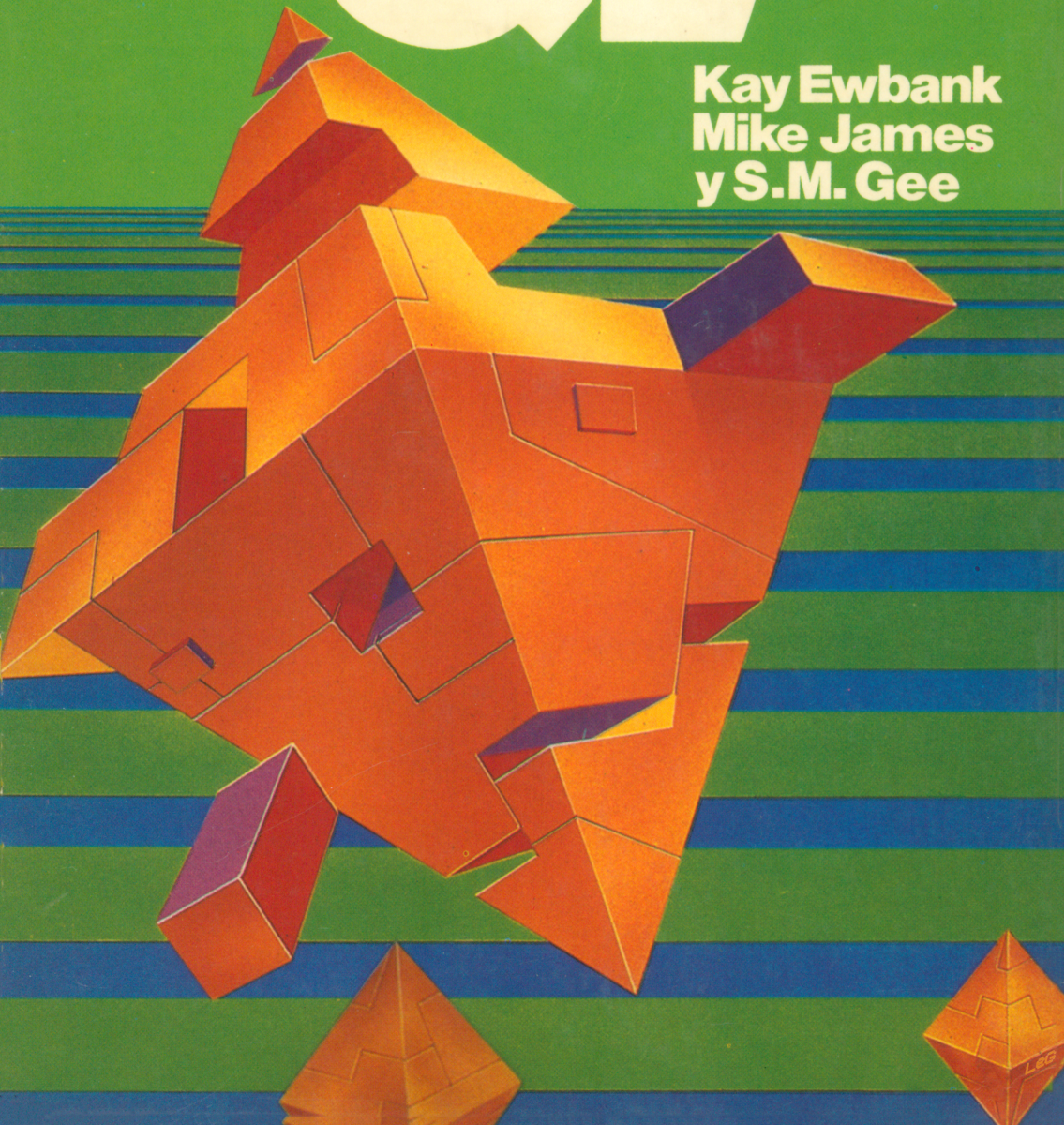
GENIO DE LOS JUEGOS con el QL

Kay Ewbank
Mike James
y S.M. Gee



Genio de los Juegos con el QL

Kay Ewbank, Mike James y S. M. Gee

indescomp

Collins Professional and Technical Books
William Collins Sons & Co. Ltd
8 Grafton Street, London W1X 3LA

Primera publicación en Gran Bretaña por
Collins Professional and Technical Books 1985

Versión en Castellano

Editado por **INDESCOMP, S.A.**
Avda. del Mediterráneo, 9 - 28007 MADRID (ESPAÑA)

Traduce, compone e imprime: CONORG, S.A.

I.S.B.N.: 84-86176-31-X

Depósito Legal: M-18275-1985

Contenido

Prefacio

Dominar la programación, como cualquier otra disciplina, exige práctica. Pero no solamente consiste en trabajar con ejercicios preestablecidos -en algún momento tienes que partir de la nada y comenzar a desarrollar tus propias ideas. Montones de principiantes ven esto como una perspectiva amenazadora y lo encontrarían mucho más fácil si pudieran conseguir que alguien les ayudara, enseñándole sus habilidades. Este libro pretende proporcionar la misma clase de ayuda que un programador experto -y además con capacidad didáctica- sería capaz de ofrecer a un principiante. Te lleva a través de las diversas etapas en el diseño y la confección de programas, señalándote problemas y peligros y ofreciéndote pistas y soluciones.

Dado nuestro intento de enseñar el arte de escribir programas grandes, te puedes estar preguntando cómo hemos elegido concentrarnos en juegos. Hay dos razones principales. Por un lado los juegos son divertidos -divertidos para jugar y también divertidos para diseñarlos. Más importante desde el punto de vista de la enseñanza es que en ellos se combinan todos los aspectos de la programación; porque ¿dónde encontraríamos efectos gráficos y sonoros combinados con números aleatorios y lógica Booleana? Los programas desarrollados en este libro son todos juegos con figuras animadas y algunos de ellos, como mínimo, son los habituales juegos clásicos de video. Puedes haber pensado que programar tales juegos está fuera de tu alcance, pero este libro te demuestra cómo abordar la escritura de programas para juegos, incluso con ese nivel de complejidad y 'sofisticación'.

Antes de embarcarte en el diseño de juegos concretos, hemos tenido que hallar un remedio para el fallo del QL en incorporar **símbolos gráficos** que son esenciales en el tipo de juegos que queremos realizar. La solución que decidimos sobre este tema, fue añadir un modo gráfico completamente nuevo -un modo en **baja resolución** que complemente los dos modos de alta resolución ya disponibles en el QL. Ampliando las facilidades del QL de esta manera demuestra que es mucho más aprovechable de lo que se pudiera haber imaginado y simplemente requiere dos nuevos **procedimientos** que explicamos en el Capítulo Uno y luego usamos a lo largo del libro.

Todos los programas para juegos están construidos como una serie de **módulos de programación** y sugerimos que teclees cada módulo a medida que se presentan. No solamente con eso se disminuye la labor de teclear programas largos, sino que también se te da la oportunidad de comprender hasta los detalles más finos. Una vez que hayas tecleado los juegos, esperamos que disfrutes jugándolos, tengas confianza para remendarlos -y mejorarlos- y hayamos fomentado en tu ánimo la disposición para escribir programas completos por ti mismo.

Debemos dar gracias a Richard Miles y Prue Harrison de la Editorial Collins, sin cuya ayuda este libro no hubiera sido posible.

Kay Ewbank
Mike James
S. M. Gee

Capítulo Uno

El Aprendiz de Brujo

Este libro es bastante diferente de cualquiera que puedas haber leído anteriormente sobre programación de ordenadores. No es simplemente una colección de juegos divertidos, ni tampoco es otro libro sobre **aprender a programar.** Lo que el libro hace es combinar el gozo de crear con la oportunidad de adquirir esa valiosa clase de conocimiento que sólo surge de la experiencia práctica. Hoy en día hay más gente que conoce un **lenguaje de programación** -pero saber un lenguaje es sólo el comienzo de la labor de aprender **cómo usarlo.**

Muchas personas se percatan que después de haber aprendido BASIC todavía tienen problemas en escribir programas que **efectúen** una labor concreta. El tipo de dificultad con que se encuentran, varía desde no ser capaz de comenzar un programa, hasta no ser capaz de resolver los problemas que se le acumulan durante el desarrollo del programa, y hasta no ser capaz de acabar un programa que sea de utilidad. En otras palabras, las pegas pueden surgir al principio, en el medio o al final del desarrollo del programa, o desde luego, como combinación de las tres etapas. La observación que hacemos es que, como mínimo al principio, es necesario una gran cantidad de energía para abordar un programa largo. Se pueden escribir pequeños programas, se pueden comprobar y se pueden desechar en una tarde, pero los programas largos puede tardarse meses en completarlos. Las dificultades que surgen al trabajar con un gran número de líneas de BASIC, a lo largo de un período extenso de tiempo, realmente son nuevas y bastante diferentes de todo lo que te dicen en la "Introducción al BASIC". Cuando llega la hora de abordar problemas reales con BASIC, no es suficiente saber por ejemplo lo que hace la instrucción IF...; es esencial saber **cómo usarla** en combinación con el resto de las instrucciones del BASIC para producir una **solución** en forma de un programa que no sólo funciona, sino que también apetece usar.

Este libro intenta aliviar el problema de la escritura de programas extensos, explicando cómo escribimos programas largos para juegos. La razón de poner el énfasis en los juegos, es simplemente que su atractivo es (casi) universal y que en ellos se contiene la mayoría de los problemas encontrados en otras áreas de aplicación. Si simplemente quieres usar el libro como una colección de juegos, no hay nada que te lo impida -simplemente salta directamente a los listados dados al final de cada capítulo y tecléalos- pero te encontrarás con que el disfrute del juego es muchísimo mayor si sigues los comentarios del capítulo y tecleas los programas en pequeñas dosis.

Cómo escribir programas

Nadie niega el hecho de que la mejor manera de escribir programas que tengan sustancia es trabajar con un programador experto. En este sentido, la programación no es diferente de ningún otro "oficio artesano" -necesitas servir como aprendiz antes de graduarte como oficial maestro en este gremio. El problema es que en este momento los **programadores expertos** no abundan demasiado. Si puedes encontrar alguno que te ayude, aprovecha la oportunidad para aprender de primera mano -esa siempre es la mejor manera. Pero encuentres o no esa ayuda, este libro será valioso para ti al mejorar tus técnicas de programación.

Cada capítulo comienza con una descripción del juego que con el programa se va a **implementar**, señalando algunas de las dificultades potenciales y sugiriendo métodos que probablemente serán útiles. Después de leer esa introducción, debieras tener la misma información que un programador experto tendría antes de comenzar a resolver los problemas inesperados que brotan durante el desarrollo del programa.

Detrás de esa introducción, el programa se presenta sección por sección. Cada sección del programa efectúa una **sola acción** dentro del programa y eso se explica en el texto que la acompaña. Sugerimos que teclees cada sección del programa tal y como te la presentamos. No solamente es una manera menos tediosa de teclear un programa, sino que también te da la oportunidad de analizar y estudiar concienzudamente cada sección.

Al final de cada capítulo, te encontrarás un comentario de lo que se ha conseguido en la primera versión del programa y cómo puede ser mejorada. Luego, y si es necesario para hacer el juego más atractivo, se sugieren modificaciones y se **lista** la versión final del programa. Este listado completo del programa debiera ser usado para comprobar que has tecleado todo, incluyendo las modificaciones, correctamente. Sobre este tema de la exactitud merece la pena decir que todos los programas de este libro han sido sacados por impresora directamente de las versiones de trabajo sin que intervenga una etapa de composición (propensa a errores), es decir, están todos presentados en la forma de **listados por impresora.**

Las explicaciones de cómo trabaja el programa y por qué las cosas se hicieron de una manera particular, debiera ayudarte a comprender el proceso de escribir programas largos, aunque por razones de la longitud del texto no es posible mostrar todas las etapas por las que pasaron los programas durante el desarrollo. Sin embargo, los programas tal y como están listados no han sido 'pulidos' como puedes ver por ejemplo en la **numeración** de líneas. De esta manera, aunque continúan reflejando la clase de producto acabado que sería aceptable para ser usado, siguen conservando la evidencia de los problemas de programación. Hay una tendencia con los programas publicados de pulirlos hasta el punto en que clínicamente quedan pulcros y esconden las dificultades que el programador tuvo, o bien simplemente dejarlos como marañas complejas que puede que sean utilizables, pero que es imposible de comprender o modificar. Los programas en este libro no se han pulido del todo y así puede verse dónde se insertaron líneas en el programa en una etapa posterior. Por ejemplo, si encuentras una secuencia de números de línea 10, 20, 25, 30, 40... puedes apostar con seguridad que el programador tuvo que volver atrás e insertar la línea 25 para tener en cuenta algo que se le había escapado. Por otro lado, todos los programas han sido confeccionados usando un **método estructural de programación** que tiende a producir programas más fáciles de entender.

A medida que teclees y analices cada sección del programa, no debes retraerte de intentar comprobar si lo has comprendido ejecutando un programa parcialmente completo; desde luego, a no ser que añadas **temporalmente** líneas pensadas por ti mismo, obtendrás mensajes de error de estos programas parciales. Sin embargo, en la mayoría de los casos observarás algo de interés que te permitirá explicar, de acuerdo con lo que ya sabes sobre el programa, y eso incrementaría tu confianza y te haría pensar y razonar más concienzudamente sobre el programa. Este libro es un **libro práctico**, así que siéntete completamente libre en modificar y experimentar con los programas tanto a medida que avanzas como una vez que los hayas completado. Desde luego, incluso este término "completado" es relativo -hay un viejo dicho en programación que dice que nunca ningún programa estará acabado, ¡simplemente alcanza un punto donde es usable! Para animarte a modificar y mejorar/estropear los programas, al final de la mayoría de los capítulos te hacemos sugerencias concernientes normalmente con las mejoras -pero tú puedes hacer las "peoras"- e incluso se dan nuevas versiones del juego.

Aunque es un libro práctico, incluso los temas más prácticos necesitan **algo de teoría**, y por tanto, la siguiente parte de este capítulo describe algunas de las ideas que fundamentan la programación.

La necesidad de metodología

Como ya hemos mencionado, hay una diferencia real entre escribir programas pequeños y grandes. Un programa pequeño puede escribirse de una sentada y la mayoría de él puede retenerse en la memoria del programador. Sin embargo, un programa grande es demasiado largo para escribirlo de una pasada y tiene tantas líneas de instrucciones BASIC que el programador que pueda recordarlo todo es la rara excepción. Es posible escribir programas largos de la misma manera que se escriben los cortos, pero es un trabajo muy duro y una de las razones por la que muchos programadores abandonan proyectos largos y producen muy pocos programas acabados. Incluso si se emplea la necesaria cantidad de tiempo y esfuerzo en escribir un programa largo a la manera de los cortos, el producto acabado es habitualmente tal guirigay que cualquier modificación para mejorarlo o intentar **depurarlo** -es decir, quitarle las pifias- es a menudo totalmente imposible.

Esta situación es una gran faena porque representa muchas horas de esfuerzo desperdiciado en programación. La verdad es que si usas un **método** de programación para organizar tu esfuerzo, la escritura de un programa largo no es más difícil que la de uno corto. A menudo, la sugerencia de usar un método de programación ataca las entrañas de los programadores amantes de la libertad. El deseo de no tener restricciones al escribir un programa es especialmente comprensible si sólo programas por diversión. Después de todo, ¿cuál es la razón de convertir una actividad placentera de **entretenimiento** en otra actividad rutinaria y reglamentada que se parece en todo a un **trabajo**? La respuesta es que el método de programación no es una receta para regimentar la actividad, es simplemente el arte de programar.

Refinamientos graduales y procedimientos

El refinamiento paso a paso no es sólo un método de programación; es la estrategia general para resolver problemas de cualquier clase y por tanto merece la pena conocerlo. Expresado con sencillez, el desglose de una labor en tareas y el gradual 'refino' de los subprogramas que las llevan a cabo, se basa en la vieja idea de 'divide y conquistarás'. Un programa grande puede abordarse mejor dividiéndolo en una colección estructurada de programas más pequeños. Los programas más pequeños pueden luego dividirse todavía más y así hasta que el resultado es lo suficientemente concreto como para ser tratado **en una sola sesión.**

Descrito de esta manera, este método de programación resulta obvio, pero todavía permanece el problema de cómo dividir el programa grande. ¡Eso es algo que queda demostrado por cada uno de los programas para juegos que hay en este libro! Es posible, sin embargo, decir un poco más sobre esta tarea de **desglosar una labor en tareas.** Supongamos que quieres escribir un programa que juege al ajedrez. El máximo problema que la mayoría de los programadores tendrían es realmente el de ¡comenzar! Si piensas sobre ello por un momento, serás capaz de percatarte que cualquier programa que juegue al ajedrez, primero tiene que dibujar el tablero y colocar las piezas en las casillas iniciales (inicializar) y luego ofrecer al jugador humano su turno, luego efectuar él su movimiento, y así hasta llegar al jaque mate. Dicho en palabras informáticas de estilo SuperBASIC, diríamos:

```
10 dibuje_tablero
20 REPeat jugada
30 mueve_humano
40 mueve_mecano
50 IF jaquemate THEN EXIT jugada
60 END REPeat jugada
70 STOP
```

Esta es, de hecho, la primera etapa en el trayecto para confeccionar un programa completo de juego al ajedrez. En otras palabras, este es el primer nivel en el **desglose** de la labor a realizar por el programa. El siguiente nivel consistirá en escribir las secciones de programa (los procedimientos) mencionadas en la sección anterior del programa, que normalmente se denomina la **subrutina principal.**

Cada uno de los procedimientos, de las subrutinas, se escribiría desglosándolas a su vez en una lista de tareas más pequeñas y concretas, y así hasta que estuviera escrito todo el programa. Observa que usando este método estructurado siempre es fácil de iniciar la confección de un programa, escribiendo los **nombres de los procedimientos** a los que se ha de recurrir y que todavía no están escritos. También demora la solución de cualquier problema difícil hasta que realmente tienen que resolverse tareas concretas. Por ejemplo, en el programa de ajedrez no hay ninguna necesidad de resolver el problema de cómo ha de mover la máquina en su turno hasta una etapa bastante posterior en la confección del programa.

Otra ventaja de usar **procedimientos** como parte del desglose de la labor a realizar por el programa, es que el programa resultante es mucho más fácil de comprender y mucho más fácil de cambiar.

Por ejemplo, cualquiera que analizara la rutina principal presentada para el juego de ajedrez, se percataría inmediatamente que siempre el jugador humano mueve primero. Eso podría cambiarse fácilmente, canjeando las líneas 30 y 40. Imagínate lo difícil que este simple cambio sería si las partes del programa que efectúan la tarea de mover las fichas, no estuvieran en la forma de tareas y procedimientos asociados, y no estuvieran por tanto separadas del resto del programa.

Hay tantas ventajas en usar este desglose gradual de la labor a realizar, y de asociar a cada labor un procedimiento concreto y manejable, que es imposible enunciarlas todas. Sin embargo, muchas de esas ventajas se harán patentes al realizar los ejemplos presentados en los capítulos siguientes.

Estructura y estilo

El desglose gradual de una labor es un método que ayuda con la escritura de un programa, al dividirlo en segmentos o secciones más pequeñas y concretas. Sin embargo, se alcanza un punto en ese desglose en el que el procedimiento tiene que hacer algo y no simplemente apelar a otros procedimientos más sencillos. El desglose gradual no establece la guía sobre cómo debiera aprovecharse el BASIC para conseguir cualquier resultado buscado. El método de programación que **sí** dice algo sobre esas cosas es lo que se denomina programación estructurada.

La programación estructurada es esencialmente una **técnica** para escribir programas claros y transparentes, al evitar enredar el flujo de control con montones de bucles y nudos. Por ejemplo, usando la sentencia de salto GOTO es posible hacer un brinco hasta cualquier otra parte del programa, pero si la usas con demasiada liberalidad, descubrirás que tus programas son imposibles de comprender y depurar. Es muy difícil seguir lo que está pasando en un programa que brinca de un lado a otro, y si no puedes seguir el orden en que se ejecutan las instrucciones, entonces habrá un montón de sitios para que se cuelen las **pifias**. La programación estructurada asegura que el flujo de control es siempre fácil de seguir al permitir únicamente al programador emplear unas pocas maneras de cambiarlo y desviarlo.

La selección más habitual de cambios en el flujo de control es la instrucción condicional IF..., el bucle preconfinado FOR... y el bucle condicionado REPEAT..., y es posible escribir cualquier programa usando solamente estas tres estructuras lingüísticas. En otras palabras, **nunca** es imprescindible emplear la instrucción de salto GOTO dentro de un programa, y eso es lo que ha provocado que se denomine -incorrectamente- a la programación estructurada como el arte de no usar la instrucción VAYA A... En la práctica es mejor considerar esta instrucción de salto como peligrosa, pero no como prohibida. Siempre y cuando una sección de programa quede claramente escrita, en base a saber cuándo y cuáles son las instrucciones que se ejecutan, entonces todo estará perfecto y muchas veces la mejor manera de asegurar la claridad y nitidez en un programa es usar la sentencia de salto GOTO.

Otro elemento que contribuye a la claridad de un programa es el uso de **nombres apropiados**, tanto para las variables como para las subrutinas y procedimientos; es decir, nombres que significuen y recuerden algo. Por ejemplo, si un procedimiento efectúa la tarea de exponer el tablero sobre el que se desarrolla el juego, debiera llamarse algo así como **expo_tablero** y no **xy4**. Si puedes imaginar nombres apropiados para las variables y los procedimientos, te encontrarás que no sólo tus programas son más fáciles de comprender y son casi autodocumentados -es decir, puedes prescindir de las sentencias **momorandum** REM- sino también tú mismo entiendes mejor lo que estás programando. En la práctica, habitualmente es muy difícil inventar la cantidad de nombres necesarios y mantenerlos breves, y no hay nada peor que tener que teclear un nombre excesivamente largo una y otra vez en un programa. Buscar denominaciones para las variables y los procedimientos es algo en que o eres bueno o malo (como la mayoría de los programadores), pero que sin ninguna duda merece el esfuerzo intentar mejorar.

Toda esta charla sobre claridad del programa y estilo de programación puede que te preocupe si siempre te han enseñado que el aspecto más importante de la programación era la eficiencia, i.e. maximizar la velocidad y minimizar la utilización de memoria. Desde luego que es importante la eficiencia en cualquier trabajo práctico y que ocasionalmente el estilo ocupará un segundo lugar; por ejemplo, a menudo es necesario usar variables enteras aunque eso haga que el programa luzca muy poco, con el signo de porcentaje detrás de cada nombre. Sin embargo, si produces un programa largo con montón de memoria disponible para añadidos y modificaciones, y nadie puede comprenderlo (incluso tú mismo al cabo de un corto período de tiempo), entonces los añadidos y las modificaciones nunca se harán y cualquier **gazapo** permanecerá con toda probabilidad. El énfasis debe siempre colocarse sobre un programa claramente inteligible -y cualquier cosa que haya de hacerse para aumentar la velocidad o reducir la memoria usada debiera hacerse como modificaciones a un programa bien confeccionado.

La manera más obvia de incrementar la eficiencia de cualquier programa es usar **lenguaje ensamblador.** El microprocesador 68008 usado en el QL es un microprocesador muy bueno para escribir programas en ese lenguaje. Desafortunadamente, por el momento no hay disponible ningún programa ensamblador standard para el 68008 y eso hace más difícil incluir **subrutinas en ensamblador** dentro de cualquiera de los juegos de este libro. Sin embargo, no hemos ignorado totalmente esta posibilidad, como descubrirás en la siguiente sección.

Un modo para juegos en el QL

Los dos modos gráficos de **alta resolución** en el QL son excelentes para dibujar gráficos y para una amplia variedad de otras aplicaciones, pero realmente no son los más adecuados para programar juegos. La razón descansa en que los comandos de alta resolución trabajan en términos de rayas y motas, y los programas para juegos involucran el movimiento de figuras sólidas definibles por el usuario -de **motivos**- a lo largo de la pantalla. La solución obvia es usar los comandos de textos, i.e. de exposición PRINT y de ubicación AT del cursor en combinación con los gráficos definidos por el usuario.

Sin embargo, los gráficos definidos por el usuario disponibles en el QL (y descritos en una hoja de información separada proporcionada por Sinclair Research) sólo ocupan un **cuadratín de 5 x 9 puntos,** y cada uno está rodeado por un reborde del color del fondo. Como resultado de eso, es difícil definir motivos gráficos con la suficiente resolución como para que sean interesantes, y es imposible empalmarlos hasta formar una figura continua más grande, i.e. la del típico submarino.

