuando tiene el valor de falso, el cursor se hace invisible en el área de juego, de lo contrario es visible.

También puedes asignar el cursor como uno de los predefinidos en Windows usando la siguiente función:

> **`set_cursor(cur)`** Da al cursor el valor asignado. Puedes usar las siguientes constantes: cr_default, cr_none, cr_arrrow, cr_cross, cr_beam, cr_size_nesw, cr_size_ns, cr_size_nwse, cr_size_we, cr_uparrow, cr_hourglass, cr_drag, cr_nodrop, cr_hsplit, cr_vsplit, cr_multidrag, cr_sqlwait, cr_no, cr_appstart, cr_help, cr_handpoint, cr_size_all.

Por cierto, es muy fácil hacer tu propio objeto cursor. Solo crea un objeto con profundidad negativa que, en el evento sep, sigue la posición del mouse.

Para saber la resolución del monitor, puedes usar las siguientes variables de sólo lectura:

monitor_width El ancho del monitor, en píxeles.
monitor_height La altura del monitor, en píxeles.

32.2 Sprites e imágenes

Cada objeto tiene un sprite asociado a él. Ésta puede ser una imagen simple o consistir de imágenes múltiples. Por cada instancia del objeto, el programa dibuja la imagen correspondiente en la pantalla con su origen (definido en las propiedades del sprite) en la posición (x,y) de la instancia. Cuando hay imágenes múltiples, cicla a través de las imágenes para tener un efecto de animación. Hay un número de variables que afectan la manera en la que se dibuja la imagen. Estas pueden ser usadas para cambiar el efecto. Cada instancia tiene las siguientes variables:

visible Si es verdadera (1) la imagen es dibujada, de lo contrario no se dibuja. Las instancias invisibles están aún activas y generan eventos de colisión; sólo tú no las ves. Poner la visibilidad en falso es útil, por ejemplo, objetos controladores (hazlos no sólidos para evitar eventos de colisión) o interruptores escondidos.
sprite_index Este es el índice del sprite actual de la instancia. Puedes cambiarlo para darle a la instancia un sprite diferente. Como valor puedes usar los nombres que le hayas asignado a los diferentes sprites creados. Cambiar el sprite no cambia el índice de la sub-imagen visible actual.
sprite_width* Indica el ancho del sprite. Este valor no puede ser cambiado pero puede que quieras usarlo.
sprite_height* Indica la altura del sprite. Este valor no puede ser cambiado pero puede que quieras usarlo.
sprite_xoffset* Indica el origen horizontal del sprite como fue definido en las propiedades del sprite. Este valor no puede ser cambiado pero puede que quieras usarlo.
sprite_yoffset* Indica el origen vertical del sprite como fue definido en las propiedades del sprite. Este valor no puede ser cambiado pero puede que quieras usarlo.
image_number* El número de sub-imágenes del sprite actual. (No puede ser cambiado)
image_index Cuando la imagen tiene múltiples sub-imágenes el programa cicla a través de ellos. Esta variable indica la sub-imagen actualmente dibujada (se numeran a partir del 0). Puedes cambiar la imagen actual cambiando esta variable. El programa va a continuar ciclando a partir de este nuevo índice.
image_single Algunas veces quieres que una sub-imagen particular sea visible y no quieres que el programa cambie a través del resto. Esto se puede lograr asignando a esta variable el índice el la sub-imagen que quieres que quieres ver (la primera sub-imagen tiene el índice 0). Dale el valor de –1 para ciclar a través de las sub-imágenes. Esto es útil cuando un objeto tiene múltiples apariencias. Por ejemplo, supón que tienes un objeto que puede rotar y creas un sprite que tiene sub-imágenes para varias orientaciones (contra-reloj) Entonces, en el evento step del objeto puedes poner:

```
{
```

```
  image_single = direction * image_number/360;
}
```

image_speed La velocidad con la que se cicla a través de las sub-imágenes. Un valor de 1 indica que en cada paso (step) se obtiene la siguiente imagen. Valores más pequeños cambian la sub-imagen más lento, mostrando cada imagen múltiples ocasiones. Valores más grandes saltarán imágenes para hacer el movimiento más rápido.
depth Normalmente las imágenes son dibujadas en el orden en el que se crearon las instancias. Puedes cambiar esta indicando la profundidad (depth) de la imagen. El valor original es 0 a menos que especifiques un valor diferente en las propiedades del objeto. Mientras más alto sea el valor, la imagen está más lejos. (También puedes usar valores negativos). Las instancias con una profundidad mayor quedarán dibujadas detrás de las instancias de profundidad menor. Indicar la profundidad garantiza que las imágenes se dibujaran en el orden que quieras. (Por ejemplo, el avión enfrente de la nube). Las instancias que se usen como fondo deben de tener una profundidad positiva muy alta, y las de primer plano (foreground) una profundiad negativa muy baja.
image_scale Un factor de escala que hace grande o pequeña a las imágenes. Un valor de 1 indica el tamaño normal. Cambiarla escala también cambia los valores del ancho y alto de la imagen e influencia las colisiones como puedes esperar. Nota que las imágenes escaladas (en particular cuando las haces más pequeñas) toman más tiempo de dibujar. Cambiar la escala puede ser usado para obtener un efecto de 3-D.
image_alpha El valor de la transparencia (alpha) para usar en la imagen. Un valor de 1 es la condición normal. Un valor de 0 es completamente transparente. Úsalo con cuidado. Dibujar imágenes parcialmente transparentes toma mucho tiempo y puede reducir la velocidad del juego.
bbox_left* Lado izquierdo de la caja perimetral usado para la imagen de la instancia. (Tomando en cuenta la escala)
bbox_right* Lado derecho de la caja perimetral de la imagen de la instancia.
bbox_top* Lado superior de la caja perimetral de la imagen de la instancia.
bbox_bottom* Lado inferior de la caja perimetral de la imagen de la instancia.

32.3 Fondos

Cada cuarto puede tener hasta 8 fondos. También tiene un color de fondo. Todos los aspectos de estos fondos pueden ser cambiados usando las siguientes variables. (Algunas variables son arreglos que van del 0 al 7 indicando los diferentes fondos):

background_color El color de fondo del cuarto
background_showcolor Whether to clear the window in the background color.
background_visible[0..7] Indica sI la imagen de fondo es visible.
background_foreground[0..7] Indica si el fondo es una imagen de primer plano.
background_index[0..7] Índice de la imagen del fondo.

background_x[0..7] Posición X de la imagen de fondo.
background_y[0…7] Posición Y de la imagen de fondo.
background_width[0…7]* Ancho de la imagen de fondo.
background_height[0…7]* Altura de la imagen de fondo.
background_htiled[0..7] Indica si el fondo es de mosaico horizontalmente.
background_vtiled[0..7] Indica si el fondo es de mosaico vertical.
background_hspeed[0..7] Velocidad de desplazamiento horizontal del fondo (pixeles por paso).
background_vspeed[0..7] Velocidad de desplazamiento vertical del fondo (pixeles por paso).
background_alpha[0..7] Valor de transparencia (alpha) al dibujarse el fondo. Un valor de 1 es el valor normal; un valor de 0 es completamente transparente. Úsalo con cuidado. Dibujar un fondo parcialmente transparente toma mucho tiempo y puede hacer lento el juego.

32.4 Tiles

Como sabes, puedes agregar Tiles a los romos (cuartos). Un tile es una parte de un recurso de fondo 8background). Los tiles simplemente son imágenes visibles. No reaccionan a los eventos y no generan colisiones. Como resultado de esto, los tiles se manejan mucho más rápido que los objetos. Cualquier cosa que no requiera de una colisión es mejor que se maneje con Tiles. Además, es frecuente usar los tiles para un gráfico agradable mientras que el objeto es usado, por ejemplo, pars generar los eventos de colisión.

Actualmente, tú tienes más control sobre los tiles de lo que piensas. Puedes agregarlas cuando diseñas el room pero también puedes agregarlas mientras corres el juego. Puedes cambiar su posición e incluso escalarlas o hacerlas parcialmente transparentes. Un tile tiene las siguientes propiedades:

- **background.** El fondo del cual es tomado el tile.
- **left, top, width, height.** La parte del fondo que es usada.
- **x,y.** La posición de la esquina superior izquierda del tile en el room.
- **depth.** La profundidad del tile. Cuando diseñas un room sólo puedes indicar si usas tiles de fondo (con profundidad 1000000) o de primer plano (de profundidad –1000000) pero puedes seleccioner cualquier profundidad que quieras haciendo que los tiles aparezcan entre las instancias de los objetos.
- **visible.** Indica si el tile es visible.
- **xscale, yscale.** La escala con la que se dibuja el sprite. (El valor original es 1)
- **alpha.** El valor alpha indicando la transparencia del tile. 1 = no transparente. = = completamente transparente. Debes de tener cuidado porque dibujar tiles parcialmente transparentes es muy lento y puede ocasionar problemas en ciertos sistemas.

Para cambiar las propiedades de un tile en particular necesitas saber su Id. Cuando agregas tiles al crear un cuarto la id es mostrada in la barra de información en la parte

inferior. También hay una función para encontrar la id de un tile en una posición específica.

