PROYECTO ARKANOID

Grzegorz Adam Hankiewicz (gradha@iname.com)
Carlos Pereda Moreno (peremore@euskalnet.net)

25 de junio de 2000

Introducción

Este es el resultado de un trabajo presentado para la asignatura de C de segundo de Informática de Gestión en la universidad de Deusto. Se trata de un proyecto especial porque no entraba dentro de la oferta "estándar" y se deseaba realizar algo fuera de lo normal, en nuestro caso, que el programa fuese portable a varias plataformas diferentes. Fue presentado en enero del 2000.

El juego en sí no es apenas comparable con muchos juegos gratuíto existentes. Sin embargo, bajo criterio del profesor se merece la calificación de matrícula de honor. Nos enorgullece haber recibido tal calificación, y con el espíritu de compartir nuestro conocimiento, ofrecemos el trabajo al público para que pueda ser analizado, estudiado y por supuesto modificado.

El trabajo original se presentó con la documentación impresa y el código fuente más compiladores y librerías necesarias en un CDROM, dado que el conjunto superaba las 16 megas y no era conveniente presentarlo en disquetes. Obviamente en el CDROM sólo se incluían aquellas herramientas libres, como el DJGPP. Esta versión electrónica del trabajo es casi idéntica al original, con la diferencia de que la documentación está escrita en LaTeXy no se tiene el CDROM original. Al final de la introducción se muestra un listado que relaciona las páginas web de donde se puede conseguir cada fichero, para reproducir prácticamente el CDROM presentado.

A día de hoy vemos muchas cosas que se pueden mejorar, pero ya no vamos a dedicar tiempo a eso. Una de las mejoras más notables sería reescribir la rutina de detección de colisiones de la pelota contra el resto de los objetos. A pesar de que las rutinas actuales funcionan *más o menos*, a veces fallan y la pelota puede salir por ejemplo por un borde de la pantalla. Esto se puede hacer fácilmente mediante vectores, como se explica por ejemplo en el tutorial de colisiones que está en la sección El Rincón Gráfico de http://www.terradixital.com/gamedev/tutoriales.htm.

Se debería poder acceder a esta versión electrónica del trabajo desde *Zyphro* $\Longrightarrow$ *Programas* $\Longrightarrow$ *Arkanoid* en http://gradha.infierno.org, página web de uno de los autores. Ahí se podrá encontrar el código fuente del juego y de esta documentación. Tanto el juego como la documentación se distribuyen con la licencia "gift-ware":

Este trabajo es "gift-ware". Fue creado por dos personas trabajando en cooperación, y se le da como un regalo. Puede usarlo, modificarlo, redistribuirlo, y en general hackearlo de cualquier forma, y no devolvernos nada a cambio. Si redistribuye partes de este código o hace un juego con él, sería muy amable por su parte que nos mencione en alguna parte de sus créditos, pero no se le obliga hacerlo. Confiamos en que no abuse de nuestra generosidad.

No aceptamos ninguna responsabilidad por cualquier efecto, bueno o malo que este programa pueda tener en en usted, su ordenador, su salud, su perro o cualquier otra cosa que pueda imaginar. Uselo bajo su propia responsabilidad.

Listado original del CDROM presentado

Código fuente preparado para cada plataforma: ..... [http://gradha.infierno.org/]

```
/proy/dos
/proy/win
/proy/linux
```

Ficheros mínimos del compilador DJGPP para DOS: ..... [http://www.delorie.com/djgpp/]

```
/files/djgpp/compilad/djdev202.zip
/files/djgpp/compilad/readme.1st
/files/djgpp/compilad/bnu281b.zip
/files/djgpp/compilad/mak377b.zip
/files/djgpp/compilad/gcc281b.zip
/files/djgpp/compilad/fil316b.zip
/files/djgpp/compilad/faq211b.zip
/files/djgpp/compilad/copying.dj
/files/djgpp/compilad/csdpmi4b.zip
```

Herramientas adicionales para DOS:

```
/files/djgpp/tools/mss12.zip ..... [http://hem1.passagen.se/blizzar/mss/index.html]
/files/djgpp/tools/rhid147b.zip ..... [http://www.tu-chemitz.de/~sho/rho/rhide/rhide.html]
/files/djgpp/tools/edi0441i.zip ...[http://www.geocities.com/SiliconValley/Vista/6552/setedit.html]
/files/djgpp/tools/upx081d.zip ..... [http://upx.tsx.org/]
```