La solución a este problema de los gráficos definibles por el usuario que se ha adoptado en este libro, es la de agregar un modo gráfico **completamente nuevo;** lo que puede parecer un poco drástico pero de hecho no es difícil y sirve para corroborar lo fácilmente que se pueden ampliar las facultades nativas del QL. El nuevo modo de gráficos convierte el modo 8 (256 x 256 puntos) en un modo gráfico de **baja resolución** usando motivos gráficos que ocupan un cuadratín de 8 x 8 puntos, obteniendo por tanto 32 líneas de 32 columnas cada una, en pantalla. Lo que es sorprendente es que la creación de este nuevo modo de gráficos exige sólo dos procedimientos: **prepa_grafo,** usado para definir la forma de los motivos gráficos, y **porte_grafo** usado para llevar a un determinado lugar de la pantalla un motivo gráfico concreto. La idea fundamental es que en **prepa_grafo** se almacena el patrón o modelo de bits necesario para describir la forma y color del gráfico definido en una tabla adecuada, y luego **porte_grafo** simplemente transfiere esos bits de la tabla a la pantalla proyectando sobre ella las correspondientes motas de color.

La sintaxis del procedimiento **prepa_grafo** es:

```
prepa_grafo s,f,b,r0,r1,r2,r3,r4,r5,r6,r7
```

donde **s** es el número identificativo del motivo gráfico que se está definiendo; **f** es el color del frente, **b** es el color del fondo, y **r0** a **r7** son números en base 10 equivalentes a las 8 rayas de motas que describen ese gráfico. Observa que en la definición el color tanto del frente como del fondo está prefijado y no puede ser cambiado por un uso posterior por un comando de tinta INK o de papel PAPER. Eso tiene la ventaja de incrementar la velocidad con la que el 'grafo' puede ser llevado a la pantalla, y la desventaja de tener que utilizar una nueva descripción de un motivo gráfico cuando se requiera otro que teniendo la misma forma tenga una combinación diferente de color para el frente/fondo.

La manera en que los números **r0** hasta **r7** describen bit a bit una determinada raya horizontal, ha de ser familiar para cualquiera que haya usado otro ordenador personal, por ejemplo el Spectrum. Si escribes la combinación concreta de ocho motas usando 0 para representar las de color del fondo, y 1 para las del color del frente, ese número **binario** puede ser expresado en base 10, asignando a cada uno de los bits el peso que le corresponde según su posición y sumando los resultados obtenidos. Los pesos de acuerdo con cada posición del punto son:

```
mota más a la izquierda                    mota más a la derecha
 |128 | 64 | 32 | 16 | 8 | 4 |  2 | 1
```

de manera que por ejemplo, una raya concreta que sea:
`*---****`

si representamos por el asterisco (*) la mota del color del frente y con el guión (-) la mota con color del fondo, o representado en la forma normal:
10001111

puede ser transformado al número equivalente en base 10 simplemente con la operación:

128x1+64x0+32x0+16x0+8x1+4x1+2x1+1x1

o después de haberla evaluado, el **143.** Desde luego, para describir un motivo gráfico completo este proceso ha de ser repetido para cada una de las ocho rayas horizontales que lo definen. Por ejemplo, para definir dos motivos gráficos, un hombrecito y un bloque sólido, debe apelarse por dos veces al procedimiento **prepa_grafo** en la forma siguiente:

```
prepa_grafo 0,7,0,24,24,126,24,60,102,195,195
prepa_grafo 1,2,0,255,255,255,255,255,255,255,255
```

Con la primera definimos el motivo gráfico identificado por 0 como un hombrecillo en blanco sobre un fondo en negro, mientras que el segundo corresponde a un cuadratín totalmente relleno de rojo.

Una vez que se ha descrito un motivo gráfico cualquiera, puede ser situado en cualquier parte de la gradilla 32 x 32 de la pantalla, recurriendo a:

```
porte_grafo x,y,s
```

donde **x** e **y** representan las coordenadas horizontal y vertical respectivamente, y **s** es el número identificativo del motivo gráfico ya definido que va a ser llevado a pantalla o expuesto. Las filas de cuadratínes en la pantalla están numeradas a partir de la parte superior, comenzando con 0, e igualmente las columnas de cuadratínes se numeran a partir de la izquierda y comenzando por 0. De manera que:

```
porte_grafo 0,0,1
```

nos llevaría nuestro motivo gráfico 1 a la esquina superior izquierda y

```
porte_grafo 31,31,0
```

nos expondría el hombrecillo grafo 0, en la esquina inferior derecha.

Además de definir y recurrir a los procedimientos **prepa_grafo** y **porte_grafo** también es necesario reservar espacio en memoria para la tabla descriptiva de los motivos gráficos empleados en los procedimientos, y saber además la dirección de la base de dicha tabla. Sin embargo, eso no es difícil: si un programa está usando n motivos gráficos que cada uno exige 32 octetos para su definición, la sentencia de asignación

```
C_TAB=RESPR(32*N)
```

ejercerá esa **función,** siendo C_TAB el nombre dado a la tabla usada por los procedimientos.

Los dos procedimientos mencionados, **prepa_grafo** y **porta_grafo** son realmente todo lo necesario para aprovechar el nuevo modo de gráficos implantado. Desde luego, tienes que recordar que hay que incluir dichos procedimientos en tus programas, pero aparte de eso, el nuevo modo de baja resolución es tan fácil de usar como los modos de alta resolución intrínsecos del QL.

La siguiente sección da el listado y una breve explicación de ambos procedimientos; si no estás interesado en cómo funciona sáltate simplemente esas descripciones pero no olvides incluir los procedimientos en los programas presentados en los siguientes capítulos.

Procedimientos prepa_grafo y porte_grafo

El formato de la información que ha de almacenarse en memoria para producir cualquiera de las rayas horizontales de 8 motas de color, se explica en la "QL User Guide". Esencialmente, el color de un grupo de cuatro motas consecutivas está controlada por los dieciséis bits de un doble octeto (una palabra sencilla) tal y como se muestra a continuación:

```
pixel     0     1     2     3     0     1     2     3
        G F G F G F G F R B R B R B R B
bit     15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0
```

donde la G representa el verde (green), la F el parpadeo (flash), la R el rojo (red) y la B el azul (blue), con lo que el aspecto de cada mota de color observada en pantalla viene determinada por 4 bits. Por ejemplo, el pixel 1 está controlado por los valores de los bits 13, 12, 5 y 4, pertenecientes a un doble octeto de la parte de la memoria de usuario dedicada a la pantalla. Usando la anterior y sabiendo además que la memoria de pantalla comienza a partir de la dirección 131072 (que es la que corresponde a los cuatro pixels situados en la esquina superior izquierda de la pantalla) y continúa a partir de ahí según el orden en que se efectúa el recorrido de la pantalla según las líneas de la imagen, es relativamente fácil escribir el procedimiento **prepa_grafo** usando únicamente SuperBASIC. Sin embargo, y dado que involucra bastante manipulación de los bits de la pareja de octetos, el procedimiento resultante es precioso pero algo difícil de comprender:

```
1300 DEFine PROCedure prepa_grafo(S%,F%,B%,
     R0%,R1%,R2%,R3%,R4%,R5%,R6%,R7%)
1310 LOCal A,FH%,FL%,BH%,BL%
1320 A=C_TAB+32*S%
1330 FH%=(F% DIV 4)*2
1340 FL%=F% && 3
1350 BH%=(B% DIV 4)*2
1360 BL%=B% && 3
1370 prepa_raya(R0%)
1380 prepa_raya(R1%)
1390 prepa_raya(R2%)
1400 prepa_raya(R3%)
1410 prepa_raya(R4%)
1420 prepa_raya(R5%)
1430 prepa_raya(R6%)
1440 prepa_raya(R7%)
1499 END DEFine prepa_grafo
1500 DEFine PROCedure prepa_raya(R%)
1510 LOCal M%,J,I,DL%,DH%
```

```
1520 M%=128
1530 FOR J=1 TO 2
1540  DL%=0:DH%=0
1550  FOR I=1 TO 4
1560   IF (R% && M%)=M% THEN
1570    DL%=FH%+DL%*4
1580    DH%=FL%+DH%*4
1590   ELSE
1600    DL%=BH%+DL%*4
1610    DH%=BL%+DH%*4
1620   END IF
1630   M%=M% DIV 2
1640  END FOR I
1650  POKE A,DL%
1660  POKE A+1,DH%
1670  A=A+2
1680 END FOR J
1699 END DEFine prepa_raya
```

Este procedimiento **prepa_grafo** confecciona primero las combinaciones de bits que corresponden a los colores del frente (f=foreground) y del fondo (b=background) en las líneas 1330 a 1360; y luego recurre al procedimiento **prepa_raya** para amoldar cada raya horizontal del motivo gráfico descrito, al correspondiente doble octeto que se necesita reflejar en la tabla descriptiva de los motivos gráficos a usar en el programa.

El procedimiento **porte_grafo** es mucho más simple que los anteriores:

```
1700 DEFine PROCedure porte_grafo(X%,Y%,S%)
1710 LOCal A,B,I
1720 A=C_TAB+32*S%
1730 B=131072+(X%+Y%*256)*4
1740 FOR I=0 TO 7
1750  POKE_L B,PEEK_L(A+I*4)
1760  B=B+128
1770 END FOR I
1799 END DEFine porte_grafo
```

En la línea 1720 se calcula la dirección de comienzo de la subtabla correspondiente al motivo gráfico identificado como S%; en la línea 1730 se calcula la dirección en la memoria de pantalla que corresponde a la fila y columna en que queremos aparezca el motivo gráfico; y finalmente, en la línea 1750 transfiere ese octeto desde la tabla descriptiva del motivo gráfico hasta la memoria de pantalla. Eso se repite dentro de un bucle preconfinado FOR (líneas 1740 a 1770) para cada una de las ocho rayas horizontales que describen el cuadratín ocupado por el motivo gráfico.

Esta versión de **porte_grafo** desde luego funciona, pero para la mayoría de los programas de juegos es demasiado lenta. La única solución a este problema es recurrir a la programación en código máquina del microprocesador 68000. La conversión no es difícil, simplemente laboriosa.

```
0       -5245   ASL #5,D3               E9 83
2       -10109  ADD D3,D4               D8 83
4       -7806   ASL #8,D2               E1 82
6       -11646  ADD D2,D1               D2 80
8       -6783   ASL #2,D1               E5 81
10      1665    ADDI #131072,D1         06 81
12      2                               00 02
14      0                               00 00
16      8769    MOVEA D1,A1             22 41
18      8316    MOVEA #0,A0             20 7C
20      0                               00 00
22      0                               00 00
24      9264    LOOP MOVE 0(A0,D4),D2   24 30
26      18432                           48 00
28      8834    MOVE D2,(A1)            22 82
30      -11524  ADDA #128,A1            D2 FC
32      128                             00 80
34      22664   ADDQ #4,A0              58 88
36      -20228  CMPA #32,A0             B0 FC
38      32                              00 20
40      26350   BNE LOOP                66 EE
42      28672   MOVEQ 0,D0              70 00
44      20085   RTS                     4E 75
```

Para usar este programa en código máquina en lugar del procedimiento **porte_grafo,** basta simplemente usar la instrucción de apelación a un programa en código máquina, de palabra clave CALL=cite, reclame, llame. Pero desde luego, previamente ha tenido que cargarse en alguna parte de la memoria este programa en código máquina. La nueva versión del procedimiento **porte_grafo** sería pues:

```
1700 DEFine PROCedure porte_grafo(X%,Y%,S%)
1710 CALL dire_rus,X%,Y%,S%,C_TAB
1799 END DEFine porte_grafo
```

y además un cargador de programas en código máquina (incluyendo el propio código máquina) que podría ser:

```
360 DATA -5245,-10109,-7806,-11646
370 DATA -6783,1665,2,0,8769,8316
380 DATA 0,0,9264,18432,8834,-11524,128
390 DATA 22664,-20228,32,26350,28672,20085
400 dire_rus=RESPR(100)
410 RESTORE 360
420 FOR I=dire_rus TO dire_rus+22*2 STEP 2
430 READ B
440 POKE_W I,B
450 END FOR I
```

Es importante observar que el procedimiento **prepa_grafo** y el procedimiento **porte_grafo** en la versión que incluye la cesión a la subrutina en código máquina que comienza en la dirección dada por **dire_rus**, se usan en todos los programas de este libro y aunque se incluyen en el listado final al término de cada capítulo, no siempre se vuelven a listar a medida que se desarrollan los programas.

Observación del color de las motas

Una faceta importante en muchos programas para juegos es la posibilidad de hallar el color que presenta una determinada mota en pantalla. Desafortunadamente, a SuperBASIC le falta esa función en el lenguaje, pero también aquí esta deficiencia es fácil de rectificar. Sabiendo cómo se trazan las imágenes en pantalla a partir de la zona de memoria correspondiente, es fácil de definir una función que permita evaluar el color de cualquier 'pixel' concreto:

```
9500 DEFine FuNction tinte(R%,C%,X%,Y%)
9510 LOCal A%,P%,B%,A
9520 P%=C%*8+X%
9530 A=131072+(512*R%+Y%*64+P% DIV 4)*2
9540 B%=PEEK(A+1)
9550 A%=PEEK(A)
9560 P%=P% MOD 4
9570 IF P%=3 THEN RETurn ((2 && A%)*2)+
     (3 && B%)
9580 IF P%=2 THEN RETurn ((8 && A%) DIV 2)+
     ((12 && B%) DIV 4)
9590 IF P%=1 THEN RETurn ((32 && A%) DIV 8)+
     ((48 && B%) DIV 16)
9600 IF P%=0 THEN RETurn ((128 && A%) DIV 32)+
     ((192 && B%) DIV 64)
9699 END DEFine tinte
```

La función **tinte** así definida nos dará un valor que guarda relación con el color de la mota que dentro del cuadratín situado en pantalla en la columna X% y la fila Y%, está colocada dentro de la raya horizontal R% en el cruce con la raya vertical C%. En otras palabras, hay dos etapas implicadas al especificar la mota cuyo color se quiere saber -primeramente, el cuadratín de la pantalla donde está contenida, y en segundo lugar, su situación dentro de ese cuadratín. Así por ejemplo:

```
tinte(10,11,5,4)
```

nos dará el **color** de la mota señalada por el cruce de la línea vertical 5 con la línea horizontal 4, dentro del cuadratín que ocupa la décima fila y la undécima columna de la gradilla de pantalla. Esta función **tinte** se usa en todos los programas de este libro y es importante destacar que aunque está incluida en el listado final de cada programa, no se repite dentro del capítulo.

Técnicas para juegos

Antes de comenzar realmente con el primer programa de este libro, merece la pena dar una panorámica de lo que trata cada juego del libro y la técnica que ilustra. El juego del Capítulo Dos 'Hormiguero' es un ejemplo simple de animación en una dimensión y sirve para presentar los motivos gráficos y los bucles de animación. También es nuestro primer ejemplo de usar rutinas en lenguaje máquina dentro de un programa en BASIC. El Capítulo Tres, con el juego de las 'Caza Ranas', avanza hasta una animación completa en dos dimensiones y comenta cómo controlar un objeto que rebota alrededor de la pantalla implicando el efecto de la gravedad. El Capítulo Cuatro usa una clase de animación completamente diferente, la animación por deslizamiento de la imagen para producir una versión personal de un juego muy conocido de 'Ranas Urbanas'. Los Capítulos Cinco y Seis tratan de uno de los más interesantes motivos gráficos alargados que pueden ser animados con facilidad: las 'culebras'. El Capítulo Cinco desarrolla el juego básico, trata con los problemas de animar eficazmente ese motivo de la culebra, y presenta la idea de las **colas de datos.** El Capítulo Seis amplía el juego para incluir un número de motivos gráficos adicionales y el resultado lo hemos denominado 'renacuajos'. Para hacer las cosas lo suficientemente rápidas, el Capítulo Seis presenta una subrutina en código máquina muy amplia que reemplaza con mayor eficacia uno de los procedimientos en BASIC usados dentro del juego. El Capítulo Siete nos lleva al juego tradicional de culebras y escaleras, y efectúa una versión actualizada para los ordenadores de hoy. La técnica principal explicada en este capítulo es la manera en que puede animarse un motivo gráfico sobre una escena de fondo complicada, sin destruir dicha imagen de fondo cada vez que se mueve el motivo.

Cada uno de los juegos presenta algo nuevo en cuanto a las técnicas de animación y al uso de tu ordenador, y el capítulo final combina todo esto con comentarios sobre cómo diseñar e implementar juegos en general y cómo usar el QL en particular. Para el momento en que alcances ese capítulo final, deberás haber aprendido una gran cantidad de cosas sobre ambos temas y habrás disfrutado bastante a lo largo del camino recorrido con este libro.

Capítulo Dos

Hormiguero

Hormiguero es un sencillo pero efectivo programa que involucra la animación de unos cuantos objetos. La idea básica del juego es guiar un hombre a través de túneles que pertenecen a una colonia de hormigas termitas, con la meta de alcanzar y destruir el nidal de huevos que tienen localizado en el punto más profundo del hormiguero. La dificultad de esta labor se incrementa por tener que trabajar dentro de un límite de tiempo y evitar hormigas soldado situadas en cada nivel de los túneles. El problema mayor al implementar un juego de esta suerte es el de animar un gran número de objetos, el hombre y todas las hormigas soldado, al mismo tiempo. Además de tratar con este problema particular, este capítulo también desarrolla algunos de los métodos standard que se usarán sin comentarios adicionales en los capítulos subsiguientes.

El diseño del juego

Antes de comenzar a escribir cualquier programa, es una buena idea intentar especificar con el mayor detalle posible lo que debe hacer. Los programas para juegos son ligeramente diferentes a otras aplicaciones informáticas en que habitualmente no es posible dar un esquema exacto del juego final antes de haber implementado parte del juego. La razón para ello es que es muy difícil predecir cómo los elementos de un juego funcionarán sin ensayarlos. Después de haber escrito un gran número de programas para juegos es todavía difícil predecir el efecto global de combinar diferentes elementos tomados de juegos existentes para producir uno nuevo. Sin embargo, todavía merece la pena especificar lo que se espera que un juego haga antes de comenzar a escribir nada, por la simple razón de que así te da tiempo a eliminar cualquier cambio importante de tu sistema.

El diseño de Hormiguero (y muchos otros juegos dinámicos) cae con naturalidad en tres partes:

1. el gráfico que sirve de escena de fondo,
2. los motivos gráficos móviles y las reglas de movimiento, y
3. las consecuencias de ganar y perder.

La escena de fondo para Hormiguero consiste en unos cuantos túneles horizontales conectados por pozos verticales (véase Fig. 2.1). La razón de usar una combinación de uno o dos pozos para conectar los túneles horizontales es que promete dar una mayor variedad en las estrategias. Cuando sólo hay un pozo conectando dos niveles, sólo hay una posible ruta y el desarrollo del juego se convierte en un tema de temporización; pero cuando hay dos pozos, el jugador tiene la oportunidad de elegir cuál le interesa para bajar.

El trazado general de la escena sugiere que vamos a necesitar tres colores: uno para los túneles, probablemente negro; otro para la tierra, probablemente rojo; y un tercero para el cielo situado encima, casi con toda certeza, azul.

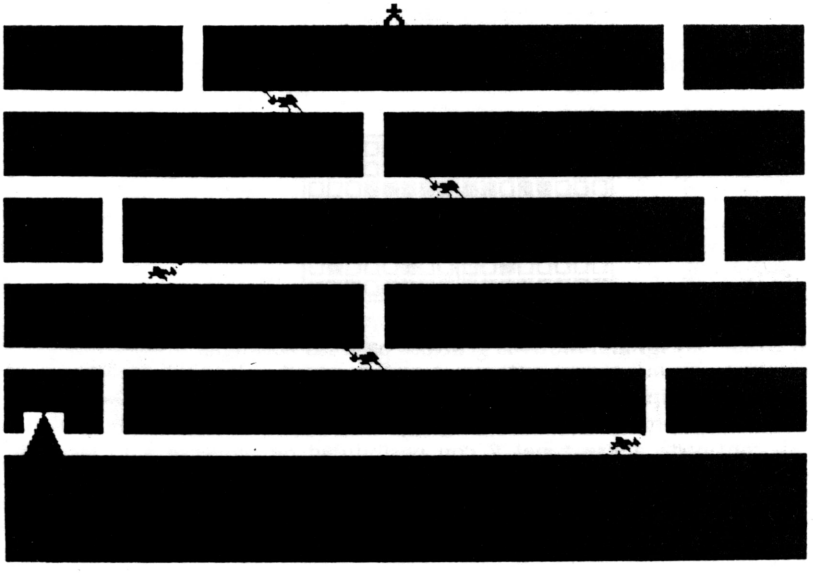


Fig. 2.1

La forma del hombre se implementa fácilmente como un **motivo gráfico** definido por el usuario y **sencillo** (véase Fig. 2.2). Usando un motivo que ocupe un solo cuadratín significa que los túneles y los pozos que los conectan, sólo tienen que tener un cuadratín de ancho, que es el espacio ocupado por cualquier carácter. Sin embargo, las formas dadas a las hormigas son mucho más difíciles de implementar en un solo cuadratín de 8 x 8 motas, porque son bastante largas y delgadas. La solución es usar motivos gráficos **dobles** definidos por el usuario (véase Fig. 2.3). El hecho de que las hormigas sólo se mueven horizontalmente a lo largo de los túneles y nunca suben o bajan por los pozos, significa que incluso aunque estén compuestas por dos cuadratínes, los túneles y pozos sólo necesitan tener un cuadratín de ancho.

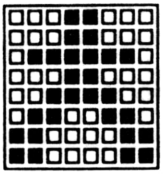


Fig. 2.2. Motivo gráfico para el hombre

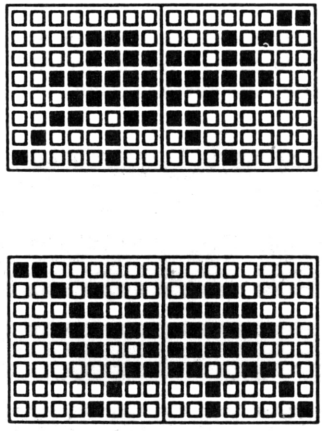


Fig. 2.3. Motivos gráficos para las hormigas

Las reglas de movimiento para las hormigas son fáciles de definir. En cada nivel hay una hormiga confinada a ese túnel y con posibilidad de moverse a lo largo de él a derecha o a izquierda. Si sucede que una hormiga llega a la posición corriente ocupada por el hombre, entonces el juego se termina y el hombre es llevado abajo al nidal para ser comido. En esta etapa es difícil decir exactamente cómo deben moverse las hormigas para producir un buen juego, pero parece razonable comenzar con la hipótesis de que deben moverse **aleatoriamente** a lo largo del túnel y ocasionalmente cambiar su dirección, también **al azar**. Esta lección de "merodeos aleatorios de las hormigas" deben proporcionar suficiente dificultad para el jugador en cuanto el límite de tiempo para completar su trayecto es lo suficientemente corto.

Las reglas de movimiento para el hombrecillo son un poco más complicadas. Al igual que las hormigas, debiera permitírsele mover a lo largo de los túneles pero también bajar por los pozos. El juego probablemente sería demasiado fácil si se le permitiera retroceder subiendo por los pozos, por lo que el control del jugador sobre el hombrecito está limitado a las teclas de flecha **izquierda** y **derecha** para el movimiento horizontal y la tecla de flecha **abajo** para el movimiento vertical. Si se pulsa la tecla de flecha abajo cuando el hombre está encima de un pozo, entonces deberá bajar de un salto hasta el siguiente nivel. Cuando el hombre no está situado sobre un pozo, la tecla de flecha hacia abajo no tiene ningún efecto. Aparte de detalles tales como asegurarte que ni las hormigas ni el hombre pueden salirse por el borde de la pantalla, aquí se completa la descripción de cómo deben moverse.

Hay tres maneras posibles en que el juego puede terminar:

1. el hombre alcanza el nidal de huevos,
2. una hormiga captura al hombrecillo, o
3. se sobrepasa el límite de tiempo y de los huevos brotan más hormigas.

Aunque no hay ninguna duda que un juego con éxito tiene que ser excitante, también hay un amplio panorama para construir terminaciones interesantes.

Por ejemplo, en este caso, cuando el hombre alcanza el nidal podría ser recompensado con una alegre fanfarria de trompetas; cuando una hormiga captura al hombre podría arrastrarla hasta el nidal, y cuando se alcanza el límite de tiempo podría quedar señalizado por una explosión demográfica de la población hormigüela.

Después de la especificación del juego, es casi el momento de comenzar a escribir el programa, pero primero merece la pena repasar alguna de las ideas y dificultades de la animación de motivos gráficos.

La técnica de animación

La animación en el ordenador de pequeñas figuras es la simplicidad por sí misma. Para hacer que algo aparente moverse, todo lo que tienes que hacer es exponerlo repetidamente en una posición, luego borrarlo (habitualmente mediante la exposición de un espacio en blanco) y luego volver a exponerlo en su nueva posición. Si puedes repetir este ciclo de "exponer-borrar-exponer" lo suficientemente rápido, crearás la ilusión de un movimiento bastante suave.