Las siguientes funciones existen para manejar los tiles:

tile_add(background,left,top,width,height,x,y,depth) Agrega un nuevo tile en el room con los valores indicados (Ve arriba para ver su significado). Esta función regresa la id del tile que puede ser usada después.
tile_delete(id) Elimina el tile con la id indicada.
tile_find(x,y,foreground) Regresa la id del tile en la posición (x,y). Cuanod no existe un tile en esa posición el valor –1 es regresado. Cuando el parámetro foreground es verdadero (true) sólo las tiles con profundidad menor a cero son regresados. De otra manera sólo los tiles con prufundidad mayor o igual es regresado. Cuando multiples tiles de fondo o primer plano existen en la posición dada, la primera es regresada.
tile_delete_at(x,y,foreground) Elimina el tile en la posición (x,y). Cuando el parámetro foreground es verdadero (true), solo los tiles con profundidad menor a 0 son eliminadas. De lo contrario, sólo los tiles con profundidad igual o mayor a 0 son eliminadas. Cuando múltiples tiles (de fondo o de primer plano) existen en la posición dada, todas son eliminadas.
tile_exists(id) Indica si el tile con el id indicado existe.

tile_get_x(id) Regresa la posición "x" del tile con el id indicado.
tile_get_y(id) Regresa la posición "y" del tile con el id indicado.
tile_get_left(id) Regresa el valor "left" del tile con el id indicado.
tile_get_top(id) Regresa el valor "top" del tile con el id indicado.
tile_get_width(id) Regresa el ancho del tile con el id indicado.
tile_get_height(id) Regresa la altura del tile con el id indicado.
tile_get_depth(id) Regresa la profundidad del tile con el id indicado.
tile_get_visible(id) Regresa el tile con el id indicado es visible o no.
tile_get_xscale(id) Regresa la escala x del tile con el id indicado.
tile_get_yscale(id) Regresa la escala y del tile con el id indicado.
tile_get_background(id) Regresa el fondo del tile con el id indicado.
tile_get_alpha(id) Regresa el valor alpha del tile con el id indicado.

tile_set_position(id,x,y) Asigna la posición del tile con el id indicado.
tile_set_region(id,left,right,width,height) Asigna la región del tile con el id indicado dentro de su fondo.
tile_set_background(id,background) Asigna el fondo para el tile con el id indicado.
tile_set_visible(id,visible) Asigna si el tile con el id idicado es visible o no.
tile_set_depth(id,depth) Asigna la profundidad del tile con el id indicado.
tile_set_scale(id,xscale,yscale) Asigna la escala del tile con el id indicado.

tile_set_alpha(id,alpha) Asigna el valor alfa del tile con el id indicado.

32.5 Funciones de dibujo

Es posible permitir que los objetos se vean diferentes con respecto a sus imágenes. Hay una colección completa de funciones disponibles para dibujar diferentes formas. También hay funciones para dibujar texto. Sólo puedes usar estas funciones en el evento drawing de un objeto. Estas funciones no sirven en ningún otro evento. Ten en cuenta que el hardware de gráficos de la computadora sólo dibuja las imágenes rápido. Así que cualquier otra rutina de dibujo va a ser relativamente lenta. También Game Maker está optimizado para dibujar imágenes. Así que evita usar estas rutinas de dibujo tanto como te sea posible. (Cuando sea posible, crea mejor un bitmap) También ten en cuenta que la colisión entre instancias se determina por sus sprites (o máscaras de colisión) y no por lo que actualmente se dibuja. Las siguientes funciones de dibujo existen:

draw_sprite(n,img,x,y) Dibuja la sub-imagen img (-1 = actual) del sprite con el índice n con su origen en la posición (x,y).
draw_sprite_scaled(n,img,x,y,s) Dibuja el sprite escalado con el factor s.
draw_sprite_stretched(n,img,x,y,w,h) Dibuja el esprite estirado de manera que llene la región con la esquina superior izquierda (x,y) y de ancho w y alto h.
draw_sprite_transparent(n,img,x,y,s,alpha) Dibuja el sprite escalado con factor s mezclado con su fondo. Alpha indica el valor de transparencia. Un valor de 0 hace el sprite completamente transparente. Un valor de 1 hace el sprite completamente sólido. Esta función puede crear grandes efectos (por ejemplo, explosiones parcialmente transparentes). Es lento porque se hace con software y se dede de usar con cuidado.
draw_sprite_tiled(n,img,x,y) Dibuja el sprite varias veces de manera que llene el cuarto entero. (x,y) es el lugar donde uno de los sprites es dibujado.

draw_background(n,x,y) Dibuja el fondo con índice n en la posición (x,y).
draw_background_scaled(n,x,y,s) Dibuja el background escalado.
draw_background_stretched(n,x,y,w,h) Dibuja el fondo ajustándolo a la región indicada.
draw_background_transparent(n,x,y,s,alpha) Dibuja el fondo con escala s y transparencia alpha (0-1). (¡Lento!).
draw_background_tiled(n,x,y) Dibuja el background varias veces hasta que llene el room completo.

The following drawing functions draw basic shapes. They use a number of properties, in particular the brush and pen color that can be set using certain variables.

draw_pixel(x,y) Draws a pixel at (x,y) in the brush color.
draw_getpixel(x,y) Returns the color of the pixel at (x,y).

draw_fill(x,y) Flood fill from position (x,y) in the brush color.
draw_line(x1,y1,x2,y2) Draws a line from (x1,y1) to (x2,y2).
draw_circle(x,y,r) Draws a circle at (x,y) with radius r.
draw_ellipse(x1,y1,x2,y2) Draws an ellipse.
draw_rectangle(x1,y1,x2,y2) Draws a rectangle.
draw_roundrect(x1,y1,x2,y2) Draws a rounded rectangle.
draw_triangle(x1,y1,x2,y2,x3,y3) Draws a triangle.
draw_arc(x1,y1,x2,y2,x3,y3,x4,y4) Draws an arc of an ellipse.
draw_chord(x1,y1,x2,y2,x3,y3,x4,y4) Draws a chord of an ellipse.
draw_pie(x1,y1,x2,y2,x3,y3,x4,y4) Draws a pie of an ellipse.
draw_button(x1,y1,x2,y2,up) Draws a button, up indicates whether up (1) or down (0).
draw_text(x,y,string) Draws the string at position (x,y). A # symbol or carriage return chr(13) or linefeed chr(10) are interpreted as newline characters. In this way you can draw multi-line texts. (Use \# to get the # symbol itself.)
draw_text_ext(x,y,string,sep,w) Similar to the previous routine but you can specify two more things. First of all, sep indicates the separation distance between the lines of text in a multiline text. Use -1 to get the default distance. Use w to indicate the width of the text in pixels. Lines that are longer than this width are split-up at spaces or – signs. Use -1 to not split up lines.
draw_text_sprite(x,y,string,sep,w,sprite,firstchar,scale) Drawing text using the functions above is relatively costly. This function works exactly the same as the previous one but takes its character images from a sprite. This sprite must have a subimage for each character. The first character is indicate with the argument firstchar. From this character on the characters should follow the ASCI order. You can check the character map tool of windows to see the correct order of the characters. If you only need the first few (e.g. up to the numbers or the uppercase characters) you don't need to provide the other characters. scale indicates the scale factor used (1 is the normal size). Be careful will scaling though, it can slow down the game considerably. Please realize that these sprites tend to be large. Also, you obviously don't need precise collision checking for them.
draw_polygon_begin() Start describing a polygon for drawing.
draw_polygon_vertex(x,y) Add vertex (x,y) to the polygon.
draw_polygon_end() End the description of the polygon. This function actually draws it.

You can change a number of settings, like the color of the lines (pen), region (brush) and font, and many other font properties. The effect of these variables is global! So if you change it in the drawing routine for one object it also applies to other objects being drawn later. You can also use these variables in other event. For example, if they don't change, you can set them once at the start of the game (which is a lot more efficient).

brush_color Color used to fill shapes. A whole range of predefined colors is available:

c_aqua

c_black
c_blue
c_dkgray
c_fuchsia
c_gray
c_green
c_lime
c_ltgray
c_maroon
c_navy
c_olive
c_purple
c_red
c_silver
c_teal
c_white
c_yellow

Other colors can be made using the routine **make_color(red,green,blue)**, where red, green and blue must be values between 0 and 255.

brush_style Current brush style used for filling. The following styles are available:

bs_hollow
bs_solid
bs_bdiagonal
bs_fdiagonal
bs_cross
bs_diagcross
bs_horizontal
bs_vertical

pen_color Color of the pen to draw boundaries.

pen_size Size of the pen in pixels.

font_color Color of the font to use.

font_size Size of the font to use (in points).

font_name Name of the font (a string).

font_style Style for the font. The following styles are available (you can add them if you want the combination of the styles):

fs_normal
fs_bold
fs_italic
fs_underline
fs_strikeout

font_angle Angle with which the font is rotated (0-360 degrees). For example, for vertical text use value 90.

font_align Alignment of the text w.r.t. the position given. The following values can be used

fa_left
fa_center
fa_right

A few miscellaneous functions exist:

string_width(string) Width of the string in the current font as it would drawn using the draw_text() function. Can be used for precisely positioning graphics.
string_height(string) Height of the string in the current font as it would drawn using the draw_text() function.
string_width_ext(string,sep,w) Width of the string in the current font as it would drawn using the draw_text_ext() function. Can be used for precisely positioning graphics.
string_height_ext(string,sep,w) Height of the string in the current font as it would drawn using the draw_text_ext() function.
screen_gamma(r,g,b) Sets the gamma correction values. r,g,b must be in the range from –1 to 1. The default is 0. When you use a value smaller than 0 that particular color becomes darker. If you use a value larger than 0 that color becomes lighter. Most of the time you will keep the three values the same. For example, to get the effect of lightning you can temporarily make the three values close to 1. This function works only in exclusive mode!
screen_save(fname) Saves a bmp image of the screen in the given filename. Useful for making screenshots.
screen_save_part(fname,left,top,right,bottom) Saves part of the screen in the given filename.