Versiones de Allegro para cada plataforma: ..... [http://www.sunsite.auc.dk/allegro/wip.html]

```
/files/djgpp/all3928.zip
/files/djgpp/allegro-demo.dat
/files/linux/all3929.tar.gz
/files/linux/allegro-demo.dat
/files/win/all3930_bin.zip
/files/win/dx70_min.zip
/files/win/all3930.zip
```

Compresor UPX para linux: ..... [http://upx.tsx.org/]

```
/files/linux/upx-0.93-linux.tar.gz
```

Documentación del proyecto en varios formatos: ..... [http://gradha.infierno.org/]

```
/docs
```

Binarios precompilados para DOS y Windows: ..... [http://gradha.infierno.org/]

```
/bin
```

1 VISIÓN DEL JUEGO

Este juego consiste en destruir un muro flotante de ladrillos mediante una pelota y una raqueta. La raqueta está situada en la zona inferior de la pantalla y sólo puede desplazarse horizontalmente. La pelota rebota contra la raqueta y los ladrillos, destruyendo estos últimos con uno o varios contactos.

El juego consta de varios niveles. A medida que se avanza por los niveles, el nivel de complejidad de estos aumenta, y la pelota se desplaza con mayor velocidad debido a los impactos con los objetos de la pantalla.. El jugador tiene tres "vidas" durante el juego.

2 CARACTERÍSTICAS PRINCIPALES DEL JUEGO:

* Portabilidad a cualquier plataforma que use ANSI C y Allegro.
* El juego funciona en cualquier resolución superior a 200x200 pixels.
* El juego está optimizado para funcionar en máquinas lentas, inferiores incluso a 486 si se juega en la resolución adecuada.
* Se puede extender el juego añadiendo más niveles al directorio apropiado.
* El juego tiene opción de logging, guardando en un fichero de texto los sucesos importantes que pudiesen ayudar a detectar fallos en el juego, como datos sobre colisiones.
* Se guardan las puntuaciones en un fichero de texto aparte (arkanoid.rec).
* Se guarda la configuración de pantalla y logging (por defecto desactivado) en un fichero aparte (arkanoid.cfg).

3 CÓDIGO DEL JUEGO:

El juego está escrito completamente en C. Se usa la biblioteca de funciones Allegro para dibujar en la pantalla y detectar las teclas del jugador. El código es compilable con DJGPP (DOS), Microsoft Visual C 5.0 (Windows) y GCC (Linux), en otras palabras: el programa es compatible a nivel de código con cualquiera de las plataformas mencionadas, debido a un estricto uso de C ANSI y Allegro. Al compilar el código, no hemos obtenido ningún warning con ninguno de los compiladores mencionados.

El código del juego está comentado (sólo los ficheros .c). Cada función tiene un comentario explicativo con la siguiente forma:

```
/**[txh]********************************************************************

  Description:

  Return:
  Example:

********************************************************************************/
```

Los comentarios tienen como mínimo el campo Description, que indica qué hace la función. Los otros dos campos son opcionales. El campo Return sólo se especifica si la función no es de tipo void. El campo Example sólo se indica en alguna función compleja o que se podría usar en otros proyectos sin conocimientos previos de éste.

4 NORMAS DE ESTILO

Los nombres de las variables se escriben en minúsculas, y si son palabras compuestas, separadas por un subrayado. Ejemplo: directorio_programa.

Los nombres de las funciones se escriben en minúsculas, y si son palabras compuestas, separadas por un subrayado. Ejemplo: destruye_datos_juego (DATOS_JUEGO *datos);

Cuando se escribe una función, si acepta parámetros de algún tipo, los paréntesis se separan con un espacio del nombre de la función, en caso contrario se escriben junto al nombre de la función. Los parámetros van separados por espacios después de las comas, siempre. Ejemplo:

```
void dibujar_menu (void);

marca_zona_borrado (lista_borrado_logica, 0, 0, RES_LOGICA_X, RES_LOGICA_Y);

dibuja_menu();
```

Los prototipos de las funciones, sean locales o externas, se declaran en los ficheros de cabecera (.h). Las variables globales también se escriben ahí, enmarcadas en condiciones de preprocesado para que sean declaradas sólo en un fichero .c.

El texto se tabula a 3 columnas, y las llaves comienzan y se cierran en la misma columna:

```
while (f < variable)
{
   f  = resultado (5, f);
   guarda_valor (f);
}
```

El texto se escribirá para que sea de fácil lectura en cualquier terminal de 80 o más columnas. Si en alguna llamada a función el número o longitud de los parámetros desbordan la anchura de la pantalla, se escriben en la siguiente línea, a ser posible desde el paréntesis de la función, aunque en caso extremo se tabulará sólo con 3 columnas a partir de la función.