En la práctica, hay dos factores que controlan la suavidad aparente del movimiento generado por el ordenador. La primera es el tiempo entre el borrado y la re-exposición del motivo. Eso lo llamaremos 't1', y debiera ser lo más corto posible porque es la longitud del tiempo en que la figura **no es visible** en pantalla. Si 't1' es demasiado largo, entonces el objeto parece que "titila" o incluso que "parpadea". El segundo factor es el lapso de tiempo entre dos sucesivas exposiciones del motivo gráfico. Es lo que denominaremos 't2' y puede ser tan largo como deseemos, porque controla la velocidad a la que el objeto aparenta moverse. En la mayoría de los casos querremos que el objeto se mueva rápidamente, y por tanto 't2' también necesita ser corto, pero no es siempre así. La manera en que los diferentes intervalos de tiempo afectan a la animación puede verse en la Fig. 2.4.

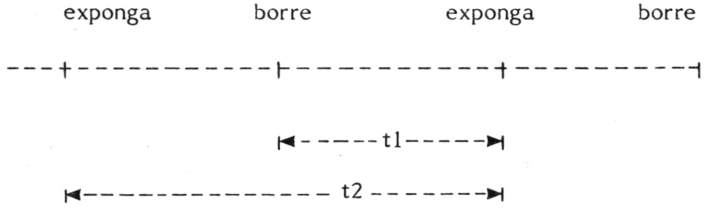


Fig. 2.4. Temporización al animar motivos gráficos

Hay otro factor que controla la velocidad a la que algo se desplaza y que ha sido ignorado hasta ahora. Se ha supuesto que cada vez que el objeto se mueve en la pantalla, se mueve en pasos lo más pequeños posible -es decir, el espacio del cuadratín que ocupa cualquier carácter. Moviendo el objeto de cuadratín en cuadratín, tiene la ventaja de que el movimiento aparece lo menos "espasmódico" posible, y que el objeto realmente se expone en cada posición por la que parece atravesar. Por otro lado, hay una gran ventaja en mover el objeto con zancadas que ocupen más de un cuadratín cada vez -¡la velocidad! A menudo es posible hacer que un objeto aparente moverse bastante rápido, usando únicamente BASIC, pero haciendo que se desplace más de un cuadratín en cada paso. En Hormiguero, ambos métodos de animación serán usados. El hombre se moverá únicamente un cuadratín en cada paso, pero las hormigas algunas veces se moverán mayores distancias de una sola zancada.

El ciclo "exponer-borrar-exponer" es el fundamento de la animación de motivos gráficos pero todavía deja el problema de cómo seguir el rastro de unos cuantos objetos que se mueven dentro de un programa. Cada objeto en la pantalla está asociado con cinco magnitudes y de aquí con cinco variables: la **forma** característica del cuadratín que ocupa; su **posición** corriente en virtud de sus coordenadas de pantalla X e Y; y los **incrementos** que sus coordenadas X e Y deben sufrir en cada ronda del bucle de animación. Estas dos magnitudes pueden ser pensadas como **desplazamientos** o incluso como **velocidades.** Porque son las que gobiernan el **paso en cada dirección** que el objeto da cada vez que se efectúa una ronda del bucle de animación. Por ejemplo, la forma de un objeto puede estar definida como el carácter gráfico S, su posición como X e Y y su velocidad como VX y VY. Su bucle de animación sería:

```
porte_grafo X,Y,B        :' borre objeto
X=X+VX:Y=Y+VY            :' actualice coordenadas
porte_grafo X,Y,S        :' exponga objeto
```

representando el objeto gráfico 'B' un blanco con el color correcto para el fondo (véase Capítulo Uno). En la práctica, los bucles de animación son generalmente un poco más complicados porque la velocidad del objeto habitualmente cambia como consecuencia de dónde está dentro de la pantalla. Por ejemplo, la animación de una bola que rebota implicaría cambiar el signo de la velocidad siempre que la pelota rebotara contra los límites de la pantalla.

Esa es toda la teoría que hay sobre la animación sencilla. Un objeto que está animado de la manera descrita anteriormente, es lo que se denomina **motivo** gráfico, y se conoce también como 'sprite' aunque no tiene nada de 'espíritu'. El programa Hormiguero usa un total de seis motivos gráficos, cinco hormigas y un hombrecillo, y es una buena demostración del hecho que en computación ¡la teoría a menudo no es más que una mera directriz!

El programa principal

El programa principal de todos los juegos dinámicos parece ser que siempre es el mismo. Usualmente hay algunas labores de **inicialización** -establecimiento de las condiciones iniciales- seguido de un bucle de animación y luego rutinas que llevan a cabo la tarea de terminar el juego.

Hormiguero no es ninguna excepción:

```
 10 REMark HORMIGUERO
 20 MODE 8
 25 WINDOW 512,256,0,0
 26 prologue
 30 prepa_juego
 35 REPeat manga
 36  prepa_manga
 50  trace_escena
 60  REPeat movida
 70   mueva_ambos
 80   IF TIEMPO OR GOLPE OR NIDAL THEN EXIT movida
 90  END REPeat movida
100  fine_manga
110  IF OTRA=FALSO THEN EXIT manga
120 END REPeat manga
130 INK 0
140 PAPER 7
150 CLS
199 STOP
```

Las líneas 26 a 50 son citas directas al procedimiento concernientes a conseguir la preparación del juego; las líneas 60 a 80 forman el bucle de animación de los objetos y las líneas 90 a 199 tratan con el final del juego. El procedimiento **prologue** simplemente expone un título enmarcado y establece el nivel de dificultad del juego así como las instrucciones. El procedimiento **prepa_juego** implanta el código máquina usado por el procedimiento **porte_grafo** (véase Capítulo Uno) y los motivos gráficos usados por el resto del programa. El procedimiento **prepa_manga** establece las dimensiones de las tablas y los valores iniciales de las variables que lo necesiten. Observa que mientras el procedimiento **prepa_juego** se efectúa únicamente una vez cuando empieza el programa, el procedimiento **prepa_manga** se ejecuta cada vez que el jugador solicita otra jugada, o sea, otra 'manga'. El procedimiento **trace_escena** está pensado para dibujar el sistema de túneles y pozos preparando así el escenario del juego.

Podríamos decir que el corazón del programa es el procedimiento **mueva_ambos** que es el responsable de efectuar un movimiento tanto para el hombre como para las hormigas. Este procedimiento se repite indefinidamente hasta que una de las tres variables TIEMPO, GOLPE o NIDAL, adopta el valor CERTO. Estas tres variables se usan para indicar las tres razones posibles para que la manga alcance un final y sus valores se establecen mediante comprobaciones efectuadas dentro del procedimiento **mueva_ambos.** Observa que cualquier variable numérica puede usarse para asignar estos dos valores de verdad CERTO o FALSO, que se prefijan dentro del procedimiento de preparación del juego, y que pueden por tanto ser usadas para traspasar entre procedimientos el resultado de una comparación. Por ejemplo, el procedimiento **fine_manga** no sólo entrega un final apropiado a cada jugada, sino que también pregunta al jugador si va a jugar otra vez y entrega el resultado en la variable OTRA, dándole el valor CERTO para indicar que sí y FALSO para indicar que no. Todo lo que queda ahora por hacer es escribir cada uno de los procedimientos usados por el programa principal.

Procedimientos prepa_juego, prepa_manga y prologue

Los tres primeros procedimientos del programa no fueron muy difíciles de escribir. Sin embargo, los procedimientos **prepa_juego** y **prepa_manga** cambiaron durante la confección del resto del programa. La razón para eso fue que no estaba totalmente claro las variables que tenían que ser inicializadas hasta que se hubieran escrito los otros procedimientos. En lugar de presentarte las versiones primitivas de estos procedimientos y luego introducir en ellas todos los cambios a medida que se explican los otros procedimientos, es menos confuso darla simplemente en su forma final.

```
350 DEFine PROCedure prepa_juego
360 DATA -5245,-10109,-7806,-11646
370 DATA -6783,1665,2,0,8769,8316
380 DATA 0,0,9264,18432,8834,-11524,128
390 DATA 22664,-20228,32,26350,28672,20085
400 dire_rus=RESPR(100)
410 RESTORE 360
420 FOR I=dire_rus TO dire_rus+22*2 STEP 2
430 READ B
440 POKE_W I,B
450 END FOR I
520 C_TAB=RESPR(32*12)
530 prepa_grafo 0,3,0,0,14,15,63,31,51,68,132
540 prepa_grafo 1,3,0,3,20,216,252,168,192,64,16
550 prepa_grafo 2,3,0,0,112,248,252,248,204,34,33
560 prepa_grafo 3,3,0,192,40,27,63,25,3,4,8
570 prepa_grafo 4,6,0,24,24,126,24,60,102,195,195
580 prepa_grafo 5,4,0,1,1,3,3,7,7,15,15
590 prepa_grafo 6,4,0,31,31,63,63,127,127,255,255
600 prepa_grafo 7,4,0,128,128,192,192,224,224,240,240
610 prepa_grafo 8,4,0,248,248,252,252,254,254,255,255
620 prepa_grafo 9,0,0,255,255,255,255,255,255,255,255
630 prepa_grafo 10,1,1,255,255,255,255,255,255,255,255
640 prepa_grafo 11,6,1,24,24,126,24,60,102,195,195
650 FALSO=0
660 CERTO=1
670 OTRA=CERTO
699 END DEFine prepa_juego
```

La primera parte del procedimiento (líneas 360 a 450) carga en memoria el código máquina usado por **porte_grafo** en un área reservada de la memoria tal y como se describió en el Capítulo Uno. La segunda parte crea los motivos gráficos definidos por el usuario necesarios. La línea 520 reserva el espacio suficiente para 12 de esos motivos gráficos. Las líneas 530 y 540 definen los dos motivos gráficos que conjuntamente determinan la forma de una hormiga yendo hacia la derecha, y las líneas 550 y 570 hacen lo mismo para las hormigas que van hacia la izquierda. Es decir, detrás del procedimiento **prepa_juego**, citando:

```
porte_grafo X,Y,0
porte_grafo X+1,Y,1
```

se expondrá una hormiga mirando hacia la derecha en la posición X,Y; y citando

```
porte_grafo X,Y,3
porte_grafo X+1,Y,2
```

se expondrá una hormiga en X,Y mirando hacia la izquierda. La necesidad de usar dos formas diferentes para el motivo hormiga no ha sido mencionada hasta ahora, pero el hecho de que las hormigas se muevan a la izquierda o a la derecha lo hace absolutamente imprescindible. Si solamente se utilizara una forma para el motivo gráfico hormiga habría veces en que ésta parecería que se mueve hacia atrás como los cangrejos.

Puede que pienses que mantener el rastro de qué pareja de motivos gráficos ha de usarse para cualquier hormiga concreta será difícil, pero de hecho es bastante fácil como se hará patente cuando consideremos el procedimiento **mueva_hormi.** Las líneas 570 y 640 definen dos motivos gráficos en forma de hombrecillo. El identificado como 4 es un hombrecillo sobre un fondo negro, y el identificado como 11 es el mismo hombrecillo sobre un fondo azul. Obviamente, debemos usar en los túneles el identificado como 4, y el identificado como 11 ha de usarse al comienzo del juego cuando el hombrecillo está por encima de la tierra y por tanto tiene como fondo el cielo. Las líneas 580 a 610 definen otros cuatro motivos gráficos que encajados conjuntamente nos dan la forma de pirámide que representa el nidal de las hormigas. Es decir:

```
porte_grafo X,Y-1,5:porte_grafo X+1,Y-1,7
porte_grafo X,Y,6:porte_grafo X+1,Y,8
```

hará que se expongan los cuatro motivos que constituyen el nidal con su esquina inferior izquierda en las coordenadas X,Y. Finalmente, el motivo gráfico identificado como 9 y descrito en la línea 620, y el identificado como 10 descrito en la línea 630, son sólo cuadratínes en blanco con el color adecuado. El identificado como 9 es negro y se usará para dejar en 'blanco' los objetos dentro de los túneles, y el identificado como 10 es azul y también se usa para 'blanquear' los objetos por encima del terreno. Un resumen de estos motivos gráficos se puede ver en la Tabla 2.1.

Tabla 2.1. Motivos gráficos

Número	Forma	Frente	Fondo
0	trasera de hormiga a derecha	magenta	negro
1	frente de hormiga a derecha	magenta	negro
2	trasera de hormiga a izquierda	magenta	negro
3	frente de hormiga a izquierda	magenta	negro
4	hombre en túnel	amarillo	negro
5	superior izquierda de nidal	verde	negro
6	inferior izquierda de nidal	verde	negro
7	superior derecha de nidal	verde	negro
8	inferior derecha de nidal	verde	negro
9	espacio en negro	negro	negro
10	espacio azul	azul	azul
11	hombre en pozo	amarillo	azul

El procedimiento **prepa_manga** concierne a la creación y a los valores iniciales de variables usadas más adelante en el programa.

```
1000 DEFine PROCedure prepa_manga
1050 RANDOMISE
1060 XM%=RND(0 TO 20)+10
1070 YM%=2
1080 DIM A%(4,3),A1%(4,2)
1090 FOR I=0 TO 4
1100  A%(I,1)=RND(0 TO 25)+5
1110  A%(I,2)=(I+1)*5+2
1120  A%(I,3)=1
1130  A1%(I,1)=0:A1%(I,2)=1
1140 END FOR I
1160 OTRA=FALSO
1170 GOLPE=FALSO
1180 NIDAL=FALSO
1190 TIEMPO=FALSO
1299 END DEFine prepa_manga
```

Las líneas 1060 y 1070 fijan la posición inicial del hombre en XM% e YM%. La posición en vertical del hombre siempre está prefijada a 2 de manera que comience el juego por encima del terreno, pero su posición horizontal se elige aleatoriamente. Las líneas 1080 a 1140 inicializan las posiciones corrientes de las cinco hormigas soldado, su dirección de movimiento y su forma pertinente (a derecha o a izquierda). La tabla A%(4,3) se utiliza para retener la mayoría de la información sobre cada hormiga. A%(I,1) da la coordenada X de la hormiga I; A%(I,2) da la coordenada Y asociada y A%(I,3) da la dirección en que se mueve la hormiga. Si A%(I,3) es 1 entonces la hormiga I se mueve hacia la derecha, y si es -1 se mueve hacia la izquierda. Finalmente, los identificativos numéricos de la pareja de motivos gráficos que determinan la forma de la hormiga está conservado en A1%(I,2) por lo que para exponer la hormiga I (recuerda que I ha de estar en la banda 0 a 4) citaremos:

```
porte_grafo A%(I,1),A%(I,2),A1%(I,1)
porte_grafo A%(I,1)+1,A%(I,2),A1%(I,2)
```

La línea 1100 establece aleatoriamente la posición horizontal inicial de cada hormiga en la banda 5 a 30. La línea 1110 establece la posición inicial vertical de cada hormiga de manera que haya una por cada túnel. Es decir, la hormiga 0 se expone en la línea 7, la hormiga 1 en la línea 12 y la hormiga 2 en la línea 17, y así sucesivamente. Todas las direcciones iniciales del movimiento de las hormigas se hacen igual a 1, haciendo que todas se muevan a la derecha (línea 1120) y así la línea 1130 hace una forma de hormiga mirando a la derecha en las variables A1%(I,1) y A1%(I,2). La parte final de este procedimiento fija las variables que van a usarse para indicar el final del juego al valor FALSO (líneas 1160 a 1190).

El procedimiento **prologue** es la propia sencillez. Desde luego, escribir este procedimiento simplemente implica encontrar las palabras razonables para explicar el juego a un jugador novato.

```
8000 DEFine PROCedure prologue
8010 BORDER 0:INK 6:PAPER 0:CLS
8020 AT 3,14:PRINT 'H O R M I G U E R O'
8040 AT 7,3:PRINT 'en este juego tienes una
     cierta'
8050 PRINT TO 2;'cantidad de tiempo para destruir
     el nido de las hormigas'
8060 PRINT TO 2;'antes de que las hormigas resuciten'
8070 PRINT TO 2;'ATENCION: las hormigas soldado
     que guardan los'
8080 PRINT TO 2;'tuneles te capturaran y
     llevaran'
8090 PRINT TO 2;'al nido como alimento de sus crias'
8100 PRINT TO 2;buena suerte!'
8200 REPeat dif
8300  AT 18,5:PRINT 'dime nivel de dificultad -'
8310  PRINT TO 3;'1. Experto, 2. Medio
      3. Novato ';
8320  INPUT DF
8330  IF DF>0 AND DF<4 THEN EXIT dif
8335 END REPeat dif
8340 MAX=(20+DF*10)
8350 AT 22,15
8360 FLASH 1
8370 PRINT 'Por favor espera'
8380 FLASH 0
8399 END DEFine prologue
```

Las líneas 8300 a 8320 piden al usuario que elija el grado de dificultad del juego. La línea 8330 comprueba que DF está en la banda correcta y la línea 8340 asigna a MAX la cantidad de tiempo que el usuario está permitido para un grado de dificultad dado.

Procedimiento trace_escena

Este procedimiento es el primero que realmente hace muy mucho en la forma de gráficos. Su labor es dibujar el sistema de túneles y pozos preparando el comienzo del juego. Dado que sólo se usará una vez durante el juego, realmente no es crítico en cuanto a velocidad y nuestra preocupación principal es que al escribirlo debiera ser la de producir un procedimiento claro y conciso que sea fácil de modificar.

```
2000 DEFine PROCedure trace_escena
2010 PAPER 2
2020 CLS
2040 FOR X=0 TO 31
2050  porte_grafo X,7,9
2060  porte_grafo X,12,9
2070  porte_grafo X,17,9
```

```
2080  porte_grafo X,22,9
2090  porte_grafo X,27,9
2100 END FOR X
2120 FOR Y=0 TO 2
2125  FOR X=0 TO 31
2130  porte_grafo X,Y,10
2135  END FOR X
2140 END FOR Y
2160 pon_pozos 2,3,6
2170 pon_pozos 1,8,11
2180 pon_pozos 2,13,16
2190 pon_pozos 1,18,21
2200 pon_pozos 2,22,26
2220 porte_grafo XM%,YM%,11
2240 porte_grafo 1,26,5:porte_grafo 2,26,7
2250 porte_grafo 1,27,6:porte_grafo 2,27,8
2270 FOR I=0 TO 4
2280  porte_grafo A%(I,1),A%(I,2),A1%(I,1)
2285  porte_grafo A%(I,1)+1,A%(I,2),A1%(I,2)
2290 END FOR I
2300 SDATE 2000,1,1,0,0,0
2999 END DEFine trace_escena
```

Las líneas 2010 a 2020 clarean la pantalla a rojo y luego las líneas 2040 a 2100 trazan los túneles horizontales en negro. Eso crea cinco tiras negras a través del fondo rojo. Luego, las líneas 2120 a 2140 exponen un bloque de espacios azules para representar el cielo. Después de eso, todo lo que queda es trazar los pozos verticales que conectan los túneles. Eso es un problema que se resuelve mejor recurriendo a otro procedimiento -**pon_pozos** (NO,Y1,Y2)- que trazará tantos pozos verticales como indique NO, cada uno comenzando en Y1 y terminando en Y2. Así por ejemplo, **pon_pozos** (2,7,9) trazará dos pozos entre los túneles en Y=6 e Y=10. Las líneas 2160 a 2200 recurren a ese procedimiento, todavía por definir, para conectar los túneles mediante pozos.

La línea 2220 lleva el motivo gráfico hombre a las coordenadas XM%,YM% usando un frente rojo y un fondo azul. En esta etapa del juego, el hombre está por encima del terreno y por tanto necesita proyectarse sobre un fondo azul, pero posteriormente estará en los túneles y pozos y el color del fondo necesariamente será negro. Las líneas 2240 y 2250 llevan a la esquina inferior izquierda del túnel más inferior el nidal del hormiguero. Finalmente, las líneas 2270 a 2290 llevan las hormigas a escena y la línea 2300 coloca a 0 el **cronómetro** preparándolo para que el juego comience.

El único procedimiento citado por **trace_escena** es el de **pon_pozos** (NO,Y1,Y2) y éste ha de confeccionarse antes de pasar al resto del programa:

```
8400 DEFine PROCedure pon_pozos (NO,Y1,Y2)
8410 X1%=RND(0 TO 8)+6:X2%=RND(0 TO 10)+15
8420 FOR Y=Y1 TO Y2
8430  IF NO=1 THEN porte_grafo INT((X1%+X2%)/2),Y,9
8440  IF NO=2 THEN porte_grafo X1%,Y,9:porte_grafo X2%,Y,9
8450 END FOR Y
8499 END DEFine pon_pozos
```

La línea 8410 establece aleatoriamente las posiciones de los dos pozos en la variables X1% y X2%. Los números aleatorios se generan de manera tal que el pozo especificado por X1% está a la izquierda de la pantalla y el pozo especificado por X2% a la derecha. Si se requieren dos pozos, ambos serán pintados por la línea 8440 dentro del bucle preconfinado FOR (líneas 8420 a 8450). Si sólo se requiere un pozo se coloca en el valor medio de X1% y X2%, que es una posición aleatoria que tiende aproximadamente hacia la mitad de la pantalla (línea 8430).

Procedimiento mueva_ambos - el núcleo del juego

El procedimiento más importante en todo este programa es el procedimiento **mueva_ambos.** Como va a ser citado tantas veces, es importante que él y cualquier procedimiento al que recurra, trabaje lo más rápidamente posible. Habiendo dicho eso, todavía merece intentar mantener las cosas claras y simples y con esta meta la mejor manera es escribir el procedimiento **mueva_ambos** como una serie de citas o apelaciones directas a otros procedimientos:

```
4000 DEFine PROCedure mueva_ambos
4010 mueva_hombre
4020 IF GOLPE=FALSO THEN
4030  avance
4040  mire_hora
4050 END IF
4099 END DEFine mueva_ambos
```

La tarea del procedimiento **mueva_hombre** es la de permitir al jugador la oportunidad de desplazar al hombre y comprobar la posibilidad de que su trayecto haya alcanzado el nidal del hormiguero. Igualmente, el procedimiento **mueva_hormi** concierne al desplazamiento de las hormigas y a la comprobación de si el hombre ha sido capturado. El procedimiento **mire_hora** inspecciona el cronómetro y comprueba si ya ha concluido el límite de tiempo permitido.

Claramente, la manera en que se ha escrito el procedimiento **mueva_ambos** demora todas las decisiones difíciles hasta que se hayan confeccionado los otros procedimientos, pero esta estrategia es intencional: los problemas de programación deben siempre ser desglosados en pequeños pasos. El procedimiento **mueva_hombre** es el siguiente procedimiento a escribir:

```
5000 DEFine PROCedure mueva_hombre
5001 LOCal I%
5003 I%=KEYROW(1)
5004 IF I%=0 THEN END DEFine mueva_hombre
5010 IF YM%>4 THEN
5015  porte_grafo XM%,YM%,9
5020 ELSE
5022  porte_grafo XM%,YM%,10
5025 END IF
```

```
5060 IF I%=2 AND XM%>1 THEN XM%=XM%-1
5070 IF I%=16 AND XM%<30 THEN XM%=XM%+1
5080 IF I%=128 AND tinte(YM%+1,XM%,1,1)=0 THEN
     baje_pozo
5100 IF YM%>4 THEN
5102  porte_grafo XM%,YM%,4
5104 ELSE
5106  porte_grafo XM%,YM%,11
5107 END IF
5110 IF XM%=3 AND YM%=27 THEN NIDAL=CERTO
5199 END DEFine mueva_hombre
```

La línea 5010 examina la posición vertical del hombre. Si todavía está por encima de la tierra se lleva a pantalla un espacio azul en la línea 5022, y en el caso contrario un espacio de fondo negro en la línea 5015. Por tanto en ambos casos, el hombrecillo desaparece de la pantalla.

Las líneas 5060 a 5080 comprueban luego cuál de las tres posibles teclas de flecha es la pulsada. El valor correspondiente ha sido previamente guardado en I% al haber usado la función KEYROW (fila de tecla) en la línea 5003. La línea 5060 corresponde a la flecha izquierda, la línea 5070 a la flecha derecha y la línea 5080 a la flecha de bajada. El único otro aspecto digno de mención es que las líneas 5060 y 5070 también se aseguran que el movimiento intentado no saca al hombre fuera de la pantalla. La línea 5080 cita o apela directamente al procedimiento **baje_pozo** para hacer que el hombre salte al siguiente túnel únicamente si se ha pulsado la tecla de flecha hacia abajo **y si** también el cuadratín situado por debajo de la posición corriente del hombre es negro. La función **tinte** (véase Capítulo Uno) se usa para descubrir el color de ese cuadratín. El procedimiento **mueva_hombre** termina volviendo a llevar el motivo gráfico hombre a la nueva posición dada por XM%,YM% (líneas 5100 a 5107) y comprueba luego si ha llegado en su trayecto al nidal donde están los huevos de hormiga (línea 5110).

El único procedimiento que queda por abordar para que pueda moverse el hombrecillo es el procedimiento **baje_pozo** que será:

```
5500 DEFine PROCedure baje_pozo
5520 FOR I=1 TO 4
5530  YM%=YM%+1
5540  porte_grafo XM%,YM%,4
5550  BEEP .1,YM%
5560  porte_grafo XM%,YM%,9
5570 END FOR I
5575 YM%=YM%+1
5580 BEEP
5599 END DEFine baje_pozo
```

Las líneas 5520 a 5570 hacen que el hombre baje por el pozo una posición en cada paso. Cada paso va acompañado del sonido producido mediante la línea 5550.