32.6 Views

As you should know you can define up to eight different views when designing rooms. In this way you can show different parts of the room at different places on the screen. Also, you can make sure that a particular object always stays visible. You can control the views from within code. You can make views visible and invisible and change the place or size of the views on the screen or the position of the view in the room (which is in particular useful when you indicated no object to be visible), you can change the size of the horizontal and vertical border around the visible object, and you can indicate which object must remain visible in the views. The latter is very important when the important object changes during the game. For example, you might change the main character object based on its current status. Unfortunately, this does mean that it is no longer the object that must remain visible. This can be remedied by one line of code in the creation event of all the possible main objects (assuming this must happen in the first view):

```
{
  view_object[0] = object_index;
}
```

The following variables exist that influence the view. All, except the first two are arrays ranging from 0 (the first view) to 7 (the last view).

view_enabled Whether views are enabled or not.

view_current* The currently drawn view (0-7). Use this only in the drawing event. You can for example check this variable to draw certain things in only one view. Variable cannot be changed.
view_visible[0..7] Whether the particular view is visible on the screen.
view_left[0..7] Left position of the view in the room.
view_top[0..7] Top position of the view in the room.
view_width[0..7] Width of the view (in pixels).
view_height[0..7] Height of the view (in pixels).
view_x[0..7] X-position of the view on the screen.
view_y[0..7] Y-position of the view on the screen.
view_hborder[0..7] Size of horizontal border around the visible object (in pixels).
view_vborder[0..7] Size of vertical border around visible object (in pixels).
view_hspeed[0..7] Maximal horizontal speed of the view.
view_vspeed[0..7] Maximal vertical speed of the view.
view_object[0..7] Object whose instance must remain visible in the view. If there are multiple instances of this object only the first one is followed. You can also assign an instance id to this variable. In that case the particular instance is followed.

Note that the size of the image on the screen is decided based on the visible views at the beginning of the room. If you change views during the game, they might no longer fit on the screen. The screen size though is not adapted automatically. So if you need this you have to do it yourself, using the following variables:

screen_width Width of the image on the screen, that is, the area in which we draw. When there are no views, this is the same as `room_width`.
screen_height Height of the image on the screen.

32.7 Transitions

As you know, when you move from one room to another you can indicate a transition. You can also set the transition for the next frame without moving to another room using the variable called `transition_kind`. If you assign a value between 1 and 17 to it the corresponding transition is used (these are the same transitions you can indicate for the rooms). A value of 0 indicates no transition. It only affects the next time a frame is drawn. You can also set these variables before going to the next room using code.

transition_kind Indicates the next frame transition (0-17).
transition_time Total time used for the transition (in milliseconds).
transition_steps Number of steps for the transition.

32.8 Repainting the screen

Normally at the end of each step the room is repainted on the screen. But in rare circumstances you need to repaint the room at other moments. This happens when your program takes over the control. For example, before sleeping a long time a repaint might

be wanted. Also, when your code displays a message and wants to wait for the player to press a key, you need a repaint in between. There are two different routines to do this.

`screen_redraw()` Redraws the room by calling all draw events.
`screen_refresh()` Refreshes the screen using the current room image (not performing drawing events).

To understand the second function, you will need to understand a bit better how drawing works internally. There is internally an image on which all drawing happens. This image is not visible on the screen. Only at the end of a step, after all drawing has taken place, the screen image is replaced by this internal image. (This is called double buffering.) The first function redraws the internal image and then refreshes the screen image. The second function only refreshes the image on the screen.

Now you should also realize why you couldn't use drawing actions or functions in other events than drawing events. They will draw things on the internal image but these won't be visible on the screen. And when the drawing events are performed, first the room background is drawn, erasing all you did draw on the internal image. But when you use `screen_refresh()` after your drawing, the updated image will become visible on the screen. So, for example, a script can draw some text on the screen, call the refresh function and then wait for the player to press a key, like in the following piece of code.

```
{
  draw_text(screen_width/2,100,'Press any key to continue.');
  screen_refresh();
  keyboard_wait();
}
```

Please realize that, when you draw in another event than the drawing event, you draw simply on the image, not in a view! So the coordinates you use are the same as if there are no views.

Be careful when using this technique. Make sure you understand it first and realize that refreshing the screen takes some time.

Capítulo 33 GML: Sonido y música

El sonido juega un papel crucial en los juegos de computadora. Hay dos diferentes tipos de sonidos: la música de fondo y los efectos. La música de fondo normalmente consiste de una pieza de música midi larga que se repite infinitamente. Por otro lado, los efectos de sonido son archivos wave cortos. Para tener efectos inmediatos, estas piezas son colodadas en la memoria. Por lo que asegúrate de que no sean demasiados largos.

Los sonidos se agregan a tu juego en la forma de recursos de sonido. Asegúrate de que los nombres que uses sean válidos. Hay un aspecto de los sonidos que puede ser confuso al principio, el número de buffers. El sistema puede reproducir un archivo wave solo una vez al mismo tiempo. Esto quiere decir que cuando uses el efecto nuevamente antes de que el sonido anterior haya terminado, el anterior es detenido. Esto no es muy atractivo. Por lo que cuando tengas un efecto de sonido que se emplee varias veces simultáneamente (como p. ej. un disparo de pistola) necesitas almacenarlo varias veces. Este número es el número de buffers. Entre más buffers tenga un sonido, más veces puede ser reproducido simultáneamente, pero también consume memoria. Así que usa esto con cuidado. El *Game Maker* usa automáticamente el primer buffer disponible, por lo que una vez que hayas indicado el número no debes preocuparte más por esto.

Hay cinco funciones básicas relacionadas con los sonidos, dos para reproducir un sonido, una para checar si un sonido se está reproduciendo, y dos para detener los sonidos. La mayoría reciben el índice del sonido como argumento. El nombre del sonido representa su índice. Pero también puedes almacenar el índice en una variable, y usar esta última.

`sound_play(index)` Reproduce una vez el sonido indicado.
`sound_loop(index)` Reproduce el sonido indicado, repitiéndolo continuamente.
`sound_stop(index)` Detiene el sonido indicado. Si hay varios sonidos con este índice reproduciéndose simultáneamente, todos serán detenidos.
`sound_stop_all()` Detiene todos los sonidos.
`sound_isplaying(index)` Indica si se está reproduciendo el sonido index.

Es possible emplear más efectos de sonido. Estos solo aplican a los archivos wave, no a los midi. Cuando desees usar efectos especiales de sonido, debes indicarlo en la ficha advanced de las propiedades del sonido seleccionando la opción adecuada. Nota que los sonidos que habiliten efectos consumen más recursos que los otros. Así que sólo usa esta opción cuando emplees las funciones siguientes. Hay tres tipos de efectos de sonido. Primero que nada puedes cambiar el volumen. Un valor de 0 indica que no hay sonido. Un valor de 1 es el volumen del sonido original. (No puedes indicar un volumen mayor que el original). Segundo, puedes cambiar el balance, esto es, la dirección de la que viene sl sonido. Un valor de 0 es completamente a la izquierda. Un valor de 1 indica completamente a la derecha. 0.5 es el valor por defecto que indica justo en el medio. Puedes usar el balance para p. ej. Escuchar que un objeto se mueve de izquierda a derecha. Finalmente puedes cambiar la frecuencia del sonido. Esto puede usarse p. ej. Para cambiar la velocidad de un motor. Un valor de 0 es la frecuencia más baja; un valor de 1 es la frecuencia más alta.

sound_volume(index,value) Cambia el volumen del sonido index (0 = bajo, 1 = alto).
sound_pan(index,value) Cambia el balance del sonido index (0 = izquierda, 1 = derecha).
sound_frequency(index,value) Cambia la frecuencia del sonido index (0 = baja, 1 = alta).

El sonido es un tema complicado. Los archivos midi son reproducidos usando el reproductor multimedia estándar. Sólo puede reproducirse un archivo midi a la vez y no hay soporte para efectos de sonido. Para los archivos wave el *Game Maker* usa DirectSound. En este caso todos los archivos wave son almacenados en la memoria y pueden tener efectos. De hecho, el *Game Maker* intenta reproducir otros archivos de música cuando los especificas, en particular archivos mp3. Para esto emplea el reproductor multimedia estándar. Pero ten cuidado. El que esto funcione depende del sistema y algunas veces de que otro software esté instalado o ejecutándose. Por lo que se recomiendoa no usar archivos mp3 cuando quieras distribuir tus juegos.

También hay varias funciones relacionadas con la reproducción de música desde un CD:

cd_init() Debe ser llamada antes de usar las otras funciones. Debiera llamarse también cuando se cambia el CD (o simplemente de vez en cuando).
cd_present() Indica si hay un CD en la unidad CD por defecto.
cd_number() Devuelve el número de pistas del CD.
cd_playing() Indica si el CD está reproduciéndose.
cd_paused() Indica si el CD está en pausa o detenido.
cd_track() Devuelve el número de la pista actual (1=la primera).
cd_length() Devuelve la duración total del CD en milisegundos.
cd_track_length(n) Devuelve la duración de la pista n del CD en milisegundos.
cd_position() Devuelve la posición actual del CD en milisegundos.
cd_track_position() Devuelve la posición actual de la pista en reproducción en milisegundos.
cd_play(first,last) Le indica al CD reproducir las pistas desde first hasta last. Si deseas reproducir el CD completo usa 1 y 1000 como argumentos.
cd_stop() Detiene la reproducción.
cd_pause() Pausa la reproducción.
cd_resume() Continua la reproducción.
cd_set_position(pos) Establece la posición del CD en milisegundos.
cd_set_track_position(pos) Establece la posición de la pista actual en milisegundos.
cd_open_door() Abre la bandeja del reproductor de CD.
cd_close_door() Cierra la bandeja del reproductor de CD.

Hay una función muy general para acceder a las funciones multimedia de Windows.

`MCI_command(str)` Esta función envía el commando str al sistema multimedia de Windows usando la Interfaz de Control de Medios (MCI – Media Control Interface). Devuelve la cadena return. Puedes usarla para controlar todo tipo de dispositivos multimedia. Lee la documentación de Windows para más información sobre cómo usar este commando. Por ejemplo `MCI_command('play cdaudio from 1')` reproduce un cd (después de que lo hayas inicializado correctamente usando otros comandos). ¡Esta función es solo para uso avanzado!