Todos los ficheros llevan un comentario de cabecera con los siguientes puntos:

* Nombre del proyecto
* Nombre del fichero
* Nombre de los autores
* Fecha
* Descripción

5 HERRAMIENTAS USADAS

El proyecto a sido escrito en dos sistemas diferentes. Por un lado, MS-DOS bajo Windows98 en modo ventana, por otro, Linux en modo consola. Los programas usados (están marcados con un asterisco los que están incluidos en el CD del proyecto) son:

* DJGPP(*): Este es un compilador de C gratuito para DOS de 32 bits.
  (http://www.delorie.com/djgpp/)
* Allegro(*): La biblioteca de funciones usada para dibujar gráficos, detectar teclas, y ayudar a portar el juego a cualquier plataforma.
  (http://www.talula.demon.co.uk/allegro/)
* Rhide(*): Entorno de desarrollo integrado (IDE) creado por Robert Höhne. Muy al estilo de los antiguos entornos de Turbo C o Turbo Pascal.
  (http://www.tu-chemnitz.de/%7Esho/rho/rhide/rhide.html)
* Setedit(*): Editor interno de Rhide, pero en aplicación aparte. Permite editar cualquier tipo de texto o crear proyectos, emulando buena parte del IDE Rhide.
  (http://www.geocities.com/SiliconValley/Vista/6552/setedit.html)
* SDG(*): Setedit Documentation Generator, viene incluido en el editor Setedit, y permite generar documentación html a partir de los comentarios de un proyecto C/C++.
* GDB: Gnu debugger, un depurador de línea de comando. Se puede encontrar en Linux o DOS junto con el respectivo compilador de C.
* MSVC: Requerido para compilar la versión de win32 del proyecto.
* MSS(*): Biblioteca de funciones que enmascara las funciones malloc y free, capturando sus llamadas y mostrando información de depuración sobre memoria usada, punteros erróneos, etc.
  (http://hem1.passagen.se/blizzar/mss/index.html)
* Paint Shop Pro 5.0: Para dibujar los niveles en ficheros .pcx.
* UPX(*): Compresor de ficheros ejecutables. Tanto en versión dos, como Linux, este es el mejor compresor de ejecutables y ficheros .dll que existe.
  (http://upx.tsg.org)

6 INSTALACIÓN/COMPILACIÓN DEL PROYECTO

6.1 DJGPP (DOS)

En el directorio ficheros\djgpp se encuentran los ficheros necesarios para instalar el compilador, la biblioteca de funciones Allegro, Rhide y Setedit. Primero hay que instalar el compilador. Para ello, hay que crear una directorio en el disco duro (ej: C:\DJGPP) y descomprimir ahí los ficheros del directorio ficheros\djgpp\compilad, preservando la estructura de directorios y los nombres largos.

Una vez descomprimidos los ficheros, hay que añadir las siguientes líneas al autoexec.bat:

```
SET DJGPP=C:\DJGPP\DJGPP.ENV
SET PATH=C:\DJGPP;%PATH%
```

Tras reiniciar el sistema tecleando `gcc -v` deberíamos obtener un mensaje del compilador indicando la versión de éste. Si ha habido problemas, en C:\DJGPP\ debería haber un fichero llamado readme.1st con instrucciones de instalación más detalladas.

Ahora hay que descomprimir el archivo ficheros\djgpp\all3928.zip dentro del directorio C:\DJGPP. Tras descomprimirlo, se entra en él y se teclea `make`. Esto ejecuta el comando make, que compilará la biblioteca de funciones Allegro. Tras compilarla, se teclea `make install` para instalarla en los directorios del DJGPP. Si hay problemas, es conveniente consultar los ficheros .txt del directorio C:\DJGPP\ALLEGRO.

Hecho esto, ya se puede compilar el proyecto. Para hacerlo, hay que copiar el directorio proy\dos tal cual a un directorio del disco duro (ej: C:\PROY\DOS). Nos situamos en ese nuevo directorio y tecleamos `make`. Esto compilará el código y producirá un ejecutable llamado arkanoid.exe en el directorio actual. Para ejecutar el programa, hay que tener un servidor de memoria DPMI activo.

Si se está usando Windows (cualquiera menos NT), QEMM, o se tiene el compilador DJGPP instalado, entonces no habrá problemas. Para copiar el programa en otra máquina que no disponga de este software, aparte del binario también habrá que copiar el fichero CWSDPMI.EXE del directorio C:\DJGPP\BIN, el cual da servicios DPMI a cualquier programa creado por DJGPP.