El procedimiento **mueva_hormi** aborda el primer problema real que tiene que resolverse. La cuestión es que hay un hombre y cinco hormigas.

Si el procedimiento **mueva_hormi** moviera las cinco hormigas cada vez que fuera citado, entonces el hombrecillo parecería demasiado reacio a responder a las pulsaciones de teclas del jugador. La razón para eso es el tiempo que se tarda en desplazar las cinco hormigas. La solución obvia es mover únicamente una hormiga, elegida al azar, por cada posible paso del hombre. Eso es exactamente lo que el procedimiento **mueva_hormi** hace:

```
3000 DEFine PROCedure avance
3020 Q%=RND(1 TO 5)-1
3030 porte_grafo A%(Q%,1),A%(Q%,2),9:
     porte_grafo A%(Q%,1)+1,A%(Q%,2),9
3040 IF RND<.01 THEN vire Q%
3050 A%(Q%,1)=A%(Q%,1)+A%(Q%,3)*RND(1 TO 3)
3060 IF A%(Q%,1)<5 THEN
3061  A%(Q%,1)=5
3062  vire Q%
3065 END IF
3070 IF A%(Q%,1)>28 THEN
3071  A%(Q%,1)=28
3072  vire Q%
3075 END IF
3080 porte_grafo A%(Q%,1),A%(Q%,2),A1%(Q%,1)
3085 porte_grafo A%(Q%,1)+1,A%(Q%,2),A1%(Q%,2)
3090 GOLPE=tocando(Q%)
3099 END DEFine avance
```

La línea 3020 fija Q% aleatoriamente a un número entre 0 y 4 para determinar qué hormiga se moverá. Luego, la línea 3030 borra dicha hormiga de su corriente posición. La línea 3040 apela al procedimiento **vire(Q%)** aleatoriamente para invertir la dirección de la hormiga Q%. La función de azar RND será más pequeña que .01 como media, una vez en 100 citas al procedimiento **mueva_hormi.**

La línea 3050 incrementa o decrementa la coordenada X de la hormiga dependiendo del valor reflejado en A%(Q,3). Si A%(Q,3) es +1, entonces el valor de la función RND(1 TO 3) se añade a la coordenada X, de manera que la hormiga va hacia la derecha; si A%(Q,3) es -1 entonces el valor de la función RND(1 TO 3) se resta de la coordenada X, de manera que la hormiga va hacia la izquierda. La zancada que la hormiga da queda determinada por el valor de RND(1 TO 3), lo que significa que puede moverse 1, 2 ó 3 cuadratínes a la vez. Aunque en general el desplazamiento de más de un cuadratín a la vez es una fuente potencial de problemas, esa es realmente la única manera de conseguir la impresión de que las hormigas se mueven rápidamente. Después de haber actualizado la coordenada X de la hormiga, es comprobada en las líneas 3060 a 3075 está próxima a uno u otro borde de la pantalla. Si así es, se cita el procedimiento **vire(Q%)** para cambiar la dirección de la hormiga. Finalmente, las líneas 3080 y 3085 vuelven a exponer la hormiga en su nueva posición, y la línea 3090 aplica la función **tocando(Q%)** definida por el usuario para comprobar si la hormiga que se acaba de mover está tropezando con el hombre o no.

El procedimiento **vire(Q%)** es el único procedimiento usado en el anterior. Todo lo que tiene que hacer es cambiar la dirección del movimiento de la hormiga ya la pareja de caracteres gráficos que forman ese motivo.

```
3500 DEFine PROCedure vire (Q%)
3510 IF A1%(Q%,1)=3 THEN
3520  A1%(Q%,1)=0
3525  A1%(Q%,2)=1
3530 ELSE
3540  A1%(Q%,1)=3
3545  A1%(Q%,2)=2
3550 END IF
3560 A%(Q%,3)=-A%(Q%,3)
3599 END DEFine vire
```

La función **FNtocando** también es fácil de escribir:

```
8500 DEFine FuNction tocando (Q%)
8510 IF A%(Q%,2)<>YM% THEN RETurn FALSO
8520 IF A%(Q%,1)<>XM% AND A%(Q%,1)+1<>XM%
     THEN RETurn FALSO
8530 RETurn CERTO
8599 END DEFine tocando
```

El único procedimiento que resta por ahora es el procedimiento **mire_hora:**

```
4500 DEFine PROCedure mire_hora
4510 TIME$=DATE$
4520 SEC=TIME$(17)*60
4530 SEC=SEC+TIME$(19 TO 20)
4550 IF SEC>MAX THEN TIEMPO=CERTO
4999 END DEFine mire_hora
```

Las líneas 4510 a 4530 asignan el tiempo en segundos a la variable SEC y luego la línea 4550 compara para ver si ha concluido el tiempo permitido y el juego está en una de las formas de acabar.

Ahora que el procedimiento **mueva_ambos** y todos los procedimientos y funciones que en él se citan han sido definidos, se puede ensayar el juego aparte de las rutinas para final del juego. En esta parte del programa se han usado variables enteras para dar la máxima velocidad de ejecución.

Procedimiento fine_manga

La elección de cómo hacer que finalice una manga de un juego es algo que a menudo está limitado por la cantidad de tiempo que ya se ha empleado en conseguir que funcionen las rutinas principales del juego. Hay tres posibles conclusiones para Hormiguero y el procedimiento **fine_manga** tiene que identificar cada una de ellas y apelar a los procedimientos pertinentes.

```
6000 DEFine PROCedure fine_manga
6010 IF GOLPE THEN
6012  capture
6014 ELSE
6020  IF TIEMPO THEN
6022   trampa
6024  ELSE
6030   IF NIDAL THEN
6032    destru_nido
6040   END IF
6050  END IF
6060 END IF
6100 REPeat otra_manga
6110  AT 24,3
6112  PRINT 'OTRO JUEGO S/N ?';
6114  INPUT N$
6120  IF N$(1)='S' OR N$(1)='N' THEN
      EXIT otra_manga
6125 END REPeat otra_manga
6130 IF N$(1)='S' THEN OTRA=CERTO
6199 END DEFine fine_manga
```

El procedimiento **capture_hombre** se cita si una hormiga ha alcanzado la misma posición en pantalla que el hombre. El procedimiento **rebrote_hormis** se cita si el juego termina debido a haber alcanzado el límite de tiempo establecido, y el procedimiento **quite_nidal** si el hombre llega a esa posición dentro del límite de tiempo. Observa la manera en que se han anidado las instrucciones condicionales IF para elegir cuál es el procedimiento que debe citarse. De esta manera se asegura que no importa cuáles sean los valores contenidos en las variables GOLPE, TIEMPO o NIDAL, porque únicamente uno de los tres procedimientos será el citado. Después de este tramo, las líneas 6100 a 6130 preguntan al jugador si desea otra manga y fija la variable OTRA de acuerdo con la respuesta obtenida.

De los tres procedimientos mencionados en el **fine_manga** el más complicado es el de capturar al hombre, así que su descripción la dejaremos para lo último. El procedimiento de cómo hacer que rebroten las hormigas simplemente expone hormigas en posiciones aleatorias que se dirigen hacia el centro de la pantalla acompañadas de una melodía descendente.

```
6500 DEFine PROCedure trampa
6510 AT 23,3:PRINT 'LAS HORMIGAS ESTAN RESUCITANDO '
6520 FOR I=1 TO RND(0 TO 20)+20
6530  X=RND(0 TO 25)+5
6532  Y=RND(0 TO 15)+5
6535  T%=RND(0 TO 4)
6540  porte_grafo X,Y,A1%(T%,1)
6545  porte_grafo X+1,Y,A1%(T%,2)
6550  BEEP .1,I
6555  PAUSE 1
6560 END FOR I
6570 BEEP
6599 END DEFine trampa
```

El procedimiento **quite_nidal** recompensa al jugador con toda una 'fanfarria verbenera'.

```
6800 DEFine PROCedure destru_nido
6810 AT 23,2: PRINT 'LO LOGRASTE'
6815 RESTORE 6900
6820 REPeat melodia
6830  READ D,P
6840  IF D=999 THEN EXIT melodia
6850  BEEP D,P
6860  PAUSE D*125
6865  BEEP
6870 END REPeat melodia
6900 DATA .1,14,.1,13
6910 DATA .1,12,.1,11
6920 DATA .2,10,.2,10
6990 DATA 999,999
6999 END DEFine destru_nido
```

Las notas de la fanfarria se especifican por los valores de las instrucciones DATA (líneas 6900 a 6920). A cada nota le corresponde una pareja de valores, el primero dando la duración, y el segundo el tono. Luego, al marcar una duración de 999 se señala el final de la serie de notas, al ser detectada en la línea 6840.

La idea que el procedimiento **capture_hombre** implementa es que la hormiga que captura al hombre deberá arrastrarle hasta el pozo más cercano y dejarle caer para que sea recogido por la hormiga del túnel inmediatamente inferior. Eso se repite hasta que la hormiga del último nivel le arrastra hasta el nidal, donde el juego finalmente termina. Suena como una secuencia de acciones bastante complicadas, pero en la práctica no es tan difícil:

```
6200 DEFine PROCedure capture
6210 AT 23,3: PRINT 'Te han capturado!'
6220 REPeat hormi_jala
6230  IF Q%=4 THEN EXIT hormi_jala
6240  coja_hombre(Q%)
6250  busque pozos A%(Q%,2)
6260  IF ABS(X2%-A%(Q%,1))<ABS(X1%-A%(Q%,1))
      AND X2%<>0 THEN
6262   arrastre X2%
6264  ELSE
6266   arrastre X1%
6268  END IF
6270  porte_grafo A%(Q%,1)+2,A%(Q%,2),9
6275  porte_grafo A%(Q%,1)+3,A%(Q%,2),9
6280  BEEP .1,-10
6290  baje_pozo
6300  porte_grafo XM%,YM%,4
6310  Q%=Q%+1
6320 END REPeat hormi_jala
6330 coja_hombre(Q%)
6340 arrastre 5
6399 END DEFine capture
```

El procedimiento **coja_hombre(Q%)** (línea 6240) mueve la hormiga Q% hasta la corriente posición del hombre. Una vez que la hormiga haya agarrado al hombre, la línea 6250 emplea el procedimiento **busque_pozos** para localizar la posición de los mismos. Si sólo hay uno, esa posición es entregada como valor de X1%, y X2% se fija a cero. Si hay dos pozos que conecten un túnel con el siguiente debajo, sus posiciones son los valores entregados en X1% y X2%. Una vez que se hayan localizado los pozos, la línea 6260 comprueba cuál es el más cercano a la posición corriente de la hormiga que arrastra al hombre. Luego se aplica el procedimiento **arrastre** para mover ambos hasta el pozo más cercano. Finalmente, mediante el procedimiento **baje_pozo** se deja caer al hombre por el pozo en cuestión. Esta secuencia entera se repite hasta que haya sido arrastrado por cada hormiga y alcance el nivel inferior, donde la hormiga pertinente le llevará hasta el nidal (línea 6340).

Desde luego, la sencillez del procedimiento **capture_hombre** es debida a que usa unos cuantos procedimientos nuevos. El procedimiento **coja_hombre(Q%)** simplemente tiene que mover la hormiga desde su corriente posición hasta XM%. Sin embargo, antes de que esta hormiga comience a moverse tiene que estar en la dirección correcta.

```
5900 DEFine PROCedure coja_hombre (Q%)
5910 IF A%(Q%,1)+1=XM% THEN END DEFine coja_hombre
5920 IF SGN(XM%-A%(Q%,1)-1)<>A%(Q%,3) THEN
     vire Q%
5930 REPeat avance
5940  porte_grafo A%(Q%,1),A%(Q%,2),9
5945  porte_grafo A%(Q%,1)+1,A%(Q%,2),9
5950  A%(Q%,1)=A%(Q%,1)+A%(Q%,3)
5960  porte_grafo A%(Q%,1),A%(Q%,2),A1%(Q%,1)
5970  porte_grafo A%(Q%,1)+1,A%(Q%,2),A1%(Q%,2)
5975  demore
5980  IF A%(Q%,1)+1=XM% THEN EXIT avance
5990 END REPeat avance
5999 END DEFine coja_hombre
```

La línea 5920 comprueba si la hormiga ya está dirigida hacia el hombre; si así no es, se cita el procedimiento **vire(Q%).** Después de eso, el bucle de repetición (líneas 5930 a 5980) avanza la hormiga un cuadratín cada vez hasta que está en contacto con el hombre. La línea 5920 aplica la función de signo SGN(X) que dará -1 si X es negativo, +1 si X es positivo, y 0 si es cero. Esta función es standard en la mayoría de las versiones del BASIC pero parece que falta de SuperBASIC. Sin embargo, no es mucho problema ya que es fácil de añadirla:

```
8700 DEFine FuNction SGN(N%)
8710 IF N%=0 THEN
8715  RETurn 0
8716 ELSE
8720  IF N%<0 THEN
8725   RETurn -1
8730  ELSE
8740   RETurn 1
8750  END IF
8760 END IF
8799 END DEFine SGN
```

El procedimiento **busque_pozos** trabaja examinando la fila de cuadratínes situada exactamente debajo del túnel de la hormiga y conservando esas coordenadas que son devueltas como valores en X1% y X2%.

```
5600 DEFine PROCedure busque_pozos (Y%)
5610 X%=0
5620 REPeat busque1
5630  COL=tinte(Y%+1,X%,1,1)
5640  X1%=X%
5650  X%=X%+1
5660  IF X%>28 OR COL=0 THEN EXIT busque1
5670 END REPeat busque1
5680 X2%=0
5690 REPeat busque2
5700  COL=tinte(Y%+1,X%,1,1)
5705  IF COL=0 THEN X2%=X%
5710  X%=X%+1
5720  IF X%>28 OR COL=0 THEN EXIT busque2
5730 END REPeat busque2
5799 END DEFine busque_pozos
```

El primer bucle condicionado (líneas 5620 a 5670) busca el primer pozo, y el segundo (líneas 5690 a 5730) examina la posición del segundo pozo. Si no hubiera segundo pozo, este bucle termina cuando se alcanza el borde de la pantalla y entrega el resultado X2% puesto a cero.

El tercer procedimiento nuevo empleado para capturar al hombre es el procedimiento **arrastre,** que es muy similar al procedimiento **coja_hombre.**

```
5800 DEFine PROCedure arrastre (X%)
5810 IF A%(Q%,1)+2=X% THEN XM%=X%:
     END DEFine arrastre
5820 IF SGN(X%-A%(Q%,1)-2)<>A%(Q%,3) THEN
     vire Q%
5830 REPeat paso_jala
5840  porte_grafo A%(Q%,1),A%(Q%,2),9
5845  porte_grafo A%(Q%,1)+1,A%(Q%,2),9
5848  porte_grafo A%(Q%,1)+2,A%(Q%,2),9
5850  A%(Q%,1)=A%(Q%,1)+A%(Q%,3)
5860  porte_grafo A%(Q%,1),A%(Q%,2),A1%(Q%,1)
5863  porte_grafo A%(Q%,1)+1,A%(Q%,2),A1%(Q%,2)
5866  porte_grafo A%(Q%,1)+2,A%(Q%,2),4
5870  demore
5880  IF A%(Q%,1)+2=X% THEN EXIT paso_jala
5890 END REPeat paso_jala
5895 XM%=X%
5899 END DEFine arrastre
```

La única diferencia real entre el procedimiento **arrastre** y el procedimiento **coja_hombre** es que en el primero se mueve la hormiga y el hombre hasta el pozo más cercano, mientras que en el segundo sólo se desplaza la hormiga hasta la posición ocupada por el hombre. Para permitir al jugador el tiempo suficiente como para ver lo que sucede, ambos procedimientos apelan al procedimiento **demore** para retardar las cosas un poco:

```
8600 DEFine PROCedure demore
8610 PAUSE 5
8699 END DEFine demore
```

Mejorando el juego

Después de jugar unas cuantas mangas de Hormiguero se hace rápidamente obvio que es muy fácil de conseguir llegar al nidal de huevos. Un juego que es demasiado fácil no es juego en absoluto, y en este punto muchos programadores pudieran estar tentados de considerar la idea de hormiguero como un fallo. De hecho, la primera versión de la mayoría de los juegos fracasa en funcionar tan bien como se esperaba por una u otra razón, y el siguiente paso en el desarrollo de cualquier programa de juegos es intentar determinar lo que está mal y arreglarlo.

El primer impulso es intentar hacer el juego más difícil incrementando el tiempo permitido. Si intentas eso, verás que disminuyendo el tiempo permitido tiene muy poco efecto hasta que de repente alcanzas un punto donde el juego es casi imposible. El problema es que el juego puede siempre completarse aproximadamente en la misma cantidad de tiempo. Si haces el límite de tiempo mayor que eso, entonces el juego es fácil. Pero si lo haces menor, entonces el juego es imposible. Para hacer Hormiguero más interesante, tenemos primeramente que determinar por qué el juego puede completarse en una cantidad prefijada de tiempo.

La razón más obvia es que inicialmente todas las hormigas se están moviendo hacia la derecha, y en cuanto que el hombre dé un paso hacia los túneles de la izquierda, puede llegar al nidal antes de que cualquiera de las hormigas haya tenido mucho tiempo para virar hacia el otro lado. La solución a eso es hacer que las hormigas se muevan en direcciones aleatorias al comienzo del juego. Dado que ya hemos preparado el procedimiento **vire**, eso se consigue más fácilmente añadiéndole la línea siguiente:

```
1135  IF RND>.5 THEN vire I
```

Después de este cambio el juego es más difícil, y por ende más interesante, pero todavía es posible para el motivo humano moverse a través de cualquiera de los motivos hormigueriles sin ser capturado, y eso significa que no tiene realmente que preocuparse sobre dónde están exactamente las hormigas en los túneles. La razón para que el hombre pueda ser capaz de atravesar una hormiga es que la única vez que se comprueba si ha habido captura es cuando se mueve la hormiga, y por cada hormiga dada, eso sólo sucede por término medio una vez cada cinco movimientos del hombre.

La solución a eso es añadir una línea para comprobar si el hombre se mueve sobre una posición que ya está ocupada por una hormiga. Eso puede hacerse añadiendo:

```
5090 IF tinte(YM%,XM%,4,4)=3 THEN
5095  GOLPE=CERTO
5096  Q%=(YM%-2)/5-1
5098 END IF
```

dentro del procedimiento **mueva_hombre.** Esto simplemente comprueba para estar seguro que la posición que el hombre va a tomar es de color negro.

Una vez que se haya hecho este cambio, se hace aparente otra faceta perturbadora. Dado que una hormiga puede dar una zancada de hasta tres cuadratínes cada vez es posible que una hormiga 'salte' por encima del hombre. Para hacer eso imposible, las hormigas tendrían que tener sus pasos restringidos a un máximo de dos cuadratínes cada vez. Eso puede realizarse de la manera más simple cambiando la línea 3050 del procedimiento **mueva_hormi** para que sea:

```
3050 A%(Q%,1)=A%(Q%,1)+A%(Q%,3)*2
```

Eso hace que todas las hormigas se muevan con una constante que equivale a dos cuadratínes. Después de hacer este cambio, las hormigas todavía aparentan moverse lo suficientemente rápidas y con bastante variabilidad como para parecer interesante.

En este momento se han hecho ya la mayoría de los cambios obvios, y el juego es ciertamente más difícil e interesante, pero todavía le falta el grado de desafío que un buen juego debe ofrecer. Después de haber jugado unas cuantas mangas, parece que la razón de eso es que durante gran parte del tiempo las hormigas están demasiado alejadas de los pozos para suponer ninguna amenaza al hombre. Las mejores ocasiones aparecen cuando las hormigas están, por suerte, todas reunidas hacia la mitad de la pantalla. Eso sugiere que el recorrido de las hormigas debiera reducirse y que las parejas de pozos debieran colocarse más cerca una de otra. Para hacer eso, las líneas 3060, 3061, 3070 y 3071 del procedimiento **mueva_hormi** tienen que cambiarse para que sean:

```
3060 IF A%(Q%,1)<10 THEN
3061  A%(Q%,1)=10
3070 IF A%(Q%,1)>20 THEN
3071  A%(Q%,1)=20
```

y la línea 8410 en el procedimiento **pon_pozos** se cambia a:

```
8410 X1%=RND(1 TO 4)+9:X2%=RND(0 TO 4)+16
```

Con todos estos cambios Hormiguero es un juego que combina la velocidad con la estrategia y se disfruta bastante jugándolo.

Sugerencias para trabajo adicional

La parte principal del programa Hormiguero es bastante completa y la mayoría de las perspectivas para trabajo adicional se centran sobre las rutinas de finalización del juego. El procedimiento **quite_nidal** y el procedimiento **rebrote_hormis** son bastante débiles cuando se comparan con las fascinantes representaciones que las hormigas llevan a cabo cuando se efectúa la captura del hombre. El juego por sí mismos puede mejorarse haciendo a las hormigas un poco más 'inteligentes' haciéndolas que detecten y se muevan hacia el hombre cuando desciende a su túnel.