Capítulo 34 GML: Splash screens, highscores, and other pop-ups

En muchos juegos aparecen algunas ventanas emergentes. Estas ventanas muestran un video, una imagen o algo de texto. Son usadas frecuentemente al inicio del juego (como intro), al principio del nivel o al final del juego (por ej. los créditos). En Game Maker estas ventanas con texto, imágenes o videos pueden ser mostradas en cualquier momento del juego. El juego es pausado temporalmente mientras esta ventana es mostrada. Éstas son las funciones a utilizar:

`show_text(fname,full,backcol,delay)` Muestra una ventana con texto. fname es el nombre del archivo de texto. (.txt o .rtf). Debes poner este archivo dentro del mismo directorio en que está el juego. Además al crear la versión ejecutable (stand-alone) de tu juego, no debes olvidarte de añadir este archivo junto con el juego. full indica si se mostrará en pantalla completa o no. backcol es el color de fondo, y delay es un retraso en segundos antes de regresar el juego. (El jugador puede dar un clic con el ratón sobre la pantalla para regresar al juego.)

`show_image(fname,full,delay)` Muestra una imagen en una ventana emergente. fname es el nombre del archivo de imagen (solo archivos .bmp, .jpg, y wmf). Tienes que agregar este archivo en la carpeta donde está el juego. full indica si se va a mostrar en pantalla completa o no. delay es el retraso en segundos antes de regresar al juego.

`show_video(fname,full,loop)` Muestra un video en una ventana. fname es el nombre del archivo de video (.avi, .mpg). Debes poner este archivo en la misma carpeta donde esta el juego. full indica si se va a mostrar a pantalla completa. loop si se debe repetir al finalizar su reproducción.

`show_info()` Muestra la pantalla de información del juego (game information).

`load_info(fname)` Carga la información del juego desde un archivo indicado en fname. Este debe de ser un archivo rtf. Esto hace posible mostrar diferentes archivos de ayuda en diferentes momentos. Puedes usar el recurso "data file" para poner estos archivos dentro del juego.

También hay varias funciones para mostrar pequeñas ventanas con mensajes, preguntas, un menú con opciones o un dialogo en el cual el jugador puede introducir un número, una cadena de texto, un color o un nombre de algún archivo:

`show_message(str)` Muestra un cuadro de diálogo como un mensaje.

`show_message_ext(str,but1,but2,but3)` Muestra un cuadro de diálogo con el valor "str" como un mensaje, además de 3 botones. But1, but2 y but3 contienen el texto de cada botón. Si no escribes nada significará que el botón no se mostrará. En el texto puedes utilizar el símbolo & para indicar que el siguiente caracter debe ser usado como un acceso directo para este botón. Esta función regresa un valor de el botón presionado (0 si el usuario presiona la tecla Esc).

`show_question(str)` Mustra una pregunta; devuelve "true" cuando el usuario presiona yes o falso si no.

get_integer(str,def) Pide al jugador en un cuadro de diálogo por un número. Str es el mensaje. Def es número por defecto que se mostrará
message_background(back) Establece una imagen de fondo para el cuadro de diálogo para las funciones anteriores. back debe ser uno de los backgrounds definidos en el juego.
message_button(spr) Establece el sprite usado por los botones del cuadro de diálogo. spr debe ser un sprite que contenga tres imágenes, la primera para el botón cuando no esta presionado y el ratón no está sobre el botón, la segunda para cuando el ratón está sobre el botón pero no esta presionado y la tercera es para cuando es botón está presionado.
message_text_font(name,size,color,style) Establece la fuente usada en el cuadro de diálogo.
message_button_font(name,size,color,style) Establece la fuente usada en los botones del cuadro de diálogo.
message_input_font(name,size,color,style) Establece la fuente para el cuadro de texto (si el usuario necesita introducir algún valor) del cuadro de diálogo.
message_mouse_color(col) Establece el color de la fuente del cuadro e texto (si el usuario necesita introducir algún valor) del cuadro de diálogo cuando el ratón está sobre este.
message_input_color(col) Establece el color de fondo del cuadro e texto (si el usuario necesita introducir algún valor) del cuadro de diálogo.
message_caption(show,str) Establece el título del cuadro de diálogo. show indica si el borde debe ser mostrado (1) o no (0) y str indica el título cuando el borde es mostrado.
message_position(x,y) Establece la posición del cuadro e diálogo.
message_size(w,h) Fija el tamaño del cuadro de diálogo. Si indicas 0 para el width el ancho de la imagen es utilizado. Si escoges 0 para el height el alto es calculado en base al número de líneas del mensaje.
show_menu(str,def) Muestra un menú emergente. str indica el texto del menú. Esto consiste de los diferentes elementos del menú con una barra vertical entre ellos. Por ejemplo, str =‘menu0|menu1|meny2’. Cuando la primera opción es seleccionado el valor de 0 es regresado, etc. Cuando el jugador no selecciona ningún elemento, el valor por defecto def es regresado.
show_menu_pos(x,y,str,def) Muestra un menú desplegable como en la función anterior pero en la posición x,y de la pantalla.
get_color(defcol) Pide al jugador por un color. defcol es el color por defecto. Si el usuario presiona Cancel el valor -1 es retornado.
get_open_filename(filter,fname) Solicita al jugador por un nombre de archivo para abrir con un filtro dado. El filtro tiene la forma ‘name1|mask1|name2|mask21…’. Una máscara contiene las diferentes opciones con un punto y coma entre ellos. * significa cualquier cadena de texto. Ej. ‘bitmaps|*.bmp;*.wmf’. Si el usuario presiona Cancel una cadena de texto vacía es retornada.

`get_save_filename(filter,fname)` Pide al usuario un nombre de archivo para guardar con el filtro dado. Si el usuario presiona Cancel una cadena de texto vacía es retornada.

`get_directory(dname)` Pregunta por un directorio. dname es el nombre por defecto. Si el usuario presiona Cancel una cadena de texto vacía es retornada.

`get_directory_alt(capt,root)` Una manera diferente de pedir un directorio. caps es el título a ser mostrado. root es la ruta del directorio a ser mostrado.

Usa una cadena vacía y se mostrará el árbol completo. Si el usuario presiona Cancelar se regresa una cadena vacía.

`show_error(str,abort)` Despliega un mensaje de error estándar (y/o lo escribe en un archivo log). abort indica si el juego debe ser abortado.

Una ventana especial es la de highscore que es mantenida por cada juego. Estas son las funciones a utilizar:

`highscore_show(numb)` Muestra la table highscores. numb es el nuevo record. Si el puntaje es suficientemente bueno para ser añadido a la lista, el jugador podrá ingresar un nombre. Usa -1 para desplegar la lista actual.

`highscore_show_ext(numb,back,border,col1,col2,name,size)`
Muestra la tabla highscore. numb es el nuevo puntaje. Si el puntaje es suficientemente bueno para ser añadido a la lista, el jugador podrá ingresar un nombre. Usa -1 para desplegar la lista actual. back es la imagen de fondo a usar, border es si debe o no mostrarse un borde. coll es el color para el nuevo registro, col2 es el color de los otros puntajes. name es el nombre de la fuente a usar, y size es el tamaño de la fuente.

`highscore_clear()` Borra todos los puntajes.

`highscore_add(str,numb)` Añade un jugador de nombre str con el puntaje numb a la lista.

`highscore_add_current()` Añade el puntaje actual a la lista de highscores. Se le preguntará al jugador por un nombre.

`highscore_value(place)` Devuelve el puntaje de la persona en el lugar indicado (1-10). Esto puede ser usado para dibujar tu propia lista de puntajes más altos.

`highscore_name(place)` Devuelve el nombre de la persona en el lugar indicado (1-10).

`draw_highscore(x1,y1,x2,y2)` Muestra la tabla de puntajes más altos (highscores) en la ventana y con la fuente indicada.

Por favor recuerda que ninguna de estas ventanas pueden ser mostradas cuando el juego corre en modo exclusivo de gráficos!

Capítulo 35 GML: Resources

En *Game Maker* puedes especificar varios tipos de recursos, como sprites, sounds, data files, objects, etc. En este capítulo encontrarás un número de opciones que actúan sobre estos recursos. Antes de que empieces a usarlas, asegúrate de entender lo siguiente. En cualquier momento en que modifiques un recurso el original se pierde! Esto significa que el recurso es modificado para todas las instancias que lo usen. Por ejemplo, si cambias un sprite todas las instancias que usen este sprite lo tendrán modificado hasta que juego termine. Reiniciar la habitación o el juego no regresará este recurso a su forma original! Además cuando guardes el juego los recursos modificados NO son salvados. Si tú cargas un juego guardado es tu responsabilidad tener el recurso en el estado apropiado.