En el directorio ficheros\djgpp\tool están los ficheros del RHIDE y Setedit, que fueron usados para el proyecto. Su instalación es opcional, no influyendo en la compilación del juego. El fichero del rhide se descomprime directamente en C:\DJGPP, mientras que el fichero edi0441i.zip que contiene al Setedit hay que descomprimirlo en un directorio temporal para posteriormente ejecutar el fichero install.exe.

6.2 MSVC (WINDOWS)

Compilar la versión Windows del proyecto trae grandes complicaciones, debido a que los compiladores de Windows rara vez traen buen soporte de línea de comando. Algunas versiones de MSVC traen el NMAKE, que funciona de forma parecida al make del DJGPP. Sin embargo, esto no es estándar, y hay versiones del compilador que ni siquiera lo tienen.

Una solución que no es muy buena (porque requiere otro compilador) es instalar DJGPP y usar el programa make de éste. Finalmente se ha optado por no incluir un sistema de compilación automático debido a que no es sencillo ni portable exportar escritorios de desarrollo del IDE de MSVC, e incluir en su lugar un simple fichero de proceso por lotes que crea el fichero final ejecutable, a partir del código fuente, siempre que el compilador esté bien instalado/configurado.

Pero primero hay que compilar Allegro para Windows. Para evitar el requisito de DJGPP+make, James Jonson hospeda en http://sunsite.auc.dk/allegro/wipdll/ una versión precompilada de las .dll de Allegro, que están lista para usar, y las cabeceras de DirectX necesarias para compilar cualquier programa (debido a que Allegro usa DirectX internamente). Los ficheros están incluidos en el CDROM, son

files\win\all3930_bin.zip y files\win\dx70_min.zip.

Para instalar bien Allegro, primero hay que descomprimir files\win\all3930.zip. Tras esto, se descomprime el fichero files\win\all3930.zip en el mismo directorio. Ahora se descomprime files\win\dx70_min.zip dentro del directorio de MSVC, para que instale las librerías y ficheros de cabecera de DirectX. Finalmente haciendo doble-click en el fichero msvcmake.bat del directorio Allegro se debería compilar Allegro a la perfección.

Una vez instalado Allegro, se copia el directorio proy\win al disco duro. Se entra en el nuevo directorio del proyecto y se ejecuta `build.bat`. Se asume que la variable path está bien ajustada, y que el fichero `vcvars32.bat` está bien configurado y disponible. En caso contrario, el programa no podrá ser compilado.

6.3 LINUX

Al igual que en cualquier otro sistema operativo real, en Linux ya se dispone de todas las herramientas necesarias para compilar cualquier programa. En el caso de que no se hubiesen instalado, por necesidades personales de espacio, las herramientas o paquetes necesarios son: gcc, autoconf, automake, make, binutils, sed y grep.

Primero se descomprime ficheros/linux/all3929.tar.gz en el directorio personal de un usuario. Se entra en el directorio creado, y se teclea `./configure` para generar un makefile apropiado. Si configure no ha encontrado problemas en el sistema, podremos ejecutar `make depend;make` para compilar Allegro. Una vez hecho esto, tecleamos como root `make install`, lo cual definitivamente instalará la biblioteca de funciones y el script de enlazado.

Ahora ya se puede compilar el proyecto. Se copia el directorio proy/linux a un directorio de usuario, y dentro de él se teclea make. Se generará un binario de nombre arkanoid, que estará listo para ser ejecutado. Si se está usando un dispositivo framebuffer o se está en X-Window, se podrá ejecutar el programa como usuario normal. En caso contrario, hay que tener permisos de root, debido a la forma que Linux gestiona los recursos hardware.

7 PARTICULARIDADES DE LOS BINARIOS:

Para todas las plataformas hay un factor común: en el directorio donde está el ejecutable también debe haber un subdirectorio llamado levels, con ficheros mapaxx.pcx dentro. Estos son los niveles del juego, y si no se encuentran, la opción de jugar no funcionará (en concreto, funcionará, pero saldrá inmediatamente del bucle de niveles). El juego está compilado para aceptar hasta 50 niveles seguidos, aunque este límite se puede variar recompilando el programa.

7.1 DOS

Los programas que se compilan con DJGPP funcionan bajo MS-DOS o ventana Windows. Los ejecutables resultantes son de 32bits, pueden manejar la memoria de forma plana (flat memory model) y pueden hacer uso de memoria virtual. El único problema es que necesitan un gestor de memoria DPMI.

Windows presta estos servicios de memoria, así como el QEMM para DOS. Si no se tiene ninguno de éstos, se puede usar el gestor gratuito CWSDPMI.EXE, que acompaña al juego.