La versión final - un listado completo

El listado siguiente incluye todas las modificaciones presentadas en el texto.

```
 10 REMark HORMIGUERO
 20 MODE 8
 25 WINDOW 512,256,0,0
 26 prologue
 30 prepa_juego
 35 REPeat manga
 36  prepa_manga
 50  trace_escena
 60  REPeat movida
 70   mueva_ambos
 80   IF TIEMPO OR GOLPE OR NIDAL THEN EXIT movida
 90  END REPeat movida
100  fine_manga
110  IF OTRA=FALSO THEN EXIT manga
120 END REPeat manga
130 INK 0
140 PAPER 7
150 CLS
199 STOP

350 DEFine PROCedure prepa_juego
360 DATA -5245,-10109,-7806,-11646
370 DATA -6783,1665,2,0,8769,8316
380 DATA 0,0,9264,18432,8834,-11524,128
390 DATA 22664,-20228,32,26350,28672,20085
400 dire_rus=RESPR(100)
410 RESTORE 360
420 FOR I=dire_rus TO dire_rus+22*2 STEP 2
430 READ B
440 POKE_W I,B
450 END FOR I
520 C_TAB=RESPR(32*12)
530 prepa_grafo 0,3,0,0,14,15,63,31,51,68,132
```

```
540 prepa_grafo 1,3,0,3,20,216,252,168,192,64,16
550 prepa_grafo 2,3,0,0,112,248,252,248,204,34,33
560 prepa_grafo 3,3,0,192,40,27,63,25,3,4,8
570 prepa_grafo 4,6,0,24,24,126,24,60,102,195,195
580 prepa_grafo 5,4,0,1,1,3,3,7,7,15,15
590 prepa_grafo 6,4,0,31,31,63,63,127,127,255,255
600 prepa_grafo 7,4,0,128,128,192,192,224,224,240,240
610 prepa_grafo 8,4,0,248,248,252,252,254,254,255,255
620 prepa_grafo 9,0,0,255,255,255,255,255,255,255,255
630 prepa_grafo 10,1,1,255,255,255,255,255,255,255,255
640 prepa_grafo 11,6,1,24,24,126,24,60,102,195,195
650 FALSO=0
660 CERTO=1
670 OTRA=CERTO
699 END DEFine prepa_juego

1000 DEFine PROCedure prepa_manga
1050 RANDOMISE
1060 XM%=RND(0 TO 20)+10
1070 YM%=2
1080 DIM A%(4,3),A1%(4,2)
1090 FOR I=0 TO 4
1100  A%(I,1)=RND(0 TO 10)+10
1110  A%(I,2)=(I+1)*5+2
1120  A%(I,3)=1
1130  A1%(I,1)=0:A1%(I,2)=1
1135  IF RND>.5 THEN vire I
1140 END FOR I
1160 OTRA=FALSO
1170 GOLPE=FALSO
1180 NIDAL=FALSO
1190 TIEMPO=FALSO
1299 END DEFine prepa_manga

1300 DEFine PROCedure prepa_grafo(S%,F%,B%,
     R0%,R1%,R2%,R3%,R4%,R5%,R6%,R7%)
1310 LOCal A,FH%,FL%,BH%,BL%
1320 A=C_TAB+32*S%
1330 FH%=(F% DIV 4)*2
1340 FL%=F% && 3
1350 BH%=(B% DIV 4)*2
1360 BL%=B% && 3
1370 prepa_raya(R0%)
1380 prepa_raya(R1%)
1390 prepa_raya(R2%)
1400 prepa_raya(R3%)
1410 prepa_raya(R4%)
1420 prepa_raya(R5%)
1430 prepa_raya(R6%)
1440 prepa_raya(R7%)
1499 END DEFine prepa_grafo

1500 DEFine PROCedure prepa_raya(R%)
```

```
1502 LOCal M%,J,I,DL%,DH%
1503 M%=128
1504 FOR J=1 TO 2
1505  DL%=0:DH%=0
1510  FOR I=1 TO 4
1520   IF (R% && M%)=M% THEN
1530    DL%=FH%+DL%*4
1540    DH%=FL%+DH%*4
1550   ELSE
1560    DL%=BH%+DL%*4
1570    DH%=BL%+DH%*4
1580   END IF
1590   M%=M% DIV 2
1600  END FOR I
1610  POKE A,DL%
1620  POKE A+1,DH%
1630  A=A+2
1640 END FOR J
1699 END DEFine prepa_raya

1700 DEFine PROCedure porte_grafo(X%,Y%,S%)
1710 CALL dire_rus,X%,Y%,S%,C_TAB
1799 END DEFine porte_grafo

2000 DEFine PROCedure trace_escena
2010 PAPER 2
2020 CLS
2040 FOR X=0 TO 31
2050  porte_grafo X,7,9
2060  porte_grafo X,12,9
2070  porte_grafo X,17,9
2080  porte_grafo X,22,9
2090  porte_grafo X,27,9
2100 END FOR X
2120 FOR Y=0 TO 2
2125  FOR X=0 TO 31
2130  porte_grafo X,Y,10
2135  END FOR X
2140 END FOR Y
2160 pon_pozos 2,3,6
2170 pon_pozos 1,8,11
2180 pon_pozos 2,13,16
2190 pon_pozos 1,18,21
2200 pon_pozos 2,22,26
2220 porte_grafo XM%,YM%,11
2240 porte_grafo 1,26,5:porte_grafo 2,26,7
2250 porte_grafo 1,27,6:porte_grafo 2,27,8
2270 FOR I=0 TO 4
2280  porte_grafo A%(I,1),A%(I,2),A1%(I,1)
2285  porte_grafo A%(I,1)+1,A%(I,2),A1%(I,2)
2290 END FOR I
2300 SDATE 2000,1,1,0,0,0
2999 END DEFine trace_escena
```

```
3000 DEFine PROCedure avance
3020 Q%=RND(1 TO 5)-1
3030 porte_grafo A%(Q%,1),A%(Q%,2),9:
     porte_grafo A%(Q%,1)+1,A%(Q%,2),9
3040 IF RND<5E-2 THEN vire Q%
3050 A%(Q%,1)=A%(Q%,1)+A%(Q%,3)*2
3060 IF A%(Q%,1)<10 THEN
3061  A%(Q%,1)=10
3062  vire Q%
3065 END IF
3070 IF A%(Q%,1)>20 THEN
3071  A%(Q%,1)=20
3072  vire Q%
3075 END IF
3080 porte_grafo A%(Q%,1),A%(Q%,2),A1%(Q%,1)
3085 porte_grafo A%(Q%,1)+1,A%(Q%,2),A1%(Q%,2)
3090 GOLPE=tocando(Q%)
3099 END DEFine avance

3500 DEFine PROCedure vire (Q%)
3510 IF A1%(Q%,1)=3 THEN
3520  A1%(Q%,1)=0
3525  A1%(Q%,2)=1
3530 ELSE
3540  A1%(Q%,1)=3
3545  A1%(Q%,2)=2
3550 END IF
3560 A%(Q%,3)=-A%(Q%,3)
3599 END DEFine vire

4000 DEFine PROCedure mueva_ambos
4010 mueva_hombre
4020 IF GOLPE=FALSO THEN
4030  avance
4040  mire_hora
4050 END IF
4099 END DEFine mueva_ambos

4500 DEFine PROCedure mire_hora
4510 TIME$=DATE$
4520 SEC=TIME$(17)*60
4530 SEC=SEC+TIME$(19 TO 20)
4550 IF SEC>MAX THEN TIEMPO=CERTO
4999 END DEFine mire_hora

5000 DEFine PROCedure mueva_hombre
5001 LOCal I%
5003 I%=KEYROW(1)
5004 IF I%=0 THEN END DEFine mueva_hombre
5010 IF YM%>4 THEN
5015  porte_grafo XM%,YM%,9
5020 ELSE
5022  porte_grafo XM%,YM%,10
```

```
5025 END IF
5060 IF I%=2 AND XM%>1 THEN XM%=XM%-1
5070 IF I%=16 AND XM%<30 THEN XM%=XM%+1
5080 IF I%=128 AND tinte(YM%+1,XM%,1,1)=0 THEN
     baje_pozo
5090 IF tinte(YM%,XM%,4,4)=3 THEN
5095  GOLPE=CERTO
5096  Q%=(YM%-2)/5-1
5098 END IF
5100 IF YM%>4 THEN
5102  porte_grafo XM%,YM%,4
5104 ELSE
5106  porte_grafo XM%,YM%,11
5107 END IF
5110 IF XM%=3 AND YM%=27 THEN NIDAL=CERTO
5199 END DEFine mueva_hombre

5500 DEFine PROCedure baje_pozo
5520 FOR I=1 TO 4
5530  YM%=YM%+1
5540  porte_grafo XM%,YM%,4
5550  BEEP .1,YM%
5560  porte_grafo XM%,YM%,9
5570 END FOR I
5575 YM%=YM%+1
5580 BEEP
5599 END DEFine baje_pozo

5600 DEFine PROCedure busque_pozos (Y%)
5610 X%=0
5620 REPeat busque1
5630  COL=tinte(Y%+1,X%,1,1)
5640  X1%=X%
5650  X%=X%+1
5660  IF X%>28 OR COL=0 THEN EXIT busque1
5670 END REPeat busque1
5680 X2%=0
5690 REPeat busque2
5700  COL=tinte(Y%+1,X%,1,1)
5705  IF COL=0 THEN X2%=X%
5710  X%=X%+1
5720  IF X%>28 OR COL=0 THEN EXIT busque2
5730 END REPeat busque2
5799 END DEFine busque_pozos

5800 DEFine PROCedure arrastre (X%)
5810 IF A%(Q%,1)+2=X% THEN XM%=X%:
     END DEFine arrastre
5820 IF SGN(X%-A%(Q%,1)-2)<>A%(Q%,3) THEN
     vire Q%
5830 REPeat paso_jala
5840  porte_grafo A%(Q%,1),A%(Q%,2),9
5845  porte_grafo A%(Q%,1)+1,A%(Q%,2),9
5848  porte_grafo A%(Q%,1)+2,A%(Q%,2),9
```

```
5850  A%(Q%,1)=A%(Q%,1)+A%(Q%,3)
5860  porte_grafo A%(Q%,1),A%(Q%,2),A1%(Q%,1)
5863  porte_grafo A%(Q%,1)+1,A%(Q%,2),A1%(Q%,2)
5866  porte_grafo A%(Q%,1)+2,A%(Q%,2),4
5870  demore
5880  IF A%(Q%,1)+2=X% THEN EXIT paso_jala
5890 END REPeat paso_jala
5895 XM%=X%
5899 END DEFine arrastre

5900 DEFine PROCedure coja_hombre (Q%)
5910 IF A%(Q%,1)+1=XM% THEN END DEFine coja_hombre
5920 IF SGN(XM%-A%(Q%,1)-1)<>A%(Q%,3) THEN
     vire Q%
5930 REPeat avance
5940  porte_grafo A%(Q%,1),A%(Q%,2),9
5945  porte_grafo A%(Q%,1)+1,A%(Q%,2),9
5950  A%(Q%,1)=A%(Q%,1)+A%(Q%,3)
5960  porte_grafo A%(Q%,1),A%(Q%,2),A1%(Q%,1)
5970  porte_grafo A%(Q%,1)+1,A%(Q%,2),A1%(Q%,2)
5975  demore
5980  IF A%(Q%,1)+1=XM% THEN EXIT avance
5990 END REPeat avance
5999 END DEFine coja_hombre

6000 DEFine PROCedure fine_manga
6010 IF GOLPE THEN
6012  capture
6014 ELSE
6020  IF TIEMPO THEN
6022   trampa
6024  ELSE
6030   IF NIDAL THEN
6032    destru_nido
6040   END IF
6050  END IF
6060 END IF
6100 REPeat otra_manga
6110  AT 24,3
6112  PRINT 'OTRO JUEGO S/N ?';
6114  INPUT N$
6120  IF N$(1)='S' OR N$(1)='N' THEN
     EXIT otra_manga
6125 END REPeat otra_manga
6130 IF N$(1)='S' THEN OTRA=CERTO
6199 END DEFine fine_manga

6200 DEFine PROCedure capture
6210 AT 23,3: PRINT 'Te han capturado'
6220 REPeat hormi_jala
6230  IF Q%=4 THEN EXIT hormi_jala
6240  coja_hombre(Q%)
6250  busque_pozos A%(Q%,2)
```

```
6260  IF ABS(X2%-A%(Q%,1))<ABS(X1%-A%(Q%,1))
      AND X2%<>0 THEN
6262   arrastre X2%
6264  ELSE
6266   arrastre X1%
6268  END IF
6270  porte_grafo A%(Q%,1)+2,A%(Q%,2),9
6275  porte_grafo A%(Q%,1)+3,A%(Q%,2),9
6280  BEEP .1,-10
6290  baje_pozo
6300  porte_grafo XM%,YM%,4
6310  Q%=Q%+1
6320 END REPeat hormi_jala
6330 coja_hombre(Q%)
6340 arrastre 5
6399 END DEFine capture

6500 DEFine PROCedure trampa
6510 AT 23,3:PRINT 'LAS HORMIGAS ESTAN RESUCITANDO'
6520 FOR I=1 TO RND(0 TO 20)+20
6530  X=RND(0 TO 25)+5
6532  Y=RND(0 TO 15)+5
6535  T%=RND(0 TO 4)
6540  porte_grafo X,Y,A1%(T%,1)
6545  porte_grafo X+1,Y,A1%(T%,2)
6550  BEEP .1,I
6555  PAUSE 1
6560 END FOR I
6570 BEEP
6599 END DEFine trampa

6800 DEFine PROCedure destru_nido
6810 AT 23,2: PRINT 'LO LOGRASTE'
6815 RESTORE 6900
6820 REPeat melodia
6830  READ D,P
6840  IF D=999 THEN EXIT melodia
6850  BEEP D,P
6860  PAUSE D*125
6865  BEEP
6870 END REPeat melodia
6900 DATA .1,14,.1,13
6910 DATA .1,12,.1,11
6920 DATA .2,10,.2,10
6990 DATA 999,999
6999 END DEFine destru_nido

8000 DEFine PROCedure prologue
8010 BORDER 0:INK 6:PAPER 0:CLS
8020 AT 3,14:PRINT 'H O R M I G U E R O'
8040 AT 7,3:PRINT 'En este juego tienes una
     cierta'
8050 PRINT TO 2;'cantidad de tiempo para destruir
     el nido de las hormigas'
```

```
8060 PRINT TO 2;'antes de que las hormigas resuciten!'\
8070 PRINT TO 2;'ATENCION: las hormigas soldado que
     guardan los'
8080 PRINT TO 2;'tuneles te capturaran y
     te llevaran'
8090 PRINT TO 2;'al nido como alimento de sus crias!'\
8100 PRINT TO 2;'Buena suerte!'
8200 REPeat dif
8300 AT 18,5:PRINT 'Dime nivel de dificultad -'
8310 PRINT TO 3;'1. Experto, 2. Medio,
     3. novato ';
8320 INPUT DF
8330 IF DF>0 AND DF<4 THEN EXIT dif
8335 END REPeat dif
8340 MAX=(20+DF*10)
8350 AT 22,15
8360 FLASH 1
8370 PRINT 'Por favor espera'
8380 FLASH 0
8399 END DEFine prologue

8400 DEFine PROCedure pon_pozos (NO,Y1,Y2)
8410 X1%=RND(1 TO 4)+9:X2%=RND(0 TO 4)+16
8420 FOR Y=Y1 TO Y2
8430  IF NO=1 THEN porte_grafo INT((X1%+X2%)/2),Y,9
8440  IF NO=2 THEN porte_grafo X1%,Y,9:porte_grafo X2%,Y,9
8450 END FOR Y
8499 END DEFine pon_pozos

8500 DEFine FuNction tocando (Q%)
8510 IF A%(Q%,2)<>YM% THEN RETurn FALSO
8520 IF A%(Q%,1)<>XM% AND A%(Q%,1)+1<> XM%
     THEN RETurn FALSO
8530 RETurn CERTO
8599 END DEFine tocando

8600 DEFine PROCedure demore
8610 PAUSE 5
8699 END DEFine demore

8700 DEFine FuNction SGN(N%)
8710 IF N%=0 THEN
8715  RETurn 0
8716 ELSE
8720  IF N%<0 THEN
8725   RETurn -1
8730  ELSE
8740   RETurn 1
8750  END IF
8760 END IF
8799 END DEFine SGN

9500 DEFine FuNction tinte(R%,C%,X%,Y%)
9510 LOCal A%,P%,B%,A
```

```
9520 P%=C%*8+X%
9530 A=131072+(512*R%+Y%*64+P% DIV 4)*2
9540 B%=PEEK(A+1)
9550 A%=PEEK(A)
9560 P%=P% MOD 4
9570 IF P%=3 THEN RETurn ((2 && A%)*2)+(3 && B%)
9580 IF P%=2 THEN RETurn ((8 && A%) DIV 2)+
     ((12 && B%) DIV 4)
9590 IF P%=1 THEN RETurn ((32 && A%) DIV 8)+
     ((48 && B%) DIV 16)
9600 IF P%=0 THEN RETurn ((128 && A%) DIV 32)+
     ((192 && B%) DIV 64)
9699 END DEFine tinte
```

Capítulo Tres

Salta Rana

Salta Rana es un juego que involucra animación completa en dos dimensiones. Es decir, uno de los motivos gráficos del juego, la rana, se mueve tanto horizontalmente como verticalmente (i.e. diagonalmente) al mismo tiempo. (Eso debiera compararse con Hormiguero del último capítulo, donde los motivos solamente se movían sólo o bien horizontalmente o bien verticalmente). Además, se usan algunas técnicas distintas e interesantes de animación para crear una imagen fascinante usando sorprendentemente pocas instrucciones de BASIC. El juego por sí mismo es divertido tanto para ver como para jugar y proporciona un surtido variado de posibilidades.

El diseño del juego

Salta Rana usa mucho de los elementos que se encuentran en uno de los más populares y primeros juegos para ordenador, denominado variadamente como "Tira Muro", "Batidero", "Rompebol" y muchos otros nombres. Estos juegos consisten en hacer que una bola o pelota rebote en los bordes de la pantalla por medio de un bate o pala, con el objetivo de eliminar tantos bloques coloreados expuestos en una banda -el muro de ladrillos- en la parte alta de la pantalla. En las versiones más sencillas del juego la habilidad consiste simplemente en mover el bate para interceptar la pelota y así mantenerla rebotando. En las versiones más avanzadas, el jugador puede dirigir el rebote de la bola golpeándola con diferentes partes del bate, consiguiendo "efecto" en el golpe.

En Salta Rana los bloques coloreados están sustituidos por hileras de insectos con forma de mosca y la bola por una rana saltarina. Obviamente, sería cruel usar un bate normal para mantener en movimiento a la rana, así que en lugar de eso hemos usado un columpio. Una segunda rana se sienta en el columpio y cuando la primera rana baja al columpio, la segunda es catapultada al aire mientras que la primera continúa sentada en el columpio esperando su turno.

Otra diferencia entre la forma tradicional del juego y Salta Rana se encuentra en el camino que la rana traza en su vuelo a través del aire. En los anteriores, la bola se mueve en línea recta rebotando de esa manera por toda la pantalla, pero en Salta Rana la rana se mueve como si fuera atraída hacia el terreno por la fuerza de la gravedad. Cada vez que la rana salta al columpio, la otra es catapultada un poquito más alto de manera que para alcanzar las filas traseras de insectos es necesario mantener a las ranas saltando.

Eso es todo lo que corresponde al fundamento del juego, pero antes de pasar a implementarlo en BASIC merece la pena examinar los principios de hacer que un motivo gráfico rebote por los bordes de la pantalla y se mueva bajo la acción simulada de la gravedad. La Fig. 3.1 da un calco de la imagen en pantalla cuando el programa funciona.

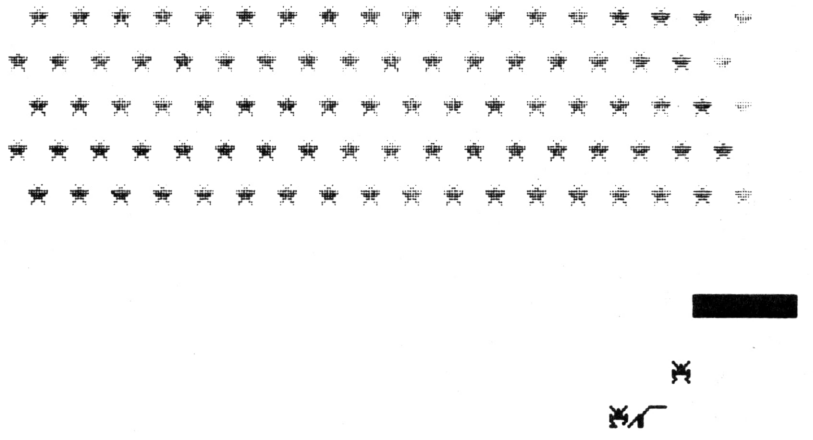


Fig. 3.1.

Rebotando y cayendo

La idea de animar un objeto mediante la actualización repetida de sus coordenadas al añadir cantidades que corresponden a sus velocidades horizontales y verticales ya se ha descrito en el Capítulo Dos. Es decir, el ciclo de animación fundamental es:

```
porte_grafo X,Y,B          :REM BORRA OBJETO
X=X+VX:Y=Y+VY              :REM ACTUALIZA COORDENADAS
porte_grafo X,Y,S          :REM MUESTRA OBJETO
```

donde X e Y representan la posición del objeto y VX y VY representan el desplazamiento horizontal y vertical del objeto. Usando este esquema, parece que el motivo se mueve por la pantalla siguiendo la misma línea de movimiento hasta que desaparece por el borde. Esta clase de movimiento no es realmente muy útil en un juego y no pasa mucho antes de que surja la necesidad de que el motivo gráfico rebote.

En principio, el rebote de un objeto sobre una frontera vertical u horizontal es fácil. Si un motivo gráfico moviéndose con la velocidad VX,VY golpea una línea horizontal, entonces su velocidad cambia a VX,-VY. Es decir, un rebote sobre una frontera horizontal invierte la velocidad vertical.

Igualmente, un rebote sobre una frontera vertical invierte la velocidad horizontal. Como ejemplo de este principio ensaya el siguiente programa:

```
10 prepa_juego
15 PAPER 0:CLS
20 VX=1:VY=1
30 X=RND(0 TO 31):Y=RND(0 TO 31)
40 porte_grafo X,Y,0
50 X=X+VX:Y=Y+VY
60 IF X<1 OR X>30 THEN VX=-VX
70 IF Y<1 OR Y>30 THEN VY=-VY
80 porte_grafo X,Y,1
90 GO TO 40

350 DEFine PROCedure prepa_juego
360 DATA -5245,-10109,-7806,-11646
370 DATA -6783,1665,2,0,8769,8316
380 DATA 0,0,9264,18432,8834,-11524,128
390 DATA 22664,-20228,32,26350,28672,20085
400 dire_rus=RESPR(100)
410 RESTORE 360
420 FOR I=dire_rus TO dire_rus+22*2  STEP 2
430  READ B
440  POKE_W I,B
450 END FOR I
510 C_TAB=RESPR(32*2)
520 prepa_grafo 0,0,0,255,255,255,255,255,255,255,255
530 prepa_grafo 1,7,7,255,255,255,255,255,255,255,255
599 END DEFine prepa_juego
```

(Recuerda incluir el procedimiento **porte_grafo**). La línea 60 detecta las colisiones con los bordes verticales de la pantalla e invierte la velocidad horizontal. La línea 70 detecta las colisiones con el borde horizontal de la pantalla e invierte la velocidad vertical. El resultado global es un bloque sólido que ocupa un cuadratín y rebota por toda la pantalla. Lo que es sorprendente es que esta suerte de animación emplee tan pocas líneas de BASIC.

Casi todas las otras clases de movimientos complicados de los motivos gráficos pueden producirse cambiando las velocidades horizontal y vertical en cada ronda del bucle de animación. Por ejemplo, para hacer que un motivo gráfico se mueva por la pantalla como si estuviera bajo la influencia de la gravedad, todo lo que tenemos que hacer es cambiar la velocidad vertical en cada ronda del bucle por una cantidad prefijada. Eso remedia lo que sucede cuando por ejemplo una bola real se lanza al aire. La bola comienza con una cierta velocidad vertical que se reduce constantemente por la aceleración de la gravedad hasta que alcanza cero, y luego cambia de dirección haciendo que la bola regrese a la tierra aumentando paulatinamente su velocidad. Observa que la gravedad no afecta a la velocidad horizontal de la bola en absoluto. Los únicos factores que cambian la velocidad horizontal son la resistencia del aire o el viento, y en la mayoría de los casos esos pueden ignorarse. El siguiente programa hace que un motivo se mueva como una pelota lanzada.

```
 10 prepa_juego
 15 PAPER 0:CLS
 20 X=0:Y=31
 30 VX=.5:VY=-.8
 40 A=.025
 50 porte_grafo X,Y,0
 60 X=X+VX:Y=Y+VY
 70 VY=VY+A
 80 IF Y>30 THEN STOP
 90 porte_grafo X,Y,1
100 GO TO 50
```

(Recuerda incluir el procedimiento **prepa_juego**, tal como se usó en el programa previo, y **porte_grafo**). La línea 60 efectúa la actualización habitual de coordenadas, pero en este programa la línea 70 también actualiza la velocidad vertical añadiéndole una constante. El resultado es un motivo gráfico que describe una trayectoria parabólica. El tamaño de la constante usada en la línea 40 controla la fuerza de la gravedad que actúa sobre el motivo. Incrementándola haremos que baje más rápidamente. Observa que para permitir actuar a la gravedad es necesario usar coordenadas que no sean enteros, y eso puede causar complicaciones.

El programa principal

Después de este comentario sobre los temas teóricos involucrados, es hora de regresar a los detalles de la programación. Los elementos del juego son directos. Al jugador se le dan diez ranas para intentar comer todos los insectos expuestos en la parte alta de la pantalla. Cada nueva rana comienza despegándose de la plataforma y el usuario tiene que situar el columpio para lanzar la segunda rana al aire. Una vez que se ha comenzado de esta manera, el bucle principal de animación continúa hasta que el usuario falla y pierde la rana con el columpio. Aparece entonces una nueva rana y el juego continúa.

Hay dos maneras en que el juego puede terminar. Bien sea porque el jugador se las arregla para comerse todos los insectos, o bien sea por haber usado las diez ranas disponibles. El programa principal es:

```
10 REMark Ranas Saltarinas
20 WINDOW 512,256,0,0
25 prologue
30 prepa_juego
35 REPeat manga
36  prepa_manga
50  expo_moscas
60  saque_rana
70  REPeat movida
80   mueva_polin
```

```
 90   mueva_rana
100   IF ACABADO THEN EXIT movida
110  END REPeat movida
120  fine_juego
130  IF NOT OTRA THEN EXIT manga
140 END REPeat manga
199 STOP
```

El procedimiento **prologue** muestra una plana de instrucciones y pide al usuario que elija el grado de dificultad. El procedimiento **prepa_juego** implanta el código máquina necesario para **porte_grafo** y los motivos u objetos definidos por el usuario. El procedimiento **prepa_manga** establece las condiciones iniciales de las variables usadas posteriormente en el juego. El procedimiento **expo_moscas** expone cinco filas de insectos en la parte superior de la pantalla preparando el juego para comenzar. El procedimiento **saque_rana** hace que una rana salte de la plataforma mientras el jugador intenta colocar el trampolín con la otra en el lugar donde vaya a caer.

El bucle principal de animación es el formado por las líneas 70 a 110. El procedimiento **mueva_polín** permite que el jugador mueva el columpio o trampolín usando las teclas de flecha izquierda y derecha, y el procedimiento **mueva_rana** efectúa el movimiento de la rana teniendo en cuenta la gravedad y cualquier rebote que sea necesario. Finalmente, el procedimiento **fine_juego** resume la puntuación del jugador y pregunta si desea jugar otra vez.

Procedimientos prepa_juego, prepa_manga y prologue

El procedimiento **prepa_juego** no es muy diferente del dado en el Capítulo Dos, sólo que ahora se definen los cuadratínes característicos que forman los motivos gráficos empleados en este juego.

```
350 DEFine PROCedure prepa_juego
360 DATA -5245,-10109,-7806,-11646
370 DATA -6783,1665,2,0,8769,8316
380 DATA 0,0,9264,18432,8834,-11524,128
390 DATA 22664,-20228,32,26850,28672,20085
400 dire_rus=RESPR(100)
410 RESTORE 360
420 FOR I=dire_rus TO dire_rus+22*2  STEP 2
430  READ B
440  POKE_W I,B
450 END FOR I
510 C_TAB=RESPR(32*9)
520 prepa_grafo 0,2,6,130,84,56,186,254,130,68,255
530 prepa_grafo 1,2,6,1,2,4,24,56,88,152,24
540 prepa_grafo 2,2,6,252,0,0,0,0,0,0,0
550 prepa_grafo 3,1,6,36,24,255,255,126,60,102,66
560 prepa_grafo 4,4,6,130,84,56,186,254,130,68,130
570 prepa_grafo 5,6,6,255,255,255,255,255,255,255,255
575 prepa_grafo 6,2,2,255,255,255,255,255,255,255,255
576 prepa_grafo 7,2,6,128,64,32,24,28,26,25,24
```

```
577 prepa_grafo 8,2,6,63,0,0,0,0,0,0,0,0
580 FALSO=0
590 CERTO=1
599 END DEFine prepa_juego
```

Las líneas 520 a 570 describen los motivos gráficos usados, que se ilustran en la Fig. 3.2 y se resumen en la Tabla 3.1.