35.1 Sprites

Las siguientes funciones te darán información acerca de un sprite:

`sprite_exists(ind)` Devuelve si el sprite con el índice (ind) especificado existe.
`sprite_get_name(ind)` Devuelve el nombre del sprite con el ind especificado.
`sprite_get_number(ind)` Devuelve el número de subimágenes del sprite con el indice dado.
`sprite_get_width(ind)` Devuelve el ancho del sprite con el índice especificado.
`sprite_get_height(ind)` Devuelve la altura del sprite con el índice dado.
`sprite_get_transparent(ind)` Devuelve si el sprite con el índice especificado utiliza transparecia.
`sprite_get_xoffset(ind)` Devuelve el x-offset (punto de origen en x) del sprite con el índice especificado.
`sprite_get_yoffset(ind)` Devuelve el y-offset (punto de origen en y) del sprite con el índice especificado.
`sprite_get_bbox_left(ind)` Devuelve el valor del límite izquierdo del sprite (bounding box) con el índice especificado.
`sprite_get_bbox_right(ind)` Devuelve el valor del límite derecho del sprite (bounding box) con el índice especificado.
`sprite_get_bbox_top(ind)` Devuelve el valor del límite superior del sprite (bounding box) con el índice especificado.
`sprite_get_bbox_bottom(ind)` Devuelve el valor del límite inferior del sprite (bounding box) con el índice especificado.
`sprite_get_precise(ind)` Devuelve si el sprite con el índice dado utiliza la coalición precisa (precise collision checking).
`sprite_get_videomem(ind)` Devuelve si el sprite con el índice especificado utiliza la memoria de video.
`sprite_get_loadonuse(ind)` Devuelve si el sprite con el índice especificado es cargado sólo en uso (load only on use).

Los sprites usan mucha memoria. Para dibujarlos más rápido es importante almacenarlos en la memoria de video. Como está indicado en el capítulo 14 puedes indicar cuales sprites deben ser almacenados en la memoria de video. Además puedes indicar que ciertos sprites sólo deben ser cargados cuando se necesiten. Esos sprites serán descartados una vez que termine el nivel. Puedes controlar parcialmente este proceso con código. Estas son las funciones a usar:

sprite_discard(numb) Libera la memoria de video usada por el sprite. Si el sprite usa la propiedad de cargar sólo en uso este será removido completamente. De lo contrario, una copia es mantenida en la memoria normal (de la cual hay normalmente suficiente) de este modo el sprite puede ser restaurado si se ocupa.
sprite_restore(numb) Restaura al sprite en la memoria de video. Normalmente esto pasa automáticamente cuando el sprite es necesitado. Pero esto puede causar un pequeño problema, en particular cuando se usa cargar sólo en uso y el sprite es grande. Entonces tal vez quieras forzar esto por ejemplo en el inicio de la habitación en la cual el sprite es necesitado.
discard_all() Descarta todos los sprites, backgrounds y sounds que usan cargar sólo en uso.

Cuando un juego usa muchos sprites grandes diferentes, esto hace al archivo del juego grande y por lo tanto carga lento. Además, si quieres mantenerlos en la memoria de video mientras los necesites, esto incrementa el tamaño de la memoria requerida de manera considerable. Como alternativa, puedes distribuir las imágenes del sprite con el juego (como archivos .bmp, .jpg, o .gif; no se permiten otros formatos) y cargarlos durante el juego. Existen tres rutinas para ello:

sprite_add(fname,imgnumb,precise,transparent,videomem,loadonuse,xorig,yorig) Añade la imagen almacenada en el archivo fname al conjunto de recursos sprites. Sólo deben tratarse de imágenes bmp, jpg y gifs. Cuando la imagen es un bmp o jpg esta puede ser un strip conteniendo cierto número de subimágenes para el sprite enseguida del otro. Usa imgnumb para indicar su número (1 para una imagen simple). Para gifs animados, este argumento no es usado; el número de imágenes del gif es usado. precise indica si la coalición precisa debe ser usada. transparent indica si la imagen es particularmente transparente, videomem indica si el sprite debe ser almacenado en la memoria de video, y loadonuse indica si el sprite debe ser cargado sólo en uso. xorig y yorig indica la posición de origen en el sprite. La función devuelve el índice del nuevo sprite para que puedas usarlo para dibujar o asignarlo a la variable sprite_index de una instancia. Cuando ocurre un error -1 es devuelto.
sprite_replace(ind,fname,imgnumb,precise,transparent,videomem,loadonuse,xorig,yorig) Lo mismo que el anterior pero en este caso el sprite con indice ind es reemplazado. La función devuelve si esto fue logrado.
sprite_delete(ind) Elimina el sprite de la memoria, liberando la memoria usada. (Este ya no puede ser recuperado.)

ADVERTENCIA: Cuando salvas el juego en el transcurso del juego, sprites añadidos o reemplazados NO son almacenados junto con el juego salvado. Por lo tanto si cargas el juego después, no podrán estar ahí más. Además hay algunas cuestiones de derecho de copia con la distribución de archivos gifs con tu aplicación (comercial). Entonces es mejor no usar éstos.

35.2 Sounds

Las siguientes funciones te darán información acerca de los sonidos:

`sound_exists(ind)` Devuelve si un sonido con el índice dado existe.
`sound_get_name(ind)` Devuelve el nombre de el sonido con el índice dado.
`sound_get_kind(ind)` Devuelve el tipo de el sonido con el indice especificado (0=wave, 1=midi, 2=mp3, 10=unknown)
`sound_get_buffers(ind)` Devuelve el número de buffers de le sonido con un índice especificado.
`sound_get_effect(ind)` Devuelve si el sonido con el índice dado acepta efectos especiales.
`sound_get_loadonuse(ind)` Devuelve si el sonido con el índice dado usa cargar sólo en uso.

Los sonidos usan muchos recursos y la mayoría de los sistemas pueden almacenar y tocar un número limitado de sonidos. Si creas un juego largo te interesaría tener un mayor control sobre que sonidos son cargados en memoria y en que tiempo. Puedes usar la opción cargar sólo en uso (load on use) para los sonido para estar seguro que los sonidos son cargados sólo cuando se utilicen. Esto aunque puede tener un ligero problema al cargar el sonido. Además, esto no ayuda mucho cuando tienes una habitación grande. Para más control puedes usar los siguientes funciones:

`sound_discard(index)` Libera la memoria usada por el sonido indicado.
`sound_restore(index)` Restaura el sonido indicado en memoria.
`discard_all()` Descarta todos los sprites, backgrounds y sonidos que usan load-on-use (cargar en uso).

Cuando tu juego usa muchos sonidos complicados diferentes, por ejemplo, como música de fondo, es mejor no almacenarlos todos en dentro del juego. Esto hace el tamaño del archivo del juego muy grande. En cambio, es mejor proveerlos como archivos separados con el juego y cargarlos cuando sean necesitados. Esto puede reducir el tiempo de carga del juego. Las siguientes tres rutinas existen para ello:

`sound_add(fname,buffers,effects,loadonuse)` Añade un sonido a el juego. fname es el nombre del archivo de sonido. buffers indica el número de buffers a ser usados, y effects y loadonuse indican si los efectos de sonido son permitidos y si el sonido debe ser almacenado en la memoria interna (true o false). La función devuelve el índice del nuevo sonido, el cual puede ser usado para tocar un sonido. (-1 si un error ocurre, por ejemplo que el archivo no exista).

sound_replace(index,fname,buffers,effects,loadonuse) Lo mismo que el anterior pero esta vez no es creado un nuevo sonido sino que el sonido con índice (index) es reemplazado, liberando el sonido anterior. Devuelve si se logró.
sound_delete(index) Borra el sonido indicado, liberando la memoria asociada con este. Este no puede ser restaurado.

ADVERTENCIA: Cuando salvas el juego mientras juegas, sonidos añadidos o reemplazados NO son guardados junto con el archivo de guardado. Así que si cargas el juego después, esos sonidos no estarán ahí.

35.3 Backgrounds

Las siguientes funciones te darán información acerca de un background:

background_exists(ind) Devuelve si el background con el índice dado existe.
background_get_name(ind) Devuelve el nombre de el background con el índice indicado.
background_get_width(ind) Devuelve el ancho de el background con el índice indicado.
background_get_height(ind) Devuelve la altura de el background con el índice especificado.
background_get_transparent(ind) Devuelve si el backgrund con el índice indicado usa la memoria de video.
background_get_videomem(ind) Devuelve si el background con el índice dado es cargado sólo en uso.
background_get_loadonuse(ind) Devuelve si el background con el índice especificado es cargado sólo en uso.

Los backgrounds usan mucha memoria. Para dibujarlos más rápido es muy útil almacenarlos en la memoria de video. Como está indicado en el capítulo 18 puedes indicar cuales backgrounds deben ser almacenados en la memoria de video. Además puedes indicar que ciertos backgrounds sólo deben ser cargados cuando se necesiten. Esos backgrounds serán descartados una vez que termine el nivel. Puedes controlar parcialmente este proceso con código. Estas son las funciones a usar:

background_discard(numb) Libera la memoria de video usada por el background. Si el backgroune usa la propiedad de cargar sólo en uso este será removido completamente. De lo contrario, una copia es mantenida en la memoria normal (de la cual hay normalmente suficiente) de este modo el sprite puede ser restaurado si se ocupa.
background_restore(numb) Restaura al background en la memoria de video. Normalmente esto pasa automáticamente cuando el background es necesitado. Pero esto puede causar un pequeño problema, en particular cuando se usa cargar sólo en uso y la imagen es grande. Entonces tal vez quieras forzar esto por ejemplo en el inicio de la habitación en la cual el sprite es necesitado.

discard_all() Descarta todos los sprites, backgrounds y sounds que usan cargar sólo en uso.

Cuando un juego usa muchos backgrounds grandes diferentes, esto hace al archivo del juego grande y por lo tanto carga lento. Además, si quieres mantenerlos en la memoria de video mientras los necesites, esto incrementa el tamaño de la memoria requerida de manera considerable. Como alternativa, puedes distribuir las imágenes de los backgrounds con el juego (como archivos .bmp, .jpg, o .gif; no se permiten otros formatos) y cargarlos durante el juego. Existen tres funciones para ello. Otro uso es cuanto quieres permitir al jugador escoger una imagen de fondo. Además tal vez quieras guardar la imagen desde el juego y usarla después como un background (por ejemplo un programa para dibujar). Finalmente, backgrounds complejos,, guardados como jpg usan mucho menos memoria. Aquí están las funciones:

background_add(fname,transparent,videomem,loadonuse) Añade la imagen almacenada en el archivo fname al conjunto de recursos backgrounds. Sólo deben tratarse de imágenes bmp y jpg. transparent indica si la imagen es particularmente transparente, videomem indica si el background debe ser almacenado en la memoria de video, y loadonuse indica si el background debe ser cargado sólo en uso. La función devuelve el índice del nuevo background para que puedas usarlo para dibujar o asignarlo a la variable background_index[0] para hacerlo visible en el cuarto actual. Cuando ocurre un error -1 es devuelto.
background_replace(ind,fname,transparent,videomem,loadonuse) Lo mismo que el anterior pero en este caso el background con índice ind es reemplazado. La función devuelve si esto fue logrado. Cuando el background es actualmente visible en el cuarto también será reemplazado.
background_delete(ind) Elimina el background de la memoria, liberando la memoria usada. (Este ya no puede ser recuperado.)