7.2 WINDOWS

Para ejecutar la versión Windows, hace falta tener DirectX instalado, y tener acceso al fichero .dll que contenga las rutinas de Allegro. El fichero .dll se suministra con el proyecto. Nota: los archivos .dll tienen diferentes versiones. Si se cambia el nombre de una a otra, el programa puede funcionar, o puede bloquear el ordenador debido a que el orden de los puntos de entrada a las funciones pueden haber cambiado.

7.3 LINUX

No se distribuye fichero binario para Linux, porque es absurdo. En primer lugar, los binarios que puede crear una persona pueden no servir a otra, debido a que dependen de otras bibliotecas dinámicas que quizás no estén instaladas en el sistema ajeno. Cada persona debe compilar su versión de Allegro a medida.

Por otro lado, no es buena idea ejecutar binarios tal cual. Pueden ser programas troyanos o que atenten contra el sistema. Ya que se tiene el código fuente del proyecto, el usuario ha de compilar primero Allegro y luego el juego. Así, si antes de ejecutarlo tiene alguna duda sobre el funcionamiento del programa, puede acudir al código fuente.

Si se es capaz de compilar el código fuente, entonces significa que no deberíamos tener problemas para ejecutarlo. En el directorio bin del cdrom se incluyen versiones precompiladas para dos y windows del proyecto, que se pueden ejecutar directamente desde el cdrom.

8 CÓDIGO DEL JUEGO

Debido a la complejidad "lógica" de algunas partes del programa, se incluye una descripción explicativa de las que hemos considerado relativamente complicadas. Para facilitar el seguimiento del código fuente, tras estas explicaciones se incluye un flujo de funciones del programa. El flujo muestra solamente las llamadas que hace cada función, e indica dónde están declaradas, permitiendo saber de un vistazo si se trata de funciones internas o externas a un mismo fichero .c.

Finalmente, en el CDROM, se encuentra el fichero docs\docs.html, que contiene las descripciones individuales de cada función, que fueron extraídas del código fuente mediante el SDG. La documentación incluye algunos hiperenlaces para agilizar la búsqueda de una determinada función.

Por longitud no se incluye una versión impresa del código fuente. Este puede ser encontrado en el CDROM, en cualquiera de los directorios src dentro de proy. Los ficheros están tabulados a 3 caracteres, y para que cualquiera pueda verlos bien, usan espacios en vez de tabulados (cosa que desde cualquier editor decente se puede cambiar en un momento).

La función juego es la que controla la parte principal del programa, pues ya se han realizado todas las inicializaciones necesarias para que el juego se pueda ejecutar en un modo aceptable. Tras haberse puesto a punto los ficheros de logging y errores y tras haber llamado a los módulos de configuración para poder utilizar las funciones de Allegro, aquí es donde comienza el juego en sí.

Lo primero de todo es crear una estructura donde almacenaremos los datos del juego. Para ello llamaremos a una función de inicialización (inicializa_datos_juego) donde reservaremos espacio para los bitmaps de la raqueta, la pelota y los ladrillos, inicializaremos el color de fondo de la pantalla, y daremos forma a la raqueta y a la pelota.

Después mostraremos un menú de opciones en la pantalla, creando un bitmap temporal, que destruiremos cada vez que se seleccione una opción, en el que escribiremos las 3 opciones que tiene el juego, rescaladas y formateadas para que tengan el tamaño adecuado para poder ser mostradas en pantalla. La opción seleccionable se diferenciará de las demás en que estará dibujada en un color diferente.

A continuación creamos un bucle que no finaliza hasta que se desea salir del juego. Dentro de él, el usuario seleccionara la opción deseada utilizando para ello los cursores y el <ENTER>. Cada vez que mueva los cursores se dibujará en pantalla el bitmap temporal del menú cambiando de color la nueva opción seleccionada.

La selección de las opciones viene dada por una estructura de control case: Según se elija, se ejecutará bucle_juego, se mostrarán los récords o se saldrá del juego. La opción de récords llama a una función externa que muestra en pantalla las mejores puntuaciones almacenadas en un fichero o las puntuaciones que se muestran por defecto si este fichero está vacío. Tras la pulsación de una tecla se vuelve al menú.

La opción de salir activa la variable que detiene el bucle del juego y destruye todos los datos y estructuras que almacenan variables de éste. La función cederá el control a otros ficheros indicándoles según el valor que devuelva si todo se ha realizado satisfactoriamente o no.

Pasemos ahora a describir la opción principal: la de empezar a jugar. Esta opción llama a la función bucle_juego.