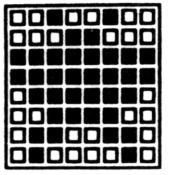
Tabla 3.1. Cuadratínes característicos para motivos gráficos

Número	Forma	Frente	Fondo
0	rana y trampolín a izquierda	rojo	amarillo
1	pivote balancín a derecha	rojo	amarillo
2	trampolín lado arriba	rojo	amarillo
3	mosca	azul	amarillo
4	rana	verde	amarillo
5	espacio en blanco	amarillo	amarillo
6	bloque	rojo	rojo
7	pivote balancín a izquierda	rojo	amarillo
8	trampolín lado abajo	rojo	amarillo

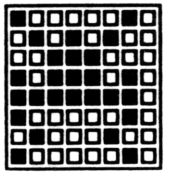
El procedimiento **prepa_manga** es muy simple:

```
1000 DEFine PROCedure prepa_manga
1010 M=1
1030 X%=23
1040 RANA=1
1050 ACABADO=FALSO
1060 OTRA=FALSO
1070 SC=0
1099 END DEFine prepa_manga
```

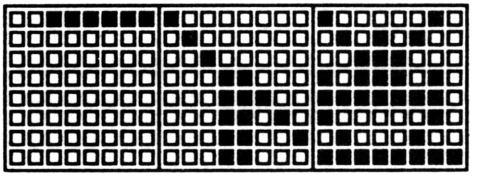
M se usa como un indicador de qué parte del trampolín, izquierda o derecha, está actualmente en uso. Si M es 1, entonces el trampolín inclinado hacia la izquierda está en pantalla, y si M es cero es el inclinado a la derecha. X% da la posición horizontal del trampolín y está fijado a 23 por la línea 1030 para dar la posición inicial cuando comienza el juego. ACABADO se usa para indicar el final del juego y SC se usa para obtener el 'score' o puntuación corriente.



(a)

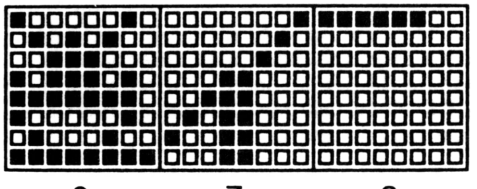


(b)



Trampolin a Derecha

(c)



Trampolin a Izquierda

Fig. 3.2. Motivos gráficos para (a) Mosca y rana. (b) Trampolín a derecha. (c) Trampolín a izquierda.

El procedimiento **prologue** simplemente muestra las instrucciones e impone el grado de dificultad en la variable D:

```
6000 DEFine PROCedure prologue
6010 PAPER 6
6015 INK 1
6020 CLS
6030 AT 5,10
6040 PRINT 'R A N A S  S A L T A R I N A S'
6050 AT 10,4
6060 PRINT 'En este juego debes comerte las moscas'\\
6070 PRINT TO 3;'haciendo que tus dos ranas salten'\\
6080 PRINT TO 3;'del trampolin usando las flechas
     izquierda y derecha'\\
6090 PRINT TO 3;'Dispones de diez ranas para dejar
     limpia'\\
6100 PRINT TO 3;'la pantalla .....'
6110 REPeat dif
6120  AT 21,5
6130  PRINT 'Dime nivel de dificultad '
6140  PRINT TO 8;'1 (dificil) hasta 10 (facil)';
6150  INPUT D
6160  IF D>0 AND D<11 THEN EXIT dif
6170 END REPeat dif
6199 END DEFine prologue
```

Procedimiento expo_moscas

Este procedimiento simplemente saca en pantalla las cinco filas de insectos de la parte superior:

```
2000 DEFine PROCedure expo_moscas
2010 CLS
2020 C%=0
2030 FOR V=2 TO 10 STEP 2
2040  C%=NOT C%
2050  FOR U=2 TO 30 STEP 2
2060   porte_grafo U-C%,V,3
2070  END FOR U
2080 END FOR V
2099 END DEFine expo_moscas
```

La única técnica interesante es el uso de C% en las líneas 2040 a 2080 para determinar el espaciamiento de las filas alternativas. Para comprender cómo funciona, todo lo que necesitas saber es que si C% es 0 entonces NOT C% es obviamente 1, y viceversa.

Procedimiento saque_rana

Después de lo anterior, el procedimiento **saque_rana** expone una rana en la tarima situada en la parte derecha de la pantalla y la mueve ininterrumpidamente hasta el borde, y cuando la alcanza salta hacia abajo. Si el balancín está situado bajo la rana que cae, entonces el juego continúa. Si no es así, se apela al procedimiento de "saque" otra vez, desde dentro del bucle de animación.

```
5000 DEFine PROCedure saque_rana
5010 mueva_polin
5020 FOR I=26 TO 31
5030  porte_grafo I,15,6
5040 END FOR I
5050 FOR F=31 TO 24 STEP -1
5060  porte_grafo F+1,14,5
5070  porte_grafo F,14,4
5080  mueva_polin
5090 END FOR F
5100 FOR G=14 TO 19
5110  porte_grafo 24,G-1,5
5120  porte_grafo 24,G,4
5130  mueva_polin
5140 END FOR G
5145 porte_grafo 24,19,5
5150 F%=24
5155 F=F%
5160 G%=19
5165 G=G%
5170 A=.5+RND/10
5180 B=-(.2+RND/2.5)
5190 P=9E-3
5200 AT 21,2
5210 PRINT 'RANA ';RANA;' PUNTOS';SC
5220 A=A+A/D
5230 B=B+B/D
5240 botando
5299 END DEFine saque_rana
```

La línea 5010 cita **mueva_polín**, que en este caso simplemente sirve para dibujar el trampolín en su posición inicial. Luego el primero de los bucles preconfinados FOR, líneas 5020 a 5040, traza una línea negra corta en la parte derecha de la pantalla que actúa como plataforma de salida de cada una de las ranas. El segundo bucle preconfinado, líneas 5050 a 5090, desplaza la rana a lo largo de esa tarima. En cada ronda de este bucle se recurre al procedimiento **mueva_polín** en la línea 5080, para dar al jugador la oportunidad de desplazar su balancín. El tercer bucle, líneas 5100 a 5140, mueve verticalmente y hacia abajo a la rana. Finalmente, las líneas 5150 a 5230 establecen los valores iniciales de las variables usadas en el resto del programa. F y G se usan para registrar la posición corriente de la rana. Es decir, la rana es llevada como gráfico a pantalla en F,G,4. Ls variables A y B son la velocidad vertical y horizontal de la rana. Ambas se fijan a valores aleatorios -por las líneas 5170 y 5180 respectivamente.

La línea 5190 fija P, la 'constante de la gravedad', al valor 0.009 (P=9E-3). Las líneas 5220 y 5230 ajustan las velocidades y la constante de gravedad para tener en cuenta el grado de dificultad escogido por el jugador. En esencia, haciendo 'D' mayor, se incrementa la velocidad vertical y horizontal, y hace que la rana se mueva aceleradamente. Finalmente, se apela al procedimiento **botando** para comprobar si la rana que sale ha logrado o ha fallado el salto sobre el trampolín.

Procedimientos mueva_polín y mueva_rana

El primero de estos procedimientos permite al jugador desplazar el trampolín usando las teclas de flechas:

```
3000 DEFine PROCedure mueva_polin
3005 LOCal A
3010 A=KEYROW(1)
3020 IF A=16 AND X%<28 THEN X%=X%+1
3030 IF A=2 AND X%>0 THEN X%=X%-1
3040 expo_polin
3099 END DEFine mueva_polin
```

Las líneas 3020 y 3030 inspeccionan las teclas de flecha derecha e izquierda y actualizan acordemente la posición horizontal del trampolín (valor de la variable X%). La línea 3040 expone el trampolín en su nueva posición, apelando a un procedimiento sencillo de hacer y denominado **expo_polín:**

```
3100 DEFine PROCedure expo_polin
3110 SELect ON M
3120  ON M=1
3130   porte_grafo X%,21,5
3140   porte_grafo X%+1,21,0
3150   porte_grafo X%+2,21,1
3180   porte_grafo X%+3,21,2
3220   porte_grafo X%+4,21,5
3240  ON M=0
3250   porte_grafo X%,21,5
3260   porte_grafo X%+1,21,8
3270   porte_grafo X%+2,21,7
3280   porte_grafo X%+3,21,0
3290   porte_grafo X%+4,21,5
3360 END SELect
3999 END DEFine expo_polin
```

Se expone en pantalla un trampolín a derecha o a izquierda dependiendo del valor de M. Observa que no hay necesidad de exponer un blanco para borrar todo el trampolín cuando se cambia del izquierdo al derecho y viceversa. Dado que sólo se mueve una posición a la izquierda o a la derecha cada vez que se da una ronda del bucle de animación, es suficiente dejar en blanco ambos extremos.

El procedimiento **mueva_rana** es bastante directo y sigue la teoría de los objetos que rebotan y caen que hemos dado al comienzo de este capítulo:

```
4000 DEFine PROCedure mueva_rana
4010 porte_grafo F%,G%,5
4020 F=F+B
4030 G=G+A
4040 A=A+P
4050 IF F>30 OR F<2 THEN B=-B
4060 IF G<2 THEN G=2:A=-A
4070 G%=G
4080 F%=F
4090 IF G%<11 THEN cazando
4100 porte_grafo F%,G%,4
4110 IF G%>20 THEN botando
4199 END DEFine mueva_rana
```

La línea 4010 borra la rana en la vieja posición. Las líneas 4020 a 4040 efectúa la actualización de posición y velocidad. (Como recordarás, la posición horizontal de la rana se guarda en F, su velocidad horizontal en B, su posición vertical en G, su velocidad vertical en A y la constante de gravedad en P). La línea 4060 trata los rebotes con el borde superior de la pantalla. La línea 4090 comprueba si la rana está lo suficientemente arriba como para poder 'cazar' una mosca y recurre a **cazando** si lo está. El volver a exponer la rana es labor de la línea 4100. La línea 4110 comprueba si la rana está lo suficientemente baja como para poder rebotar en el trampolín y apela al procedimiento **botando** si así es.

Procedimientos botando, cazando y fallando

El primero de los procedimientos comprueba si la rana está en la posición adecuada para rebotar en el trampolín:

```
4500 DEFine PROCedure botando
4510 D%=F-X%-M
4520 IF D%<1 OR D%>2 THEN fallando:END DEFine botando
4530 M=NOT M
4540 A=-A
4550 G=20
4560 F=M*3+X%
4570 G%=G
4575 F%=F
4580 BEEP .1,10
4590 PAUSE 5
4600 BEEP
4699 END DEFine botando
```

Dado que la posición horizontal de la rana está registrada en F y la posición horizontal de la esquina superior izquierda del trampolín lo está en X%, la diferencia entre las dos puede usarse para descubrir si la rana está tocando sobre el trampolín o no. El único problema es que el lado del trampolín sobre el que la rana tiene que caer depende de si el trampolín está inclinado a izquierda o a derecha. Si lo está a la izquierda, entonces la diferencia tiene que ser 2 ó 3 para poder rebotar. Si lo está a derecha, esa diferencia tiene que ser 1 ó 2. Si restamos la variable M, que es 1 para trampolín a izquierdas y 0 en demás, esta diferencia puede hacerse que sea o bien 1 o bien 2 para un 'aterrizaje' correcto. Esta diferencia corregida se calcula mediante la línea 4510 y la prueba de si el rebote es correcto se lleva a cabo en la línea 4520 que apela al procedimiento **fallando** si la rana no ha logrado el rebote del trampolín. Si la rana lo ha logrado, la línea 4530 cambia el trampolín a izquierdas para que pase a derechas, y viceversa. La línea 4540 invierte la velocidad vertical de la rana, haciendo que rebote efectivamente hacia fuera del trampolín. Las líneas 4550 a 4575 actualizan las coordenadas de la rana para que sean ahora las correspondientes a la otra rana del extremo del trampolín.

Si la rana falla en el rebote, se recurre a **fallando** para sacar en acción otra nueva rana (a no ser desde luego, que ya se hayan empleado las diez disponibles).

```
8000 DEFine PROCedure fallando
8010 porte_grafo F%,G%,5
8020 RANA=RANA+1
8030 IF RANA<10 THEN
8040  BEEP .1,-10
8050  PAUSE 6
8060  BEEP
8070  M=1
8080  saque_rana
8090 ELSE
8100  ACABADO=CERTO
8110 END IF
8199 END DEFine fallando
```

Si ya se han usado las diez ranas, la línea 8100 pone ACABADO al valor CERTO y así lleva el juego al final. En los demás casos, el balancín se coloca inclinado a la izquierda si es necesario mediante la línea 8070, y luego la línea 8080 apela a **saque_rana** para continuar el juego con una nueva rana.

El procedimiento **cazando** simplemente usa la función **tinte,** descrita en el Capítulo Uno, para inspeccionar el color de un punto concreto dentro del cuadratín al que la rana va a moverse. Si es "amarillo" entonces la rana ha cazado una mosca.

```
7000 DEFine PROCedure cazando
7010 IF tinte(G%,F%,4,4)<>1 THEN END DEFine cazando
7020 BEEP .1,20
7030 PAUSE 2
7040 BEEP
7050 SC=SC+1
7060 AT 21,2
7070 PRINT 'RANA ';RANA ;'PUNTOS= ';SC
```

```
7080 IF SC=76 THEN ACABADO=CERTO
7099 END DEFine cazando
```

Si la rana caza una mosca, todo lo que sucede es que se actualiza la puntuación (línea 7050) y se comprueba para ver si ya se han cazado todas las moscas (línea 7080).

Procedimiento fine_juego

Este procedimiento simplemente expone unos cuantos mensajes acordes con la puntuación final alcanzada y luego pregunta si se requiere otro nuevo juego.

```
9000 DEFine PROCedure fine_juego
9010 CLS
9020 AT 10,3
9030 SELect ON SC
9040  ON SC=0 TO 10
9050   PRINT 'Un simple renacuajo lo haria mejor'
9060  ON SC=11 TO 20
9070   PRINT 'Tienen tres patas tus ranas?'
9080  ON SC=21 TO 40
9090   PRINT 'No esta mal para un sapo miope'
9100  ON SC=41 TO 50
9110   PRINT 'Bastante bien considerando...'
9120  ON SC=51 TO 60
9130   PRINT 'Un enorme atracon de moscas'
9140  ON SC=REMAINDER
9150   PRINT 'Inscribe a las ranas en las olimpiadas'
9160 END SELect
9170 AT 15,5
9180 PRINT 'Tus puntos ';SC
9190 REPeat otro_juego
9200  INPUT 'Otro juego (S/N)?';A$
9210  IF A$='S' OR A$='N' THEN EXIT otro_juego
9220 END REPeat otro_juego
9230 IF A$='S' THEN
9240  OTRA=CERTO
9250 ELSE
9260  OTRA=FALSO
9270 END IF
9299 END DEFine fine_juego
```

Evaluación y mejoras

En su forma actual Salta Rana ciertamente se ejecuta lo suficientemente rápido con su máxima dificultad para mantener al jugador ocupado y lo suficientemente lento a su máxima facilidad para que el principiante aprenda el juego. El juego es divertido y hay claramente ocasiones para mejorarlo en la forma de efectos sonoros y animación extra, pero el problema principal con el juego es que sólo ejerce una suerte de habilidad y puede convertirse en aburrido.

El principal reto del juego tal y como está escrito actualmente, es que es difícil situar el balancín bajo la rana que sale, y así constituye el interés inicial del juego. Una vez que has aprendido cómo controlar el trampolín, hay muy poco que puedas hacer para incrementar el número de insectos que te apañes para comer, que no seas manteniendo la rana rebotando todo lo más posible. En algunas maneras, la versión presente del juego desperdicia el uso del balancín para lanzar la rana al aire porque lo trata como si fuera un sencillo bate. Sin embargo, no es difícil mejorar eso.

Si piensas sobre ello durante un momento, verás que la rana que es lanzada fuera del trampolín debe volar según la dirección que corresponde a la inclinación del trampolín, izquierda o derecha, cuando se efectúa el rebote. Por el momento, la rana deja el trampolín con la misma velocidad horizontal que la rana que cae sobre él. Eso significa que si una rana moviéndose a la izquierda cae sobre el trampolín, la rana que sale del trampolín continúa moviéndose a la izquierda. Sin embargo, no es difícil darse cuenta que una rana que saliera catapultada de un trampolín inclinado a la izquierda debiera moverse a la derecha y una que es catapultada desde un trampolín a derecha, debiera moverse hacia la izquierda, sin importar cuál es el camino entrante que sigue la rana que cae. Es bastante fácil efectuar este cambio en el juego añadiendo la siguiente línea al procedimiento **botando:**

```
4525 IF M=1 THEN
4526  B=ABS(B)
4527 ELSE
4528  B=-ABS(B)
4529 END IF
```

Así, ciertamente se altera como la rana rebota en el trampolín de manera que hace el juego más interesante, pero todavía no hay ninguna exigencia en la habilidad necesaria para el juego. Para incrementar esa habilidad, la cosa obvia a hacer es dar al jugador control sobre qué trampolín está usando. Eligiendo el trampolín sobre el que la rana caerá, el jugador puede en un cierto grado dirigir el movimiento hacia donde queden más moscas. El siguiente añadido permite al jugador cambiar entre las dos versiones del trampolín simplemente pulsando la tecla de flecha ascendente:

```
3035 IF A=4 THEN bascule

4700 DEFine PROCedure bascule
4710 M=NOT M
4720 PAUSE 10
4799 END DEFine bascule
```

El tiempo de demora en **bascule** (línea 4720) es necesario para permitir a la flecha ascendente que cambie el trampolín a un ritmo razonable de manera que el jugador pueda cesar de pulsar la tecla cuando el trampolín correcto esté expuesto en pantalla.

Con esta modificación, el juego es bastante diferente y el jugador no sólo está involucrado en la labor bastante simple de mover el trampolín a la posición correcta, sino en la de elegir con qué trampolín se comerá más moscas. Hay muchas posibilidades más de usar el trampolín para controlar la manera en que se mueve la rana. Por ejemplo, se puede hacer que la altura del rebote dependa exactamente de dónde ha caído la rana dentro del trampolín; la velocidad horizontal puede hacerse también dependiente de cómo se estaba moviendo el trampolín justamente antes de la caída. La rana puede perder velocidad cada vez que se come una mosca... pero la elección y la implementación de estas modificaciones se deja a tu imaginación.

La versión final - un listado completo

```
 10  REMark Ranas Saltarinas
 20  WINDOW 512,256,0,0
 25  prologue
 30  prepa_juego
 35  REPeat manga
 36   prepa_manga
 50   expo_moscas
 60   saque_rana
 70   REPeat movida
 80    mueva_polin
 90    mueva_rana
100   IF ACABADO THEN EXIT movida
110  END REPeat movida
120  fine_juego
130  IF NOT OTRA THEN EXIT manga
140 END REPeat manga
199 STOP

350 DEFine PROCedure prepa_juego
360 DATA -5245,-10109,-7806,-11646
370 DATA -6783,1665,2,0,8769,8316
380 DATA 0,0,9264,18432,8834,-11524,128
390 DATA 22664,-20228,32,26850,28672,20085
400 dire_rus=RESPR(100)
410 RESTORE 360
420 FOR I=dire_rus TO dire_rus+22*2  STEP 2
430  READ B
440  POKE_W I,B
450 END FOR I
510 C_TAB=RESPR(32*9)
520 prepa_grafo 0,2,6,130,84,56,186,254,130,68,255
530 prepa_grafo 1,2,6,1,2,4,24,56,88,152,24
540 prepa_grafo 2,2,6,252,0,0,0,0,0,0,0
```

```
550 prepa_grafo 3,1,6,36,24,255,255,126,60,102,66
560 prepa_grafo 4,4,6,130,84,56,186,254,130,68,130
570 prepa_grafo 5,6,6,255,255,255,255,255,255,255,255
575 prepa_grafo 6,2,2,255,255,255,255,255,255,255,255
576 prepa_grafo 7,2,6,128,64,32,24,28,26,25,24
577 prepa_grafo 8,2,6,63,0,0,0,0,0,0,0
580 FALSO=0
590 CERTO=1
599 END DEFine prepa_juego

1000 DEFine PROCedure prepa_manga
1010 M=1
1030 X%=23
1040 RANA=1
1050 ACABADO=FALSO
1060 OTRA=FALSO
1070 SC=0
1099 END DEFine prepa_manga

1300 DEFine PROCedure prepa_grafo(S%,F%,B%,
     R0%,R1%,R2%,R3%,R4%,R5%,R6%,R7%)
1310 LOCal A,FH%,FL%,BH%,BL%
1320 A=C_TAB+32*S%
1330 FH%=(F% DIV 4)*2
1340 FL%=F% && 3
1350 BH%=(B% DIV 4)*2
1360 BL%=B% && 3
1370 prepa_raya(R0%)
1380 prepa_raya(R1%)
1390 prepa_raya(R2%)
1400 prepa_raya(R3%)
1410 prepa_raya(R4%)
1420 prepa_raya(R5%)
1430 prepa_raya(R6%)
1440 prepa_raya(R7%)
1499 END DEFine prepa_grafo

1500 DEFine PROCedure prepa_raya(R%)
1502 LOCal M%,J,I,DL%,DH%
1503 M%=128
1504 FOR J=1 TO 2
1505  DL%=0:DH%=0
1510  FOR I=1 TO 4
1520   IF (R% && M%)=M% THEN
1530    DL%=FH%+DL%*4
1540    DH%=FL%+DH%*4
1550   ELSE
1560    DL%=BH%+DL%*4
1570    DH%=BL%+DH%*4
1580   END IF
1590   M%=M% DIV 2
1600  END FOR I
1610  POKE A,DL%
```

```
1620  POKE A+1,DH%
1630  A=A+2
1640 END FOR J
1699 END DEFine prepa_raya

1700 DEFine PROCedure porte_grafo(X%,Y%,S%)
1710 CALL dire_rus,X%,Y%,S%,C_TAB
1799 END DEFine porte_grafo

2000 DEFine PROCedure expo_moscas
2010 CLS
2020 C%=0
2030 FOR V=2 TO 10 STEP 2
2040  C%=NOT C%
2050  FOR U=2 TO 30 STEP 2
2060   porte_grafo U-C%,V,3
2070  END FOR U
2080 END FOR V
2099 END DEFine expo_moscas

3000 DEFine PROCedure mueva_polin
3005 LOCal A
3010 A=KEYROW(1)
3020 IF A=16 AND X%<28 THEN X%=X%+1
3030 IF A=2 AND X%>0 THEN X%=X%-1
3035 IF A=4 THEN bascule
3040 expo_polin
3099 END DEFine mueva_polin

3100 DEFine PROCedure expo_polin
3110 SELect ON M
3120  ON M=1
3130   porte_grafo X%,21,5
3140   porte_grafo X%+1,21,0
3150   porte_grafo X%+2,21,1
3180   porte_grafo X%+3,21,2
3220   porte_grafo X%+4,21,5
3240  ON M=0
3250   porte_grafo X%,21,5
3260   porte_grafo X%+1,21,8
3270   porte_grafo X%+2,21,7
3280   porte_grafo X%+3,21,0
3290   porte_grafo X%+4,21,5
3360 END SELect
3999 END DEFine expo_polin

4000 DEFine PROCedure mueva_rana
4010 porte_grafo F%,G%,5
4020 F=F+B
4030 G=G+A
4040 A=A+P
4050 IF F>30 OR F<2 THEN B=-B
4060 IF G<2 THEN G=2:A=-A
```

```
4070 G%=G
4080 F%=F
4090 IF G%<11 THEN cazando
4100 porte_grafo F%,G%,4
4110 IF G%>20 THEN botando
4199 END DEFine mueva_rana

4500 DEFine PROCedure botando
4510 D%=F-X%-M
4520 IF D%<1 OR D%>2 THEN fallando:END DEFine botando
4525 IF M=1 THEN
4526  B=ABS(B)
4527 ELSE
4528  B=-ABS(B)
4529 END IF
4530 M=NOT M
4540 A=-A
4550 G=20
4560 F=M*3+X%
4570 G%=G
4575 F%=F
4580 BEEP .1,10
4590 PAUSE 5
4600 BEEP
4699 END DEFine botando

4700 DEFine PROCedure bascule
4710 M=NOT M
4720 PAUSE 10
4799 END DEFine bascule

5000 DEFine PROCedure saque_rana
5010 mueva_polin
5020 FOR I=26 TO 31
5030  porte_grafo I,15,6
5040 END FOR I
5050 FOR F=31 TO 24 STEP -1
5060  porte_grafo F+1,14,5
5070  porte_grafo F,14,4
5080  mueva_polin
5090 END FOR F
5100 FOR G=14 TO 19
5110  porte_grafo 24,G-1,5
5120  porte_grafo 24,G,4
5130  mueva_polin
5140 END FOR G
5145 porte_grafo 24,19,5
5150 F%=24
5155 F=F%
5160 G%=19
5165 G=G%
5170 A=.5+RND/10
5180 B=-(.2+RND/2.5)
```

```
5190 P=9E-3
5200 AT 21,2
5210 PRINT 'RANA ';RANA;' PUNTOS=';SC
5220 A=A+A/D
5230 B=B+B/D
5240 botando
5299 END DEFine saque_rana

6000 DEFine PROCedure prologue
6010 PAPER 6
6015 INK 1
6020 CLS
6030 AT 5,10
6040 PRINT 'R A N A S  S A L T A R I N A S'
6050 AT 10,4
6060 PRINT 'En este juego debes comerte las moscas'\\
6070 PRINT TO 3;'haciendo que tus dos ranas salten'\\
6080 PRINT TO 3;'del balancin usando las flechas
     izquierda y derecha'\\
6090 PRINT TO 3;'Dispones de diez ranas para dejar
     limpia'\\
6100 PRINT TO 3;'la pantalla .....'
6110 REPeat dif
6120  AT 21,5
6130  PRINT 'Dime nivel de dificultad'
6140  PRINT TO 8;'1 (dificil) hasta 10 (facil)';
6150  INPUT D
6160  IF D>0 AND D<11 THEN EXIT dif
6170 END REPeat dif
6199 END DEFine prologue

7000 DEFine PROCedure cazando
7010 IF tinte(G%,F%,4,4)<>1 THEN END DEFine cazando
7020 BEEP .1,20
7030 PAUSE 2
7040 BEEP
7050 SC=SC+1
7060 AT 21,2
7070 PRINT 'RANA ';RANA;' PUNTOS=';SC
7080 IF SC=76 THEN ACABADO=CERTO
7099 END DEFine cazando

8000 DEFine PROCedure fallando
8010 porte_grafo F%,G%,5
8020 RANA=RANA+1
8030 IF RANA<10 THEN
8040  BEEP .1,-10
8050  PAUSE 6
8060  BEEP
8070  M=1
8080  saque_rana
8090 ELSE
8100  ACABADO=CERTO
```

```
8110 END IF
8199 END DEFine fallando

9000 DEFine PROCedure fine_juego
9010 CLS
9020 AT 10,3
9030 SELect ON SC
9040  ON SC=0 TO 10
9050   PRINT 'Un simple renacuajo lo haria mejor'
9060  ON SC=11 TO 20
9070   PRINT 'Tienen tres patas tus ranas?'
9080  ON SC=21 TO 40
9090   PRINT 'No esta mal para un sapo miope'
9100  ON SC=41 TO 50
9110   PRINT 'Bastante bien considerando ...'
9120  ON SC=51 TO 60
9130   PRINT 'Un enorme atracon de moscas'
9140  ON SC=REMAINDER
9150   PRINT 'Inscribe a las ranas en las olimpiadas'
9160 END SELect
9170 AT 15,5
9180 PRINT 'Tus puntos ';SC
9190 REPeat otro_juego
9200 INPUT 'Otro juego (S/N)?';A$
9210 IF A$='S' OR A$='N' THEN EXIT otro_juego
9220 END REPeat otro_juego
9230 IF A$='S' THEN
9240  OTRA=CERTO
9250 ELSE
9260  OTRA=FALSO
9270 END IF
9299 END DEFine fine_juego

9500 DEFine FuNction tinte(R%,C%,X%,Y%)
9510 LOCal A%,P%,B%,A
9520 P%=C%*8+X%
9530 A=131072+(512*R%+Y%*64+P% DIV 4)*2
9540 B%=PEEK(A+1)
9550 A%=PEEK(A)
9560 P%=P% MOD 4
9570 IF P%=3 THEN RETurn ((2 && A%)*2)+(3 && B%)
9580 IF P%=2 THEN RETurn ((8 && A%)DIV 2)+
     ((12 && B%) DIV 4)
9590 IF P%=1 THEN RETurn ((32 && A%)DIV 8)+
     ((48 && B%) DIV 16)
9600 IF P%=0 THEN RETurn ((128 && A%) DIV 32)+
     ((192 && B%) DIV 64)
9699 END DEFine tinte
```