ADVERTENCIA: Cuando salvas el juego en el transcurso del juego, backgrounds añadidos o reemplazados NO son almacenados junto con el juego salvado. Por lo tanto si cargas el juego después, no podrán estar ahí más. Además hay algunas cuestiones de derecho de copia con la distribución de archivos gifs con tu aplicación (comercial). Entonces es mejor no usar éstos.

35.4 Paths

Las siguientes opciones te darán información acerca de los paths:

path_exists(ind) Devuelve si el path con un indice dado existe.
path_get_name(ind) Devuelve el nombre del path con un nombre dado.
path_get_length(ind) Devuelve la longitud del path con el índice indicado.
path_get_kind(ind) Devuelve el tipo de conexiones del path con el índice especificado (0=recto, 1=curvo).

path_get_end(ind) Devuelve lo que pasa al finalizar del path con el índice señalado (0=detener, 1=brinca al inicio, 2=conecta al inicio, 3=regresa, 4=continua).

35.5 Scripts

Las siguientes opciones te darán información acerca de un script:

script_exists(ind) Returns Devuelve si un script con el índice indicado existe.
script_get_name(ind) Devuelve el nombre del script con el índice indicado.
script_get_text(ind) Devuelve la cadena de texto del script con el índice dado.

35.6 Data Files

Las siguientes opciones proporcionarán información acerca de un data file:

datafile_exists(ind) Devuelve si el data file con el índice dado existe.
datafile_get_name(ind) Devuelve el nombre del data file con un índice especificado.
datafile_get_filename(ind) Devuelve el nombre del archivo del data file con el índice especificado.

Las siguientes opciones pueden ser usadas si no exportaste el data file al inicio del juego.

datafile_export(ind,fname) Exporta el data file con el nombre de archivo señalado (fname) (si no lo exportaste por defecto al inicio)
datafile_discard(ind) Libera los datos almacenados internamente por el data file.

35.7 Objects

Las siguientes opciones proporcionarán información acerca de un objeto:

object_exists(ind) Devuelve si el objeto con el índice dado existe.
object_get_name(ind) Devuelve el nombre del objeto con el índice dado.
object_get_sprite(ind) Devuelve el índice del sprite por defecto del objeto con el índice epecificado.
object_get_solid(ind) Devuelve si el objeto con el índice dado es sólido por defecto.
object_get_visible(ind) **D**evuelve si el objeto con el índice dado es visible por defecto.
object_get_depth(ind) Devuelve la profundidad del objeto con el índice dado.

`object_get_persistent(ind)` Devuelve si el objeto con el índice señalado es persistente.
`object_get_mask(ind)` Devuelve el índice de la máscara del objeto con el índice dado (-1 si no tiene máscara especial).
`object_get_parent(ind)` Devuelve el índice del objeto pariente del objeto ind (-1 si no tiene pariente).
`object_is_ancestor(ind1,ind2)` Devuelve si el objeto ind2 es un progenitor del objeto ind1.

35.8 Rooms

Las siguientes funciones darán información acerca de una habitación:

`room_exists(ind)` Devuelve si el cuarto con el índice señalado existe.
`room_get_name(ind)` Devuelve el nombre de la habitación con el índice dado.

Nota que porque las habitaciones cambian durante el juego hay otras rutinas para cambiar los contenidos de la habitación actual.

Capítulo 36 GML: Archivos, registro y ejecución de programas

En juegos más avanzados probablemente querrás leer datos de un archivo que haz incluido con el juego. Por ejemplo, pudieras crear un archivo que indique en qué momento deben ocurrir ciertas cosas. Probablemente también quieras guardar información para la siguiente vez que se ejecute el juego (por ejemplo, el cuarto actual). Para esto tenemos las siguientes funciones:

`file_exists(fname)` Indica si el archivo con el nombre fname existe (true) o no (false).

`file_delete(fname)` Borra el archivo con el nombre fname.

`file_rename(oldname,newname)` Renombra el archivo con el nombre oldname a newname.

`file_copy(fname,newname)` Copia el archivo fname al nombre newname.

`file_open_read(fname)` Abre el archivo fname para lectura.

`file_open_write(fname)` Abre el archivo fname para escritura, creándolo si no existe.

`file_open_append(fname)` Abre el archivo fname para agregar datos al final, creándolo si no existe.

`file_close()` Cierra el archivo actual (¡No olvides llamarla!).

`file_write_string(str)` Escribe la cadena str al archivo abierto actualmente..

`file_write_real(x)` Escribe el valor real x en el archivo abierto actualmente.

`file_writeln()` Escribe un caracter de nueva línea en el archivo.

`file_read_string()` Lee una cadena del archivo y devuelve esta cadena. Una cadena termina al final de la línea.

`file_read_real()` Lee un valor real del archivo y devuelve este valor.

`file_readln()` Salta el resto de la línea en el archivo e inicia al principio de la siguiente línea.

`file_eof()` Indica si hemos llegado al final del archivo.

`directory_exists(dname)` Indica si la carpeta dname existe.

`directory_create(dname)` Crea una carpeta con el nombre dname (incluyendo la ruta a esa carpeta) si no existe.

`file_find_first(mask,attr)` Devuelve el nombre del primer archivo que satisfaga las condiciones de la máscara mask y los atributos attr. Si no existe tal archivo, devuelve una cadena vacía. La máscara puede contener una ruta y wildchars, por ejemplo 'C:\temp*.doc'. Los atributos indican archivos adicionales que quieras ver. (Por lo que los archivos normales son siempre devueltos cuando satisfacen la máscara). Puedes agregar las siguientes constants para ver el tipo de archivos que desees:

- **`fa_readonly`** archivos de sólolectura
- **`fa_hidden`** archivos ocultos
- **`fa_sysfile`** archivos de sistema
- **`fa_volumeid`** archivos volume-id

fa_directory carpetas
fa_archive archivos archivados

file_find_next() Devuelve el nombre del siguiente archivo que satisface la máscara y los atributos indicados previamente. Si no existe tal archivo, devuelve una cadena vacía.
file_find_close() Debe ser llamada después de manipular los archivos para liberar la memoria.
file_attributes(fname,attr) Indica si el archivo fname tiene todos los atributos dados por attr. Usa una combinación de las constanets indicadas anteriormente.

Si el jugador ha seleccionado modo seguro en sus preferencias, para ciertas rutinas, no se permite especificar la ruta, y sólo puedes acceder a los archivos en la carpeta de la aplicación p. ej. para escribir en ellos.

Las siguientes tres variables de sólo lectura pueden ser útiles:

game_id* Identificador único para el juego. Puedes usarlo si necesitas un nombre único de archivo.
working_directory* Carpeta de trabajo del juego. (No incluye la diagonal invertida final).
temp_directory* Carpeta temporal creada para el juego. Puedes almacenar archivos temporales aquí. Serán eliminados cuando el juego finalice.

En ciertas situaciones podrías dar al jugador la posibilidad de introducir argumentos mediante la línea de commandos al juego que están ejecutando (para por ejemplo crear cheats o modos especiales). Para obtener estos argumentos puedes usar las siguientes dos rutinas.

parameter_count() Devuelve el número de parámetros de la línea de commandos (nota que el nombre del programa es uno de ellos).
parameter_string(n) Devuelve los parámetros n de la línea de commandos. El primer parámetro tiene índice 0. Se trata del nombre del programa.

Si deseas almacenar una pequeña cantidad de información entre cada ejecución del juego hay un mecanismo más simple que el emplear un archivo. Puedes usar el registro. El registro es una gran base de datos de Windows para mantener todo tipo de configuraciones para los programas. Una entrada tiene un nombre, y un valor. Puedes usar tanto cadenas como valores reales. Tenemos las siguientes funciones:

registry_write_string(name,str) Crea una entrada en el registro con el nombre name y como valor la cadena str.
registry_write_real(name,x) Crea una entrada en el registro con el nombre name y el valor real x.

registry_read_string(name) Devuelve la cadena almacenada con el nombre name. (El nombre debe existir, de otra forma se devuelve una cadena vacía).
registry_read_real(name) Devuelve el valor real almacenado en la entrada name. (El nombre debe existir, de otra forma se devuelve el número 0).
registry_exists(name) Indica si la entrada name existe.