La función bucle_juego comprueba si hasta el momento la estructura con la información del juego no está vacía, y llama a una función que inicializa las variables que deben tener un valor fijo cada vez que el programa pase por este punto del juego: El principio de la partida.

Tras esto, se entra en un nuevo bucle que sólo finalizará si se activan los bits que indican que el juego ha llegado a su fin, o que se ha pulsado <ESC>, a través de la cual se sale del juego directamente y se retorna al menú. Dentro de él se comprueba previamente si existen pantallas/niveles de juego en los cuales

se pueda jugar.

Si esto es posible se cargará el correspondiente nivel, y se inicializan los datos para que tengan el valor adecuado cada vez que el juego pase por este punto del juego: El principio de cada pantalla. Finalmente y si no se ha producido ningún error, llamamos a la función procesa_juego.

En este punto encontramos otro bucle, en el que debemos destacar ciertos aspectos característicos de este juego. Aquí se ejecutan las rutinas de comprobación de pulsación de teclas, del "bucle lógico" y de dibujado en pantalla.

Todas estas rutinas tienen algo en común: su tiempo de funcionamiento lo controlamos mediante funciones de interrupción externas, es decir, que somos nosotros los que controlamos su tiempo de funcionamiento, permitiéndonos que este programa se pueda ejecutar en ordenadores de distintas características a la misma velocidad.

El bucle que llama a la rutina de comprobación de pulsación de teclas, por ejemplo, se ejecuta un número de veces determinado por un contador de ticks previamente inicializado por una función de interrupción de allegro. Después llamamos a la función bucle_logico que de la misma forma se ejecuta un número concreto de veces con la ayuda de su respectivo contador y que llama a las funciones "lógicas" del juego.

Finalmente, comprobamos algunas variables de estado que deben ser chequeadas para determinar si el juego podrá continuar en la siguiente pasada del bucle, en cuyo caso reinicializaremos las correspondientes variables, y pasamos todas las coordenadas "lógicas" a la pantalla "física". Es importante la distinción que hacemos entre lo "lógico" y lo "físico" dado que es otra característica del juego.

Como ya hemos comentado, antes de dibujar nada en la pantalla utilizamos bitmaps y coordenadas lógicas en las que "dibujamos" lo que queremos que se vea en la pantalla. Dado que dentro de estas rutinas hay bucles de redibujado que dificultarían dibujar en cada pasada en la pantalla "física", que es el monitor, lo hacemos en "coordenadas lógicas", que son posiciones de memoria de los bitmaps, lo que nos permite que el programa funcione de forma adecuada en ordenadores a distintas velocidades y evitando problemas de parpadeo en la pantalla, característica principal del programa. También permite visualizar el juego en cualquier resolución posible, siempre y cuando sea mayor que 300x200 pixels.

Una ventaja importante de tener separado el bucle lógico del físico es que gracias a los temporizadores podemos saber si uno se está retrasando o adelantando, y en consecuencia podemos compensar al otro adecuadamente, equilibrando la velocidad real del juego para cualquier máquina, sea ésta lenta o rápida.

La rutina de detección de colisiones es también interesante: por cada pasada del bucle lógico, la rutina descompone el movimiento de la pelota. Transforma longitud y ángulo en incrementos x e y. Con éstos, puede calcular dónde estará la pelota. Por seguridad, si la distancia es grande, para evitar saltarse chequeos de colisión, trocea la longitud a recorrer en incrementos de un pixel. Luego, por cada pixel intenta detectar colisiones alrededor de la pelota, e intenta determinar la dirección que tiene que tomar la pelota.

Los mapas de los niveles del juego son simples ficheros .pcx a 8bits. Deben tener un tamaño mínimo de 50x40 pixels. Cada pixel del mapa representa un cuadro de 5x5 pixels en el juego. Se puede elegir entre 15 colores diferentes para pintar los ladrillos. El resto de los colores duplicados sirve para diferenciar dos ladrillos de mismo color. Es decir, que aunque dos ladrillos sean amarillos, por ser sus índices de paleta diferentes, en el juego aparecerán como dos ladrillos amarillos separados. De esta forma se pueden construir bloques de cualquier forma y tamaño.

El programa acepta unos pocos parámetros en la línea de comando. En todas las versiones estos son:

* -h, –help
  Muestra la ayuda rápida del programa, que indica los demás parámetros del juego.

* -r, –res
  Obliga al juego a reelegir la resolución de pantalla. Se puede usar este parámetro o directamente borrar

el fichero arkanoid.cfg.