Capítulo Cuatro

Ranurbando

Las ranas juegan el papel protagonista en éste al igual que en el juego del último capítulo. Ranurbas es una versión del clásico juego de las Ranas Urbanas. El objeto del juego es conseguir que una rana cruce una carretera de mucho tráfico, esquivando los vehículos, y luego a través de un rio de rápida corriente saltando sobre troncos flotantes. La implementación de este juego en BASIC proporciona la oportunidad de explicar y experimentar con una técnica de animación que se basa en el **des_rolle** o deslizamiento del fondo de la imagen con respecto a un motivo gráfico situado como frente o vanguardia de la escena.

Animación por des_rolle

El término des_rolle, y por desgracia el equivalente inglés "scrolling" (que recuerda la época de los rollos de papiro o de las 'chuletas' de estudiante que se desenrrollaban hasta llegar al tema del exámen) se ha hecho ahora un término familiar al hablar del corrimiento de la imagen con respecto a la pantalla. Todos estamos acostumbrados a la idea que cuando la pantalla se llena de texto, una instrucción sucesiva PRINT hará que toda la imagen se desplace hacia arriba una línea, perdiéndose así la línea superior y dejando por la parte inferior el espacio libre necesario para la nueva. Lo que no es tan familiar es que esta posibilidad de **des_rolle** puede ser útil como método de animación de motivos gráficos. Los problemas asociados con los intentos de expresar una gran cantidad de símbolos lo suficientemente rápidos como para hacer la animación posible ya nos los hemos encontrado en los primeros dos juegos de este libro. Con estos problemas en mente, seremos capaces de apreciar que el des_rolle de imagen es una manera de mover un gran número de caracteres en un tiempo muy corto y con muy poco esfuerzo de programación. Por ejemplo, ensaya con:

```
10 MODE 8
20 AT 19,31: PRINT
30 PRINT TO RND(1 TO 31);"*";
40 GO TO 20
```

Este programa expone asteriscos aleatoriamente en la fila inferior de la pantalla y luego obtiene la animación de la imagen usando el deslizamiento automático ascendente. Como este programa demuestra, es posible animar un gran número de objetos usando esta técnica. De hecho, el problema de la animación mediante deslizamiento vertical de la imagen consiste en hacer que ¡lo que es animado esté realmente quieto!

Eso puede hacerse exponiendo los objetos al final de un des_rolle de la imagen y borrándolos antes del siguiente.

La animación mediante des_rolle es un candidato posible siempre que un juego consta de un gran número de objetos moviéndose a la misma velocidad y en la misma dirección, con quizás uno o dos de esos motivos gráficos teniendo un movimiento de diferente pauta. En el caso de las Ranas Urbanas, el tráfico de la carretera y los troncos que flotan en el rio constituyen ese gran número de objetos movibles mencionados y las ranas que atraviesan la carretera y el rio es ese motivo sencillo que se mueve de forma diferente al resto de los motivos gráficos.

El diseño del juego

La idea de usar la animación por deslizamiento vertical de la imagen para 'implementar', i.e. para llevar a cabo una versión del juego, implica que los vehículos y los troncos tienen que moverse hacia arriba de la pantalla y la rana horizontalmente a través de la pantalla.

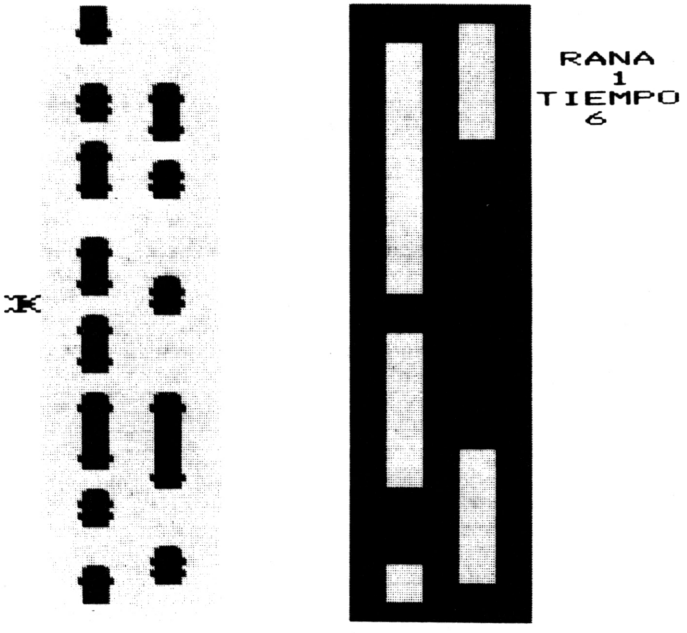


Fig. 4.1.

Para hacer el juego razonablemente interesante tiene que haber como mínimo dos carriles de tráfico y dos ramblas de troncos flotantes. Comenzando desde el extremo izquierdo de la pantalla, el jugador tiene que hacer que la rana esquive el tráfico y llegue a la banda de seguridad entre la carretera y el rio. El rio tiene que cruzarlo brincando sobre los troncos. Observa que mientras que la rana en la carretera permanece estacionaria a no ser que el jugador haga que brinque, cuando está sobre un tronco flotante es arrastrada por la corriente del rio. La rana tiene que alcanzar la otra orilla antes de que el tronco sobre el que se asienta sea llevado fuera de la pantalla. Para aumentar la presión sobre el jugador que mueve la rana, obligaremos a un tiempo límite. Además, mientras el jugador puede mover a la rana arriba y abajo de la pantalla y hacia la derecha a través de la carretera y el rio, no puede moverla a la izquierda para retirarse a lugar seguro -una vez que la rana comienza a avanzar debe siempre continuar su avance. La escena del juego puede verse en la Fig. 4.1.

El uso de las ventanas de texto

El único problema con el uso del corrimiento de imágenes para animar el tráfico y lo troncos en este juego, es que este método hace que todos se muevan al mismo ritmo y al mismo tiempo. Estos movimientos 'sincronizados' hacen el juego demasiado fácil y aburrido. El problema de cruzar por la carretera por ejemplo, se convierte en esperar que aparezca el hueco adecuado entre las dos filas de vehículos y luego brincar a lo ancho tan rápidamente como sea posible. El cruce del rio es incluso más fácil. Todo lo que tienes que hacer es esperar que aparezcan dos troncos próximos uno al otro y entonces saltar al primer tronco y de ahí inmediatamente al segundo.

Si el tráfico se mueve en ambas direcciones en cada carril y las ramblas de troncos se mueven a diferente ritmo, entonces el juego se convierte en algo mucho más interesante. Intentando apreciar los huecos es mucho más difícil cuando el tráfico se desplaza en ambas direcciones. También es difícil decidir cuándo se ha de cruzar un rio en que los troncos se mueven a diferente velocidad. Supongamos que la segunda rambla de troncos se mueve el doble de rápido que la primera. En este caso, cualquier 'solape' entre troncos de las dos hileras es sólo momentáneo -los troncos de la segunda rambla adelantan continuamente aquéllos de la primera y la rana puede saltar sobre un tronco encontrándose únicamente que el destino del salto ha sido llevado rio arriba.

Obviamente que para un buen juego tenemos que encontrar alguna manera de hacer que el tráfico en los dos carriles y los troncos de las dos ramblas se muevan con cadencias diferentes. Eso suena como un problema difícil, pero de hecho es muy fácil usando la facilidad de **ventanas de texto** del QL. Usando sus **cauces** de comunicación con la pantalla es posible delimitar el área de la pantalla usada como un 'escaparate' determinado -i.e. crear una ventana o vitrina para exposición de texto. Desde nuestro punto de vista, la faceta más importante de estas ventanas es que se comportan exactamente igual que si les correspondiera toda la pantalla, incluyendo la forma en que des_rolla el texto expuesto en ella cuando se quiere exponer algo en la línea inferior de la ventana. La idea es usar una ventana que tenga una anchura de **un cuadratín** y una altura de 32 filas, para cada carril de tráfico y para cada reguero de troncos, y hacer que la imagen expuesta a través de estas ventanas se des_rolle independientemente en cada una.

Por ejemplo, ensaya tecleando el comando que **abre** la ventana número 4, es decir:

```
OPEN #4,scr_20x256a50x0
```

y luego haz que liste a través de ese cauce de pantalla (SCReen), mediante:

```
LIST #4
```

Encontrarás que detrás de ese comando toda la salida queda confinada a una única columna de texto sobre la pantalla. Después de que hayas visto esta demostración trivial, no es difícil imaginarse cómo puede usarse esta facultad para animar el movimiento de los vehículos y de los troncos en el juego en cuestión.

Programa principal y procedimiento prologue

En este juego, la **rutina principal** es un poco diferente a las que hemos presentado anteriormente y contiene algunas instrucciones que no son meras citas de procedimientos, que más tarde confeccionemos:

```
 10  REMark ranas urbanas
 20  prologue
 30  prepa_juego
 40  REPeat movida
 50   FOR RANA=1 TO 3
 60    prepa_manga
 70    AT 2,32:PRINT 'RANA'
 80    AT 3,35:PRINT RANA
 90    SEG=0
100    SDATE 2000,1,1,0,0,0
110    REPeat saltada
120     des_rolle
130     mire_hora
140     mueva_rana
150     IF SEG-5>0 THEN
160      AT 5,34:PRINT 'TIEMPO'
170      AT 6,35:PRINT SEG
180     END IF
190     IF SEG-10>MAX OR ACABADO THEN EXIT saltada
200    END REPeat saltada
210    mate_pase
220   END FOR RANA
230   fine_juego
240   IF OTRA=FALSO THEN EXIT movida
250  END REPeat movida
299  STOP
```

Primero se apela al procedimiento **prologue** que simplemente expone las reglas del juego. Luego se cita **prepa_juego** para implantar los motivos gráficos definidos por el usuario. El resto del programa principal está en la forma de tres bucles anidados. Las líneas 40 a 250 forman un bucle condicionado que desarrolla el juego. Las líneas 50 a 220 contienen un bucle preconfinado que repite el núcleo principal del juego tres veces con tres diferentes ranas. El bucle más interno, en las líneas 110 a 200, es el bucle de animación encargado de los saltos de las ranas. El procedimiento **prepa_manga**, al que se recurre en la línea 60, efectúa la labor habitual de establecer las condiciones iniciales para todo lo necesario. Las líneas 70 y 80 exponen el número de orden de la rana corriente antes de que propiamente comience el juego. El bucle de animación apela al procedimiento **des_rolle** para 'animar' la carretera y el rio y después de inspeccionar el tiempo usando **mire_hora** en la línea 130, recurre al procedimiento **mueva_rana** para permitir que el jugador desplace dicho motivo. Finalmente, las líneas 150 a 180 exponen la HORA corriente y en la línea 190 se comprueba si el juego se ha acabado. El procedimiento **mate_pase** señala o bien la muerte de otra rana o el éxito en conseguir pasar al otro lado. Todo el juego llega a un final adecuado mediante el procedimiento **fine_manga** que también pregunta si se desea jugar OTRA.

El otro aspecto digno de mención es el uso de mención es el uso de SEG-10 para hacer que SALGA del bucle interno, mediante la instrucción EXIT de la línea 190. Eso es debido a que durante los primeros diez segundos en que está en marcha el bucle de animación, la rana no se puede mover. Esta demora inicial de diez segundos es necesaria para permitir que los coches y los troncos se desplacen hacia la parte superior de la pantalla y así hagan que el juego sea un reto. Por lo tanto, la secuencia de iniciación es que se trazan la carretera y el rio, se animan los vehículos y troncos desplazándolos arriba de la pantalla durante diez segundos, y luego aparece la rana y comienza el cronometraje.

El procedimiento **prologue** es tan simple que el listado se presenta sin ningún comentario adicional:

```
8000 DEFine PROCedure prologue
8010 PAPER 0
8020 INK 4
8030 WINDOW 512,256,0,0
8040 CLS
8050 AT 2,10:PRINT 'R A N A S  U R B A N A S'
8060 AT 5,9: PRINT 'En este juego'\
8070 PRINT TO 11;'tienes que guiar'\
8080 PRINT TO 9;'tres ranas para cruzar'\
8090 PRINT TO 10;'una carretera de mucho trafico'\
8100 PRINT TO 9;'y un rio caudaloso'\\
8110 PRINT TO 9;'Usa las flechas'\
8120 PRINT TO 11;'arriba, abajo'\
8130 PRINT TO 12;'y derecha'\\
8140 PRINT TO 9;'No las metas debajo de un'\
8150 PRINT TO 10;'coche y que no'\
8160 PRINT TO 9;'caigan en el rio.'\\
8170 PRINT TO 11;'Pulsa cualquier tecla'
8180 PRINT TO 13;'para comenzar'
8190 A$=INKEY$(-1)
```

```
8192 AT 22,12
8194 FLASH 1
8196 PRINT 'POR FAVOR ESPERA'
8198 FLASH 0
8199 END DEFine prologue
```

Procedimientos prepa_juego, prepa_manga y mire_hora

El procedimiento **prepa_juego** opera de la manera usual, preparando la forma de los caracteres que componen los motivos gráficos e implantando el código máquina usado por **porte_grafo** (recuerda que puedes consultar el Capítulo Uno para ver los detalles de este procedimiento y también de los que intervienen en él como **prepa_**, **prepa_grafo** y la función definida por el usuario **tinte**).

```
350 DEFine PROCedure prepa_juego
360 DATA -5245,-10109,-7806,-11646
370 DATA -6783,1665,2,0,8769,8316
380 DATA 0,0,9264,18432,8834,-11524,128
390 DATA 22664,-20228,32,26350,28672,20085
400 dire_rus=RESPR(100)
410 RESTORE 360
420 FOR I=dire_rus TO dire_rus+22*2 STEP 2
430 READ B
440 POKE_W I,B
450 END FOR I
500 RANDOMISE
510 C_TAB=RESPR(32*10)
520 prepa_grafo 0,1,0,255,255,255,255,255,255,255,255
530 prepa_grafo 1,2,0,255,255,255,255,255,255,255,255
540 prepa_grafo 2,0,0,255,255,255,255,255,255,255,255
550 prepa_grafo 3,1,2,60,126,126,126,126,255,255,255
560 prepa_grafo 4,1,2,126,126,126,126,126,126,126,126
570 prepa_grafo 5,1,2,126,126,255,255,255,126,126,126
580 prepa_grafo 6,2,1,255,255,255,255,255,255,255,255
590 prepa_grafo 7,4,0,185,82,28,30,28,82,185,0
600 prepa_grafo 8,4,1,185,82,28,30,28,82,185,0
610 prepa_grafo 9,4,2,185,82,28,30,28,82,185,0
620 OTRA=0
699 END DEFine prepa_juego
```

Las líneas 520 a 610 definen los vehículos, troncos, rana y los necesarios caracteres en blanco para el juego (véase Tabla 4.1). Como es habitual, tienen que prepararse diferentes caracteres para representar el mismo motivo sobre cada uno de los diferentes colores de fondo.

Tabla 4.1. Motivos gráficos usados

Número	Forma	Frente	Fondo
0	espacio	azul	negro
1	espacio	rojo	negro
2	espacio	negro	negro
3	frente coche	azul	rojo
4	mitad camión	azul	rojo
5	final coche	azul	rojo
6	tronco	rojo	azul
7	rana	verde	negro
8	rana	verde	azul
9	rana	verde	rojo

Puedes ver cómo se construyen los coches y camiones en la Fig. 4.2. Puede que te sorprenda ver que hemos elegido coches azules sobre una carretera roja y troncos rojos sobre un rio azul, pero esa particularidad simplificará el juego un montón. Cuando la rana está cruzando la carretera debe evitar coches azules, y cuando está cruzando el rio lo que debe evitar es el agua azul. Y de esta manera, a través de todo el juego el color azul indica un área donde la rana no debe estar.

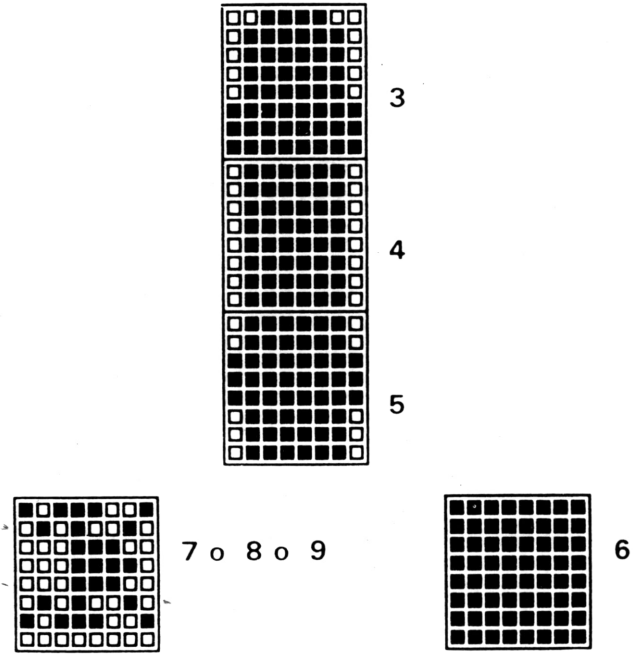


Fig. 4.2. Motivos gráficos usados para camión, rana y tronco.

El procedimiento **prepa_manga** define las variables, y en este caso también es responsable de exponer en pantalla las bandas coloreadas que representan la carretera y el rio y responsable de definir las ventanas necesarias para hacer que se desarrolle el movimiento de troncos y vehículos.

```
1000 DEFine PROCedure prepa_manga
1010 A%=1
1020 B%=A%
1030 C%=0
1040 D%=C%
1050 F%=8
1055 CH%=2
1060 XF=5
1070 YF=15
1075 CLS
1080 FOR K=0 TO 29
1090  FOR J=7 TO 12
1100   porte_grafo J,K,1
1110   porte_grafo J+9,K,0
1120  END FOR J
1130 END FOR K
1140 FALSO=0
1150 CERTO=1
1160 ACABADO=FALSO
1170 MAX=50
1180 OPEN #3, scr_16x240a128x0
1190 OPEN #4, scr_16x240a176x0
1200 OPEN #5, scr_16x240a272x0
1210 OPEN #6, scr_16x240a320x0
1220 PAPER #3,2
1230 PAPER #4,2
1240 PAPER #5,1
1260 PAPER #6,1
1299 END DEFine prepa_manga
```

Las líneas 1010 y 1020 ponen como valores iniciales de las A% y B% el valor 1 -el código para un cuadratín relleno de rojo. Igualmente, las líneas 1030 y 1040 colocan C% y D% a 0 -el código para un cuadratín azul. Estas variables son usadas por el procedimiento **des_rolle** y se describen mejor en esa sección. Las líneas 1050 a 1070 preparan inicialmente cuatro variables asociadas con el motivo rana. F% se pone a 8 y así refleja una rana sobre un fondo azul, i.e. la rana tal y como aparece al inicio del juego. CH% se fija a 2 y se usa para como carácter blanqueador de la rana. XF e YF son las coordenadas corrientes de la rana y sus valores iniciales se dan mediante las líneas 1060 y 1070.

Las líneas 1080 a 1130 exponen dos bandas de colores descendiendo por la pantalla. La banda roja del extremo izquierdo de la pantalla representa la carretera y la banda azul de la derecha representa el rio. Finalmente, la línea 1160 'baja' el testigo o banderín de ACABADO haciendo que sea FALSO y la línea 1170 establece el límite de tiempo previsto para el juego.

El procedimiento **mire_hora** saca el tiempo como un literal a partir de la función de fecha DATE$ disponible en el QL y disgrega ese literal con el fin de obtener la 'hora' como segundos transcurridos desde el comienzo del juego:

```
300 DEFine PROCedure mire_hora
310 T$=DATE$
320 SEG=T$(17)*60
330 SEG=SEG+T$(19 TO 20)
339 END DEFine mire_hora
```

Procedimiento des_rolle y los procedimientos implicados

La implementación del programa se desglosa limpiamente en dos secciones -la generación y el deslizamiento vertical de los vehículos y troncos, y el movimiento de la rana. El problema de animar la rana se tratará posteriormente. La animación de la carretera y el rio es efectuada mediante el procedimiento **des_rolle** y los procedimientos a que recurre, y es el tema de esta sección:

```
3000 DEFine PROCedure des_rolle
3010 A%=coji_coch (.2,.5,A%,1)
3020 vitrine 1,A%
3030 B%=coji_coch (.16,.5,B%,2)
3040 vitrine 2,B%
3050 C%=coji_tron (.16,.2,C%)
3060 vitrine 3,C%
3090 D%=coji_tron (.16,.2,C%)
3100 vitrine 4,C%
3110 D%=coji_tron (.1,.2,D%)
3120 vitrine 4,D%
3199 END DEFine des_rolle
```

A primera vista, el procedimiento **des_rolle** parece complicado pero sólo es debido a que el propósito de las funciones **cojí_coch** y **cojí_tron** y del procedimiento **vitrine** no son obvios de inmediato. Para comprender lo que sucede tenemos que pensar un poco sobre lo que se requiere en cada **ronda** del bucle de animación. Considera por ejemplo el primer carril de tráfico. En cada ronda de animación tenemos que decidir si exponer un espacio en blanco, el comienzo de un coche o camión, la sección media de un camión o la parte posterior de un camión o coche. Eso es lo que la función **cojí_coch** (PANT,PULT,OB%,N%) hace cada vez que se aplica.