En realidad, los valores en el registro están agrupados en claves. Las rutinas anteriores trabajan con valores dentro de la clave que está creada especialmente para tu juego. Tu programa puede usar esto para obtener cierta información sobre el sistema en el que se está ejecutando el juego. También puedes leer valores en otras claves. Puedes escribir valores en ellas pero ten mucho cuidado. PUEDES FÁCILMENTE DESTRUIR TU SISTEMA de esta forma. (La escritura no se permite en modo seguro). Nota que las claves también están colocadas en grupos. Las siguientes rutinas trabajan por defecto con el grupo HKEY_CURRENT_USER. Pero puedes cambiar el grupo raíz. Así, por ejemplo, si quisieras encontrar la carpeta temporal actual, usa Actually, values in the registry are grouped into keys. The above routines all work on values within the key that is especially created for your game. Your program can use this to obtain certain information about the system the game is running on. You can also read values in other keys. You can write them also but be very careful. YOU EASILY DESTROY YOUR SYSTEM this way. (Write is not allowed in secure mode.) Note that keys are again placed in groups. The following routines default work on the group HKEY_CURRENT_USER. But you can change the root group. So, for example, if you want to find out the current temp dir, use

```
path = registry_read_string_ext('/Environment','TEMP');
```

Tenemos las siguientes funciones.

registry_write_string_ext(key,name,str) Crea una entrada en el registro dentro de la clave key con el nombre name y como valor la cadena str.
registry_write_real_ext(key,name,x) Crea una entrada en el registro dentro de la clave key con el nombre name y el valor real x.
registry_read_string_ext(key,name) Devuelve la cadena con el nombre name dentro de la clave key. (El nombre debe existir, de otra forma se devuelve una cadena vacía).
registry_read_real_ext(key,name) Devuelve el valor real con el nombre name en la clave key. (El nombre debe existir, de otra forma se devuelve el número 0).
registry_exists_ext(key,name) Indica si el nombre name existe dentro de la clave key.
registry_set_root(root) Configura la raíz para las otras rutinas. Usa los siguientes valores:

0 = HKEY_CURRENT_USER
1 = HKEY_LOCAL_MACHINE
2 = HKEY_CLASSES_ROOT

3 = HKEY_USERS

El *Game Maker* también tiene la posibilidad de iniciar programas externos. Hay dos funciones disponibles para esto: execute_program y execute_shell. La función execute_program inicia un programa, posiblemente con algunos argumentos. Puede esperar hasta que el programa termine (pausando el juego) o continuar con el juego. La función execute_shell abre un archivo. Puede ser cualquier archivo para cuyo tipo esté definida una asociación, p. ej. un archivo html, un archivo de Word, etc. O puede ser un programa. No puede esperar a que se termine la ejecución por lo que el juego continúa.

`execute_program(prog,arg,wait)` Exeuta el programa prog con los argumentos arg. wait indica si se debe esperar a que termine la aplicación.

`execute_shell(prog,arg)` Ejecuta el programa (o archivo) en el entorno.

Ambas funciones no funcionarán si el jugador activa el modo seguro en sus preferencias. Puedes checar esto usando la variable de solo lectura:

`secure_mode*` Indica si el juego está ejecutándose en modo seguro.

Capítulo 37 GML: Juegos multiplayer

Jugar contra la computadora es divertido. Pero jugar contra oponentes humanos puede serlo mucho más. Es relativamente fácil crear este tipo de juegos porque no tienes que implementar IA compleja para un oponente manejado por la computadora. Por supuesto que puedes sentar a dos jugadores tras el mismo monitor y emplear diferentes teclas o algún otro dispositivo de entrada, pero es mucho más interesante cuando cada jugador está detrás de su propia computadora. O aún mejor, un jugador se encuentra del otro lado del océano. El *Game Maker* tiene soporte multijugador. Por favor toma en cuenta que el crear juegos multijugador efectivos que se sincronicen bien y no tengan latencia es una tarea difícil. Este capítulo da una breve descripción de las posibilidades. En el sitio web hay un tutorial con más información.

37.1 Estableciendo una conexión

Para que dos computadoras se comuniquen necesitarán sierto protocolo de conexión. Como la mayoría de los juegos, el *Game Maker* ofrece cuatro diferentes tipos de conexión: IPX, TCP/IP, Módem y Serial. La conexión IPX (para ser más precisos, se trata de un protocolo) funciona casi completamente transparente. Puede emplearse para jugar juegos con otras personas en la misma red local. Se necesita que esté instalada en tu computadora para que pueda emplearse. (Si no funciona, consulta la documentación de Windows. O ve a la opción Red dentro del Panel de Control y agrega el protocolo IPX). TCP/IP es el protocolo de internet. Puede usarse para jugar con otros jugadores en cualquier lugar mediante internet, suponiendo que conozcas su dirección IP. Una conexión modem se realiza a través del modem. Tienes que introducir cierta información del modem (una caden de inicialización y un número telefónico) para usarla. Finalmente, al usar la línea serial (una conexión directa entre las computadoras) necesitas introducir varias opciones de puertos. Hay cuatro funciones dentro del GML que pueden emplearse para inicializar estas conexiones:

`mplay_init_ipx()` inicializa una conexión IPX.

`mplay_init_tcpip(addr)` inicializa una conexión TCP/IP. `addr` es una cadena que indica la dirección web o IP, p. ej. `' www.gameplay.com'` o `' 123.123.123.12'`, posiblemente seguida por un número de puerto (p. ej. `' :12'`). Sólo se necesita la dirección para unirse a una sesión (ve a continuación). En una red local no se necesitan direcciones.

`mplay_init_modem(initstr,phonenr)` inicializa una conexión via módem. `initstr` es la cadena de inicialización para el módem (puede estar vacía). `phonenr` es una cadena que contiene el número telefónico a marcar (p. ej. `' 0201234567'`). Sólo se necesita el número telefónico al unirse a una sesión (ve a continuación).

`mplay_init_serial(portno,baudrate,stopbits,parity,flow)` inicializa una conexión serial. `portno` es el número del puerto (1-4). `baudrate` es la velocidad a emplear en baudios (100-256K). `stopbits` indica el número de bits de parada (0 = 1 bit, 1 = 1.5 bit, 2 = 2 bits). `parity` incida la paridad (0=ninguna, 1=impar, 2=par, 3=mark). Y `flow` indica el tipo de control de flujo (0=ninguno,

1=xon/xoff, 2=rts, 3=dtr, 4=rts y dtr). Indica si la conexión se ha establecido. Da un valor de 0 como el primer argumento para abrir mostrar un diálogo en el que el usuario puede cambiar la configuración.

Tu juego debe llamar una de estas funciones una sola vez. Todas las funciones indican si han tenido éxito (si se logra la conexión). No tiene éxito si el protocolo en particular no está instalado o soportado por tu máquina. Para checar si hay una conexión exitosa disponible puedes emplear la siguiente función

`mplay_connect_status()` devuelve el estado de la conexión actual. 0 = no hay conexión, 1 = conexión IPX, 2 = conexión TCP/IP, 3 = conexión via módem, and 4 = conexión serial.

Para finalizar la conexión llama

`mplay_end()` finaliza la conexión actual.

Cuando empleas una conexión TCP/IP tal vez quieras decirle a la persona con la que deseas jugar cuál es la dirección ip de tu computadora. La siguiente función te será de ayuda:

`mplay_ipaddress()` devuelve la dirección IP de tu máquina (p. ej. ' 123.123.123.12') como cadena. Puedes p. ej. mostrarla en pantalla. Notque esta rutina es lenta por lo que mejor no la llames a menudo.

37.2 Creando y uniéndose a sesiones

Cuando te conectas a una red, puede haber múltiples juegos ejecutándose en la misma red. Esos juegos reciben el nombre de sesiones. Estas diferentes sesiones pueden corresponder a juegos diferentes o al mismo. Un juego debe identificarse en la red. Afortunadamente, el *Game Maker* hace esto por ti. Lo único que debes saber es que cuanto cambias el id del juego en la ventana de opciones esta identificación cambia. De esta forma puedes evitar que personas con versiones anteriores de tu juego jueguen contra personas que cuentan con versiones más recientes. diferentes juegos o al When you connect to a network, there can be multiple games happening on the same network. We call these sessions. These different sessions can correspond to different games or to the same game. A game must uniquely identify itself on the network. Fortunately, *Game Maker* does this for you. The only thing you have to know is that when you change the game id in the options form this identification changes. In this way you can avoid that people with old versions of your game will play against people with new versions.

Si quieres iniciar un nuevo juego multijugador necesitas crear una nueva sesión. Para esto puedes emplear la siguiente rutina:

`mplay_session_create(sesname,playnumb,playername)` Crea una nueva mieva sesión en la conexión actual. `sesname` es una cadena que indica el nombre de la sesión. `playnumb` indica el número máximo de jugadores

permitidos para este juego (usa 0 para un número arbitrario de jugadores). `playname` es tu nombre como jugador. Indica si ha tenido éxito.

Una instancia del juego debe crear la sesión. La(s) otra(s) instancia(s) del juego deben unirse a esta sesión. Esto es un poco más complicado. Primero debes ver las sesiones disponibles y luego elegir a la que te unirás. Hay tres rutinas importantes para esto:

`mplay_session_find()` busca todaslas sesiones que aún aceptan jugadores y devuelve el número de sesiones encontradas.
`mplay_session_name(numb)` devuelve el nombre de la sesión número `numb` (0 es la primer sesión). Esta rutina puede ser llamada sólo después de haber llamado a la anterior.
`mplay_session_join(numb,playername)` con esta rutina te unes a la sesión número `numb` (0 es la primer sesión). `playername` es tu nombre como jutador. Indica si ha tenido éxito.

Hay una rutina más que puede cambiar el modo de la sesión. Debe llamarse antes de crear una sesión:

`mplay_session_mode(move)` indica si se mueve la sesión de host a otra computadora cuando el host actual cierre. `move` debe ser true o false (valor por defecto).

Para checar el estado de la sesión actual puedes usar la siguiente función

`mplay_session_status()` devuelve el estado de la sesión actual. 0 = no hay sesión, 1 = sesión creada, 2 = se unió a la sesión.

Un jugador puede detener una sesión empleando la siguiente rutina:

`mplay_session_end()` finaliza la sesión para este jugador.

37.3 Jugadores

Cada instancia del juego pque se una a un juego es un jugador. Como se indicó antes, los jugadores tienen nombres. Hay tres rutinas referentes a los jugadores.