* -l xxx, –log xxx
  Donde xxx es el nombre de un fichero, el programa creará este fichero y añadirá a él en modo texto, líneas con información interna del juego, en definitiva, creará un fichero log. El fichero log puede servir para que se vea qué es lo que hace el programa (qué parte del código se está ejecutando), y para mostrar valores de depuración que pudiesen ayudar a detectar fallos en el juego.

Por lo que hemos visto, el juego funciona sin problemas aparentes. Si por alguna razón el juego pareciese bloquearse debido a algún fallo, antes de resetear la máquina es mejor intentar desbloquear el juego con la combinación de teclas Ctrl+Alt+Fin. Esta es una combinación "mágica" de Allegro, que intenta desbloquear cualquier programa, y en nuestras pruebas, siempre ha interrumpido con éxito el programa.

A continuación se incluye el flujo de funciones del programa. Es recomendable leerlo mientras se tiene la ayuda de las funciones en formato html abierta en el ordenador, para poder ir consultando lo que hace cada función.

9 FLUJO DE FUNCIONES DEL PROGRAMA

```
////////////////////////////////////////////////////////////////////////
   FICHERO MAIN.C
////////////////////////////////////////////////////////////////////////

main;
{
   inicializa_programa; -> función declarada en init.c
   juego; -> función declarada en juego.c
   finaliza_programa; -> función declarada en init.c
}

////////////////////////////////////////////////////////////////////////
   FICHERO INIT.C
////////////////////////////////////////////////////////////////////////

inicializa_programa; /* función llamada desde main.c */
{
   llamadas_de_inicialización_varias_de_Allegro;
   incrementar_ticks;
   incrementar_secs;
   inicializa_mensajes_de_error; -> función declarada en errores.c
   decodifica_línea_de_comandos;
   lee_configuración_del_fichero_cfg;
   entra_en_modo_grafico;
   inicializa_logging; -> función declarada en log.c
   init_hiscore; -> función declarada en hiscore.c
}

/* Las dos siguientes funciones son ejecutadas por interrupciones, por
lo que aparentemente no son llamadas desde ningún sitio */

incrementar_ticks;
incrementar_secs;

decodifica_línea_de_comandos;
{
   comprueba_validez_fichero_log; -> función declarada en log.c
}

finaliza_programa; /*función llamada desde main.c */
{
   finaliza_logging; -> función declarada en log.c
   sal_del_modo_gráfico;
   shutdown_hiscore(); -> función declarada en hiscore.c
   muestra_mensajes_de_error; -> función declarada en errores.c
   muestra_ayuda_línea_de_comando;
   allegro_exit;
}

lee_configuración_del_fichero_cfg;
{
   comprueba_validez_fichero_log; -> función declarada en log.c
}
```

```
entra_en_modo_gráfico;
{
   comprueba_validez_fichero_log; -> función declarada en log.c
   inicializa_motor_gráfico; -> función declarada en graph.c
}

sal_del_modo_gráfico;
{
   finaliza_motor_gráfico; -> función declarada en graph.c
}

muestra_ayuda_línea_de_comando;
{
   imprime_información_sistema;   -> función declarada en log.c
}

////////////////////////////////////////////////////////////////////////
   FICHERO ERRORES.C
////////////////////////////////////////////////////////////////////////

inicializa_mensajes_de_error; /* función llamada desde init.c */

muestra_mensajes_de_error; /* función llamada desde init.c */

eprintf; /* función llamada desde múltiples ficheros fuente */
{
   finaliza_programa; -> /* función declarada en init.c */
}

////////////////////////////////////////////////////////////////////////
   FICHERO LOG.C
////////////////////////////////////////////////////////////////////////

imprime_informacion_sistema; /* función llamada desde main.c, init.c */

comprueba_validez_fichero_log; /* función llamada desde init.c */

inicializa_logging; /* función llamada desde init.c */
{
   imprime_información_sistema;
}

finaliza_logging; /* función llamada desde init.c */
{
   libera_mensajes_log;
   destruye_mensaje_log;
}

libera_mensajes_log;
{
   destruye_mensaje_log;
}

lprintf; /* función llamada desde múltiples ficheros fuente */
{
   crea_mensaje_log;
```

```
   elimina_excedente_lista_log;
}

crea_mensaje_log;

elimina_excedente_lista_log;
{
   destruye_mensaje_log;
}

destruye_mensaje_log;

////////////////////////////////////////////////////////////////////////
   FICHERO JUEGO.C
////////////////////////////////////////////////////////////////////////

juego; /* función llamada desde main.c */
{
   inicializa_datos_juego;
   dibuja_menu;
   recoge_entrada_usuario; -> función declarada en logico.c
   dibuja_menu;
   bucle_juego;
   score_table;
   destruye_datos_juego;
}

inicializa_datos_juego;
{
   destruye_datos_juego;
}

bucle_juego;
{
   inicializa_datos_fijos;
   carga_nivel;
   inicializa_datos_nivel;
   procesa_juego;
   score_table; -> función declarada en hiscore.c
   dibuja_menu;
}

inicializa_datos_fijos;

carga_nivel;
{
   carga_mapa_ladrillos;
}

carga_mapa_ladrillos;

inicializa_datos_nivel;
{
   dibuja_marco_pantalla; -> función declarada en graph.c
   marca_zona_borrado; -> función declarada en lborrado.c
}
```

```
procesa_juego;
{
   recoge_entrada_usuario; -> función declarada en logico.c
   bucle_logico; -> función declarada en logico.c
   inicializa_datos_nivel;
   dibuja_pantalla; -> función declarada en graph.c
}

destruye_datos_juego;

dibuja_menu;
{
   escribe_texto; -> función declarada en graph.c
}

////////////////////////////////////////////////////////////////////////
   FICHERO GRAPH.C
////////////////////////////////////////////////////////////////////////

dibuja_marco_pantalla; /* función llamada desde juego.c */

dibuja_pantalla; /* función llamada desde juego.c */
{
   recupera_zonas_borrado; -> función declarada en lborrado.c
   dibujar_pelota;
   dibujar_raqueta;
   dibujar_marcador;
   recupera_zonas_pantalla; -> función declarada en lborrado.c
}

dibujar_pelota;
{
   marca_zona_borrado; -> función declarada en lborrado.c
}

dibujar_raqueta;
{
   marca_zona_borrado; -> función declarada en lborrado.c
}

dibujar_marcador;
{
   escribe_texto;
}

escribe_texto;
{
   _escribe_texto;
}

_escribe_texto;

dibujar_ladrillos; /* función llamada desde lborrado.c */
{
   color_ladrillo_diferente;
```

```
}

color_ladrillo_diferente;

inicializa_motor_grafico; /* función llamada desde init.c */
{
   obten_escalado_optimo;
   inicializa_listas_borrado; -> función declarada en lborrado.c
   inicia_tabla_de_colores;
}

obten_escalado_optimo;

inicia_tabla_de_colores;

finaliza_motor_grafico; /* función llamada desde init.c */
{
   destruye_lista_borrado; -> función declarada en lborrado.c
}

inicializa_motor_grafico;
{
   obten_escalado_optimo;
   inicializa_listas_borrado; -> función declarada en lborrado.c
   inicia_tabla_de_colores;
}

obten_escalado_optimo;

inicia_tabla_de_colores;


//////////////////////////////////////////////////////////////////////
   FICHERO LBORRADO.C
//////////////////////////////////////////////////////////////////////

inicializa_listas_borrado; /* función llamada desde graph.c */

destruye_lista_borrado; /* función llamada desde graph.c */
{
   limpia_lista_borrado;
}

recupera_zonas_borrado; /* función llamada desde graph.c */
{
   dibujar_ladrillos; -> función declarada en graph.c
   marca_zona_borrado;
   limpia_lista_borrado;
}

recupera_zonas_pantalla; /* función llamada desde graph.c */
{
   limpia_lista_borrado;
}

marca_zona_borrado; /* función llamada desde graph.c y lborrado.c */
```

```
limpia_lista_borrado;

////////////////////////////////////////////////////////////////////////
   FICHERO LOGICO.C
////////////////////////////////////////////////////////////////////////

recoge_entrada_usuario; /* función llamada desde juego.c */

bucle_logico; /* función llamada desde juego.c */
{
   avanza_pelota,
   detectar_muerte_pelota;
   mover_raqueta;
}

avanza_pelota;
{
   detectar_colision_pelota;
}

detectar_muerte_pelota;

mover_raqueta;
{
   colision_pelota_raqueta;
}

colision_pelota_raqueta;

detectar_colision_pelota;
{
   hay_colision;
   elimina_ladrillo;
}

hay_colision;

elimina_ladrillo;
{
   comprueba_si_quedan_ladrillos;
   marca_zona_borrado; -> función declarada en lborrado.c
}

comprueba_si_quedan_ladrillos;

////////////////////////////////////////////////////////////////////////
   FICHERO HISCORE.C
////////////////////////////////////////////////////////////////////////

init_hiscore;

shutdown_hiscore;

score_table;
{
```

```
   draw_entry_box;
}

draw_entry_box;
```

Índice General