`mplay_player_find()` busca todos los jugadores en la sesión actual y devuelve el número de jugadores encontrados.
`mplay_player_name(numb)` devuelve el nombre del jugador número `numb` (0 es el primer jugador, el cual siempre eres tú). Esta rutina puede sólo ser llamada después de haber llamado a la anterior.
`mplay_player_id(numb)` devuelve el id único del jugador número `numb` (0 es el primer jugador, el cual siempre eres tú). Esta rutina puede llamarse sólo después de haber llamado la primera. Este id es usado al enviar y recibir mensajes de otros jugadores.

37.4 Shared data (datos compartidos)

La comunicación mediante datos compartidos es probablemente la mejor forma de sincronizar el juego. Todo el proceso de comunicación no es visible para ti. Hay un juego de 10000 valores que son comunes a todas las entidades del juego. Cada entidad puede establecer y ller valores. El *Game Maker* se asegura de que cada entidad vea los mismos valores. Un valor puede ser un número real o una cadena. Sólo hay dos rutinas:

mplay_data_write(ind,val) escribe el valor `val` (cadena o real) en la ubicación `ind` (`ind` between 0 and 10000).
mplay_data_read(ind) devuelve el valor en la ubicación `ind` (`ind` entre 0 y 10000). Inicialmente todos los valores son 0.

Para sincronizar la información en las diferentes máquinas puedes ya sea usar un modo garantizado (guaranteed mode) que asegura que el cambio llegue a la otra máquina (pero el cual es lento) o un modo no garantizado (non-guaranteed mode). Para cambiar esto usa la siguiente rutina:

mplay_data_mode(guar) indica si se usa o no transmisión garantizada para los datos compartidos. `guar` debe ser true (valor por defecto) o false.

37.5 Mensajes

El segundo mecanismo de comunicación que el *Game Maker* soporta es el envio y la recepción de mensajes. Un jugador puede enviar mensajes a un jugador o a todos los jugadores. Los jugadores pueden ver si han llegado mensajes y llevar a cabo las acciones adecuadas. Los mensajes pueden enviarse en modo garantizado en el que estás seguro de que llegarán (pero puede ser lento) o en modo no garantizado, el cual es más rápido.

Tenemos las siguientes rutinas de mensajes:

mplay_message_send(player,id,val) envia un mensaje al jugador player (ya sea un identificador o un nombre; usa 0 para enviar el mensaje a todos los jugadores). `id` es un entero que funciona como identificador del mensaje y `val` es el valor (ya sea un número real o una cadena). El mensaje es enviado en modo no garantizado.
mplay_message_send_guaranteed(player,id,val) envia un mensaje al jugador player (ya sea un identificador o un nombre; usa 0 para enviar el mensaje a todos los jugadores). `id` es un entero que funciona como identificador del mensaje y `val` es el valor (ya sea un número real o una cadena). Este es un envio garantizado.
mplay_message_receive(player) recibe el siguiente mensaje de la cola de mensajes que llegó del jugador player (identificador o nombre). Usa 0 para indicar mensajes de cualquier jugador. La rutina indica si de hecho hubo un nuevo mensaje. Si es así, puedes emplear las siguientes rutinas para obtener su contenido:

`mplay_message_id()` Devuelve el identificador del último mensaje recibido.
`mplay_message_value()` Devuelve el valor del último mensaje recibido.
`mplay_message_player()` Devuelve el jugador que envió el último mensaje recibido.
`mplay_message_name()` Devuelve el nombre del jugador que envió el último mensaje recibido.
`mplay_message_count(player)` Devuelve el número de mensajes restantes en la cola de espera del jugador player (usa 0 para contar todos los mensajes).
`mplay_message_clear(player)` Elimina todos los mensajes en la cola de espera del jugador player (usa 0 para eliminar todos los mensajes).

Debemos hacer algunas observaciones. Primero que nada, si quieres enviar un mensaje sólo a un usuario en particular, necesitarás conocer el id del jugador. Como se indicó antes puedes obtener este id con la función `mplay_player_id()`. Este identificador del jugador también es empleado al recibir mensajes de un jugador en particular. Alternativamente, puedes dar el nombre del jugador como cadena. Si varios jugadores tienen el mismo nombre, el mensaje sólo llegará al primero.

En segundo lugar, podrías preguntarte porqué cada mensaje tiene un entero identificador. La razón es que esto ayuda a tu aplicación a enviar diferentes tipos de mensajes. El receptor puede checar el tipo de mensaje usando el id y llevar a cabo las acciones apropiadas. (Como no está garantizado que los mensajes lleguen, el enviar el id y el valor en mensajes diferentes causaría serios problemas).

Capítulo 38 GML: Usando DLLs

En aquellos casos en los que la funcionalidad del GML no cubre lo que deseas, puedes extender las posibilidades empleando plug-ins. Un plug-in consiste en un archivo DLL (Dynamic Link Library – Biblioteca de enlace dinámico). En este archivo DLL puedes definir unciones. Tales funciones pueden ser programadas en cualquier lenguaje que soporte la creación de DLLs (p. ej. Delphi, Visual C++, etc.). Aunque necesitarás de cierta habilidad en programación para hacerlo. Las funciones plug-in deben tener un formato específico. Pueden tener entre 0 y 12 argumentos, cada uno de los cuales puede ser un número real (double en C) o una cadena (string) terminada en un carácter nulo. (Para más de 4 argumentos, sólo existe soporte para argumentos reales por el momento). Estas funciones deben devolver ya sea un valor real o una cadena terminada en carácter nulo.

En Delphi puedes crear una DLL haciendo clic en **New** del menú **File** y luego seleccionando DLL. Aquí tienes un ejemplo de una DLL que podrías emplear con el *Game* Maker, escrita en Delphi. (Nota: ¡este es código de Delphi, no GML!)

```
library MyDLL;

uses SysUtils, Classes;

function MyMin(x,y:real):real;  cdecl;
begin
  if x<y then Result := x else Result := y;
end;

var res : array[0..1024] of char;

function DoubleString(str:PChar):PChar; cdecl;
begin
  StrCopy(res,str);
  StrCat(res,str);
  Result := res;
end;

exports MyMin, DoubleString;

begin
end.
```

Esta DLL define dos funciones: `MyMin` toma dos argumentos reales y devuelve el de menor valor, y `DoubleString` que duplica una cadena. Recuerda que debes ser cuidadoso con el manejo de la memoria. Esta es la razón por la que declaré la cadena resultante como global. También nota el uso de la llamada cdecl por conveción. Puedes emplear las convenciones de llamada cdecl o stdcall. Una vez que crees la DLL en Delphi tendrás el archivo `MyDLL.DLL`. Este archivo debe ser colocado en la carpeta de tu juego. (O en cualquier otro lugar donde Windows pueda encontrarlo). También puedes usar un recurso data file para almacenar la DLL dentro del juego.

Para hacer uso de esta DLL en el *Game Maker* primero necesitas especificar las funciones externas que deseas emplear y qué tipo de argumentos necesitan. Para ello tenemos la siguiente función en el GML:

> **`external_define(dll,name,calltype,restype,argnumb,arg1type,arg2type, …)`** Define una función externa. `dll` es el nombre del archivo dll. `name` es el nombre de las funciones. `calltype` es la convención de llamada empleada. Usa `dll_cdecl` o `dll_stdcall`. `restype` es el tipo del resultado. Usa `ty_real` o `ty_string`. `argnumb` es el número de argumentos (0-12). Después, para cada argumento debes especificar su tipo. Para ello usa nuevamente `ty_real` o `ty_string`. Cuando hay más de 4 argumentos todos ellos deben ser de tipo `ty_real`.

Esta función devuelve el id de la función externa que debe emplearse para llamarla. En el ejemplo de arriba, al inicio del juego usarías el siguiente código GML:

```
{
  global.mmm = external_define('MYDLL.DLL','MyMin',dll_cdecl,
                                    ty_real,2,ty_real,ty_real);
  global.ddd = external_define('MYDLL.DLL','DoubleString',dll_cdecl,
                                    ty_string,1,ty_string);
}
```

Ahora en el momento que neceisties llamar a las funciones, usas la siguiente función:

> **`external_call(id,arg1,arg2,…)`** Llama a a la función externa con el id y los argumentos dados. Necesitas proporcionar el número correcto de argumentos del tipo correcto (real o string). La función devuelve el resultado de la función externa.

Así, por ejemplo, escribirías:

```
{
  aaa = external_call(global.mmm,x,y);
  sss = external_call(global.ddd,'Hello');
}
```

Ahora te preguntarás cómo hacer una función en una DLL que haga algo en el juego. Por ejemplo, podrías querer una DLL que agregue instancias de objetos a tu juego. La forma más fácil es dejar que la función DLL devuelva una cadena que contenga una pieza de código GML. Esta cadena que contiene una pieza de GML puede ser ejecutada usando la función del GML

> **`execute_string(str)`** Ejecuta la pieza de código en la cadena str.

Alternativamente puedes dejar que la DLL cree un archivo con un script que puede ser ejecutado (esta función también puede ser empleada para luego modificar el comportamiento del juego).

`execute_file(fname)` Ejecuta el código en el archivo fname.

Ahora puedes llamar una función externa y ejecutar la cadena resultante, por ejemplo:

```
{
  ccc = external_call(global.ddd,x,y);
  execute_string(ccc);
}
```

En algunos casos especiales pudieras necesitar el handle de ventana (identificador de Windows) de la ventana principal del juego. Este puede obtenerse con la siguiente función y luego ser pasado a la DLL:

`window_handle()` Devuelve el handle de ventana de la ventana principal.

Nota: las DLLs no pueden ser empleadas en modo seguro.

El empleo de DLLs externas es una función extremadamente poderosa. Pero por favor haz uso de ellas sólo si sabes lo que estás haciendo.