PANT es la probabilidad de que brote un coche o un camión -es decir, controla el número de coches en la carretera. PULT es la probabilidad de que un coche o un camión desaparezca de la carretera -es decir, controla el número de coches como contraposición al de camiones y la longitud media de los camiones (el motivo coche está formado por una parte delantera y una parte trasera; y un camión es la misma parte delantera y la misma parte trasera que el del coche, pero con un carácter intercalado entre ellos para representar la parte media del camión (véase Fig. 4.2).

OB% es simplemente el código del último OBjeto gráfico definido por el usuario que haya aparecido en el carril. Por ejemplo, si el último motivo fue un espacio en blanco (código 1) entonces el siguiente motivo a exponer en el carril puede ser bien otro espacio en blanco, o una parte frontal de coche (y obviamente no puede ser ni una sección media de camión ni un extremo final de coche). Si el siguiente motivo es un espacio o un frontal de coche se decide aleatoriamente con una probabilidad que está determinada por el valor de PANT, ya que refleja la parte anterior del motivo. N% determina en cuál de las cuatro ventanas va a tener lugar el corrimiento vertical de la imagen.

En el procedimiento **des_rolle** la variable A% se usa para registrar el motivo expuesto en el carril uno de tráfico, y la línea 3010 aplica la función **cojí_coch** para actualizarlo. Esta función da como resultado de su aplicación el código del motivo gráfico que ha de exponerse a continuación. De la misma manera, B% se usa para reflejar el motivo expuesto en el carril dos de tráfico, y C% y D% respectivamente, los códigos identificativos de los motivos gráficos que han de aparecer en la primera y segunda rambla de troncos. La función **cojí_tron** efectúa la misma acción que para los vehículos pero corresponde a los troncos -es decir, da como resultado de su aplicación el código del siguiente motivo gráfico que ha de exponerse en pantalla, teniendo en cuenta el código del último motivo que se expuso. La acción de estas dos funciones se hará mucho más clara después de que hayan sido descritas con más detalle ulteriormente.

El procedimiento **vitrine** (N%,C%) es el que realmente efectúa la labor de exponer en una ventana un motivo y desplazar verticalmente toda la imagen. N% controla en cuál de las cuatro ventanas se va a exponer el motivo gráfico, y C% es el código de dicho motivo gráfico.

Debieras ahora ser capaz de comprender el esquema de **des_rolle.** La línea 3010 aplica **cojí_coch** para generar el carácter que va a exponerse en el primer carril de tráfico. Luego, en la línea 3020 se recurre a **vitrine** para que lo exponga en la ventana 1. De la misma manera, las líneas 3030 y 3040 escogen y exponen un carácter del motivo vehículo para la ventana 2. Las líneas 3050 y 3060 escogen y exponen un carácter del motivo tronco en la ventana de texto 3, y las líneas 3090 a 3120 escogen y exponen **dos** caracteres del motivo tronco en la ventana 4 de texto. Ya que la ventana 4 de texto se desrrolla dos veces por cada vez que se desrrolle la ventana 3 de texto, la segunda rambla de troncos se mueve el doble de rápido que la primera.

Como ya se ha mencionado, para comprender completamente cómo **des_rolle** trabaja, tienes que comprender las **funciones** que aplica y los procedimientos a los que apela. La primera de esas funciones es la función **cojí_coch:**

```
2000 DEFine FuNction coji_coch(PANT,PULT,OB%,N%)
2010 LOCal T%,F%,B%
2015 IF N%=2 THEN
2016  F%=3
2017  B%=5
2018 ELSE
2019  F%=5
2020  B%=3
2021 END IF
2025 T%=OB%
2030 IF OB%=1 AND RND<PANT THEN T%=F%
```

```
2040 IF OB%=F% THEN T%=4
2050 IF OB%=B% THEN T%=1
2060 IF T%=4 AND RND<PULT THEN T%=B%
2070 RETurn T%
2099 END DEFine coji_coch
```

Las líneas 2015 a 2020 colocan los caracteres correctos en las variables asignadas a la parte anterior y ulterior de los motivos gráficos. Con eso nos preocupamos del hecho de que en el carril uno el tráfico se mueve en dirección opuesta al del carril dos. La línea 2030 comprueba si el último carácter expuesto fue un espacio en blanco. Si así fue, entonces el siguiente carácter que ha de exponerse será o bien otro blanco o bien la parte anterior de un coche. Si el último carácter expuesto fue la parte anterior de un coche, entonces la línea 2040 hace que el siguiente carácter sea la parte media de un camión. La línea 2050 comprueba si el último carácter expuesto fue la parte ulterior o final de un coche y si así fue, obviamente el siguiente carácter será un espacio en blanco. Finalmente, la línea 2060 cambia aleatoriamente las secciones medias de camión a secciones traseras de coche.

La función **cojí_tron** es mucho más fácil que la anterior:

```
5000 DEFine FuNction coji_tron(PANT,PULT,OB%)
5010 IF OB%=0 AND RND<PANT THEN RETurn 6
5020 IF OB%=6 AND RND<PULT THEN RETurn 0
5030 RETurn OB%
5099 END DEFine coji_tron
```

Si el último carácter fue un espacio en blanco, entonces la línea 5010 hace que el siguiente carácter sea la parte anterior de un tronco con probabilidad PANT. La línea 5020 hace lo contrario y cambia un carácter del motivo tronco en un espacio en blanco con probabilidad PULT, al corresponder a la parte última del tronco. Debieras ser capaz de apreciar que PANT es la probabilidad de que brote la parte anterior de un tronco y PULT es la probabilidad de que se escoja la parte posterior de un tronco.

El procedimiento **vitrine** produce el des_rolle de texto en las ventanas al exponer otro carácter en ellas:

```
4000 DEFine PROCedure vitrine (N%,C%)
4010 SELect ON N%
4020  ON N%=1
4030   SCROLL #3,8
4040   porte_grafo 8,0,C%
4050  ON N%=2
4060   SCROLL #4,-8
4070   porte_grafo 11,29,C%
4080  ON N%=3
4090   SCROLL #5,-8
4100   porte_grafo 17,29,C%
4110  ON N%=4
4120   SCROLL #6,-8
4130   porte_grafo 20,29,C%
4140 END SELect
4199 END DEFine vitrine
```

La primera ventana des_rolla la imagen en dirección opuesta a todas las otras, así que si N% es 1, la línea 4020 des_rolla hacia abajo y la línea 4040 expone el carácter correspondiente en la parte superior de la pantalla. Las líneas 4060, 4090 y 4120 efectúan el des_rolle de la imagen respectivamente en las ventanas 2, 3 y 4. Estas tres ventanas efectúan el corrimiento hacia la parte alta de la pantalla y por tanto el carácter pertinente se expone en la parte baja de la imagen, mediante las instrucciones de las líneas 4070, 4100 y 4130.

Procedimientos mueva_rana, cambie_grafo y cace_fuera

En apariencia, el procedimiento **mueva_rana** tiene una tarea muy sencilla que hacer, y debiera ser similar a todos los otros procedimientos usados para mover objetos gráficos por la pantalla. Sin embargo, tendremos que solventar unas cuantas dificultades. En particular, el carácter usado para borrar y exponer la rana ha de cambiarse para que dependa de si la rana está en la carretera, en un tronco o en ninguna cosa de esas. Ese problema se tiene en cuenta recurriendo a otro procedimiento, **cambie_grafo** que es el responsable de definir cuál es la versión de rana y de espacios 'clareadores' que deberá usarse de acuerdo con la posición horizontal de la rana en pantalla. También siempre desarrolla el carácter clareador que ha de exponerse para borrar la rana en su vieja posición.

```
6000 DEFine PROCedure cambie_grafo
6010 SELect ON XF
6020  ON XF=1 TO 7
6030   CH%=2
6040   F%=7
6050   K%=0
6060  ON XF=8
6070   CH%=1
6080   F%=9
6090   K%=-1
6100  ON XF=11
6110   CH%=1
6120   F%=9
6130   K%=1
6140  ON XF=12 TO 15
6150   CH%=2
6160   F%=7
6170   K%=0
6180  ON XF=16 TO 20
6190   CH%=1
6200   F%=9
6210   K%=0
6220  ON XF=21 TO 26
6230   CH%=2
6240   F%=7
6250   K%=0
6260 END SELect
6499 END DEFine cambie_grafo
```

La manera en que la rana se mueve depende de si está dentro de las ventanas con des_rolle de imagen o no lo está; y esa es la tarea principal encargada al procedimiento **mueva_rana.** Por ejemplo, si está dentro de la primera ventana (el primer carril de la carretera) ha de desplazar la imagen hacia abajo; y si está en el segundo, al igual que con el resto del tráfico debe desplazarlo hacia arriba. Para mantenerla en el mismo lugar es necesario clarear la vieja posición de la rana y volver a exponerla. Pero si la rana está dentro de la tercera o cuarta ventana (las ramblas lenta y rápida de troncos respectivamente) entonces, y mientras que esté asentada sobre un tronco, deberá desplazarse hacia arriba junto con el resto de los troncos.

```
6500 DEFine PROCedure mueva_rana
6505 IF SEG<10 THEN END DEFine mueva_rana
6510 cambie_grafo
6520 porte_grafo XF,YF-K%,CH%
6530 I%=KEYROW(1)
6540 IF I%=4 AND YF>2 THEN YF=YF-1
6550 IF I%=128 AND YF<31 THEN YF=YF+1
6560 IF I%=16 THEN
6570  BEEP .1,-10:PAUSE 2:BEEP
6580  XF=XF+3
6590 END IF
6600 Z%=tinte(YF,XF,4,4)
6610 IF Z%=1 THEN
6620  F%=8
6630  ACABADO=CERTO
6640  porte_grafo XF,YF,F%
6645 ELSE
6650 cambie_grafo
6660 porte_grafo XF,YF,F%
6665 END IF
6670 SELect ON XF
6680  ON XF=17
6690   cace_fuera
6710  ON XF=20
6720   cace_fuera
6721   cace_fuera
6730  ON XF=18 TO 23
6740   ACABADO=CERTO
6750 END SELect
6799 END DEFine mueva_rana
```

La línea 6505 detiene el movimiento de la rana hasta que hayan transcurrido 10 segundos desde el inicio del juego. La línea 6520 expone el carácter blanqueador correcto en la posición correcta. Las líneas 6530 a 6590 cambian las coordenadas de la rana en función de la tecla de flecha que se haya pulsado.

La línea 6600 indaga el color del cuadratín sobre el que la rana va a saltar y conserva en Z% el valor obtenido. Si ese TINTE es azul, i.e. código de color 1, entonces la línea 6630 hace que sea CERTO la variable ACABADO. Por el contrario, la línea 6550 recurre a **cambie_grafo** para elegir un cuadratín con el color de fondo adecuado y luego expone la rana en él mediante la línea 6660.

Las líneas 6670 a 6750 examinan si la rana está sobre una de las ventanas 3 ó 4 de texto, y en ese caso se apela al procedimiento **cace_fuera** no sólo para actualizar la coordenada Y de la rana como resultado del corrimiento de la imagen, sino fundamentalmente para comprobar si la rana ha alcanzado el borde superior de la pantalla, quedando fuera de la imagen. La línea 6271 vuelve a apelar por segunda vez al procedimiento **cace_fuera** para tener en cuenta el hecho de que en la ventana 4 de texto el corrimiento de imagen se efectúa en pasos dobles. Finalmente, la línea 6730 inspecciona si la rana ha alcanzado con seguridad la orilla derecha del rio. Sólo queda dar el listado del procedimiento **cace_fuera**, y recordarte que la función **tinte** ya se presentó en el Capítulo Uno.

```
7200 DEFine PROCedure cace_fuera
7210 IF YF=0 THEN
7220  ACABADO=CERTO
7230 ELSE
7250  YF=YF-1
7260 END IF
7299 END DEFine cace_fuera
```

Procedimientos mate_pase y fine_manga

Los procedimientos que quedan por definir en el juego, **mate_pase** y **fine_manga**, son muy simples. El primero de ellos halla si la rana ha pasado su viaje con éxito, y si no ha sido así cómo ha encontrado la muerte, emitiendo notas fúnebres apropiadas. El procedimiento **fine_manga** pregunta al jugador si desea jugar otra manga.

```
7500 DEFine PROCedure mate_pase
7510 IF XF>20 THEN
7520  BEEP .1,10
7530  PAUSE 5
7540  BEEP
7550 ELSE
7560  BEEP .1,5
7570  PAUSE 2
7580  BEEP
7590 END IF
7600 PAUSE 50
7699 END DEFine mate_pase

8500 DEFine PROCedure fine_juego
8510 CLS
8520 REPeat inp
8530 AT 10,3
8540 PRINT 'Otro juego S/N?';
8550 INPUT A$
8560 IF A$(1)='S' OR A$(1)='N' THEN EXIT inp
8570 END REPeat inp
8580 IF A$(1)='S' THEN OTRA=CERTO
8599 END DEFine fine_juego
```

Evaluación y mejoras

La versión comentada funciona lo suficientemente rápido como para ser interesante y la animación por corrimiento de la imagen con respecto a la ventana -por **des_rolle-** produce una escena interesante y lograda con muy poca programación. Quizás un aspecto algo molesto del juego es el zumbido sonoro persistente. Eso puede ponerse fuera de lugar añadiendo un nuevo procedimiento **-croe_poco-** que a su vez recurre a otro procedimiento **-croe_algo-** para producir un sonido con un tono rápidamente ascendente y descendente que puede que a alguien recuerde el "croar" de las ranas.

```
9800 DEFine PROCedure croe_poco
9810 croe_algo
9820 BEEP
9830 PAUSE 2
9840 croe_algo
9850 BEEP
9899 END DEFine croe_poco

9900 DEFine PROCedure croe_algo
9910 BEEP .1,20,5,0,7,2
9920 PAUSE 6
9999 END DEFine croe_algo
```

Estos procedimientos pueden citarse para hacer un sonido más apropiado cuando la rana salta. Cambia la línea 6570 para que sea ahora:

```
6570 croe_poco
```

La adición de este sencillo efecto sonoro incrementa ciertamente tanto la tensión como el disfrute del juego, y debieras ensayar practicando con el juego tanto con el efecto como sin él. Incluso aunque este es el punto en que ya dejamos el programa, eso no significa que no haya ámbito para ampliar el juego. En concreto, puedes añadir el sistema de puntuación basado en la cantidad de ranas guiadas con seguridad a la otra orilla y el tiempo empleado en el trayecto, y luego alterar el procedimiento **fine_juego** para que exponga comentarios similares a los que vimos en el programa anterior Salta Rana.

La versión final - un listado completo

```
10 REMark ranas_urbanas
20 prologue
30 prepa_juego
40 REPeat movida
50  FOR RANA=1 TO 3
60   prepa_manga
70   AT 2,32:PRINT 'RANA'
80   AT 3,35:PRINT RANA
90   SEG=0
```

```
100   SDATE 2000,1,1,0,0,0
110   REPeat saltada
120    des_rolle
130    mire_hora
140    mueva_rana
150    IF SEG-5>0 THEN
160     AT 5,34:PRINT 'TIEMPO'
170     AT 6,35:PRINT SEG
180    END IF
190    IF SEG-10>MAX OR ACABADO THEN EXIT saltada
200   END REPeat saltada
210   mate_pase
220  END FOR RANA
230  fine_juego
240  IF OTRA=FALSO THEN EXIT movida
250 END REPeat movida
299 STOP

300 DEFine PROCedure mire_hora
310 T$=DATE$
320 SEG=T$(17)*60
330 SEG=SEG+T$(19 TO 20)
339 END DEFine mire_hora
350 DEFine PROCedure prepa_juego
360 DATA -5245,-10109,-7806,-11646
370 DATA -6783,1665,2,0,8769,8316
380 DATA 0,0,9264,18432,8834,-11524,128
390 DATA 22664,-20228,32,26350,28672,20085
400 dire_rus=RESPR(100)
410 RESTORE 360
420 FOR I=dire_rus TO dire_rus+22*2  STEP 2
430 READ B
440 POKE_W I,B
450 END FOR I
500 RANDOMISE
510 C_TAB=RESPR(32*10)
520 prepa_grafo 0,1,0,255,255,255,255,255,255,255,255
530 prepa_grafo 1,2,0,255,255,255,255,255,255,255,255
540 prepa_grafo 2,0,0,255,255,255,255,255,255,255,255
550 prepa_grafo 3,1,2,60,126,126,126,126,255,255,255
560 prepa_grafo 4,1,2,126,126,126,126,126,126,126,126
570 prepa_grafo 5,1,2,126,126,255,255,255,126,126,126
580 prepa_grafo 6,2,1,255,255,255,255,255,255,255,255
590 prepa_grafo 7,4,0,185,82,28,30,28,82,185,0
600 prepa_grafo 8,4,1,185,82,28,30,28,82,185,0
610 prepa_grafo 9,4,2,185,82,28,30,28,82,185,0
620 OTRA=0
699 END DEFine prepa_juego

1000 DEFine PROCedure prepa_manga
1010 A%=1
1020 B%=A%
1030 C%=0
```

```
1040 D%=C%
1050 F%=8
1055 CH%=2
1060 XF=5
1070 YF=15
1075 CLS
1080 FOR K=0 TO 29
1090  FOR J=7 TO 12
1100   porte_grafo J,K,1
1110   porte_grafo J+9,K,0
1120  END FOR J
1130 END FOR K
1140 FALSO=0
1150 CERTO=1
1160 ACABADO=FALSO
1170 MAX=50
1180 OPEN #3, scr_16x240a128x0
1190 OPEN #4, scr_16x240a176x0
1200 OPEN #5, scr_16x240a272x0
1210 OPEN #6, scr_16x240a320x0
1220 PAPER #3,2
1230 PAPER #4,2
1240 PAPER#5,1
1260 PAPER#6,1
1299 END DEFine prepa_manga

1300 DEFine PROCedure prepa_grafo(S%,F%,B%,
     R0%,R1%,R2%,R3%,R4%,R5%,R6%,R7%)
1310 LOCal A,FH%,FL%,BH%,BL%
1320 A=C_TAB+32*S%
1330 FH%=(F% DIV 4)*2
1340 FL%=F% && 3
1350 BH%=(B% DIV 4)*2
1360 BL%=B% && 3
1370 prepa_raya(R0%)
1380 prepa_raya(R1%)
1390 prepa_raya(R2%)
1400 prepa_raya(R3%)
1410 prepa_raya(R4%)
1420 prepa_raya(R5%)
1430 prepa_raya(R6%)
1440 prepa_raya(R7%)
1499 END DEFine prepa_grafo

1500 DEFine PROCedure prepa_raya(R%)
1510 LOCal M%,J,I,DL%,DH%
1520 M%=128
1530 FOR J=1 TO 2
1540  DL%=0:DH%=0
1550  FOR I=1 TO 4
1560   IF (R% && M%)=M% THEN
1570    DL%=FH%+DL%*4
1580    DH%=FL%+DH%*4
```

```
1590    ELSE
1600     DL%=BH%+DL%*4
1610     DH%=BL%+DH%*4
1620    END IF
1630    M%=M% DIV 2
1640   END FOR I
1650   POKE A,DL%
1660   POKE A+1,DH%
1670   A=A+2
1680 END FOR J
1699 END DEFine prepa_raya

1700 DEFine PROCedure porte_grafo(X%,Y%,S%)
1710 CALL dire_rus,X%,Y%,S%,C_TAB
1799 END DEFine porte_grafo

2000 DEFine FuNction coji_coch(PANT,PULT,OB%,N%)
2010 LOCal T%,F%,B%
2015 IF N%=2 THEN
2016  F%=3
2017  B%=5
2018 ELSE
2019  F%=5
2020  B%=3
2021 END IF
2025 T%=OB%
2030 IF OB%=1 AND RND<PANT THEN T%=F%
2040 IF OB%=F% THEN T%=4
2050 IF OB%=B% THEN T%=1
2060 IF T%=4 AND RND<PULT THEN T%=B%
2070 RETurn T%
2099 END DEFine coji_coch

3000 DEFine PROCedure des_rolle
3010 A%=coji_coch (.2,.5,A%,1)
3020 vitrine 1,A%
3030 B%=coji_coch (.16,.5,B%,2)
3040 vitrine 2,B%
3050 C%=coji_tron (.16,.2,C%)
3060 vitrine 3,C%
3090 D%=coji_tron (.16,.2,C%)
3100 vitrine 4,C%
3110 D%=coji_tron (.1,.2,D%)
3120 vitrine 4,D%
3199 END DEFine des_rolle

4000 DEFine PROCedure vitrine (N%,C%)
4010 SELect ON N%
4020  ON N%=1
4030   SCROLL #3,8
4040   porte_grafo 8,0,C%
4050  ON N%=2
4060   SCROLL #4,-8
```

```
4070    porte_grafo 11,29,C%
4080   ON N%=3
4090    SCROLL #5,-8
4100    porte_grafo 17,29,C%
4110   ON N%=4
4120    SCROLL #6,-8
4130    porte_grafo 20,29,C%
4140 END SELect
4199 END DEFine vitrine

5000 DEFine FuNction coji_tron(PANT,PULT,OB%)
5010 IF OB%=0 AND RND<PANT THEN RETurn 6
5020 IF OB%=6 AND RND<PULT THEN RETurn 0
5030 RETurn OB%
5099 END DEFine coji_tron

6000 DEFine PROCedure cambie_grafo
6010 SELect ON XF
6020  ON XF=1 TO 7
6030   CH%=2
6040   F%=7
6050   K%=0
6060  ON XF=8
6070   CH%=1
6080   F%=9
6090   K%=-1
6100  ON XF=11
6110   CH%=1
6120   F%=9
6130   K%=1
6140  ON XF=12 TO 15
6150   CH%=2
6160   F%=7
6170   K%=0
6180  ON XF=16 TO 20
6190   CH%=1
6200   F%=9
6210   K%=0
6220  ON XF=21 TO 26
6230   CH%=2
6240   F%=7
6250   K%=0
6260 END SELect
6499 END DEFine cambie_grafo

6500 DEFine PROCedure mueva_rana
6505 IF SEG<10 THEN END DEFine mueva_rana
6510 cambie_grafo
6520 porte_grafo XF,YF-K%,CH%
6530 I%=KEYROW(1)
6540 IF I%=4 AND YF>2 THEN YF=YF-1
6550 IF I%=128 AND YF<31 THEN YF=YF+1
6560 IF I%=16 THEN
```

```
6570  croe_poco
6580  XF=XF+3
6590 END IF
6600 Z%=tinte(YF,XF,4,4)
6610 IF Z%=1 THEN
6620  F%=8
6630  ACABADO=CERTO
6640  porte_grafo XF,YF,F%
6645 ELSE
6650 cambie_grafo
6660 porte_grafo XF,YF,F%
6665 END IF
6670 SELect ON XF
6680  ON XF=17
6690   cace_fuera
6710  ON XF=20
6720   cace_fuera
6721   cace_fuera
6730  ON XF=18 TO 23
6740   ACABADO=CERTO
6750 END SELect
6799 END DEFine mueva_rana

7200 DEFine PROCedure cace_fuera
7210 IF YF=0 THEN
7220  ACABADO=CERTO
7230 ELSE
7250  YF=YF-1
7260 END IF
7299 END DEFine cace_fuera

7500 DEFine PROCedure mate_pase
7510 IF XF>20 THEN
7520  BEEP .1,10
7530  PAUSE 5
7540  BEEP
7550 ELSE
7560  BEEP .1,5
7570  PAUSE 2
7580  BEEP
7590 END IF
7600 PAUSE 50
7699 END DEFine mate_pase

8000 DEFine PROCedure prologue
8010 PAPER 0
8020 INK 4
8030 WINDOW 512,256,0,0
8040 CLS
8050 AT 2,10:PRINT 'R A N A S  U R B A N A S'
8060 AT 5,9: PRINT 'En este juego'\
8070 PRINT TO 11;'tienes que guiar'\
8080 PRINT TO 9;'tres ranas para cruzar'\
```

```
8090 PRINT TO 10;'una carretera de mucho trafico'\
8100 PRINT TO 9;'y un rio caudaloso'\\
8110 PRINT TO 9;'Usa las flechas'\
8120 PRINT TO 11;'arriba, abajo'\
8130 PRINT TO 12;'y derecha'\\
8140 PRINT TO 9;'No las metas debajo de un'\
8150 PRINT TO 10;'coche y que no'\
8160 PRINT TO 9;'caigan en el rio.'\\
8170 PRINT TO 11;'Pulsa cualquier tecla'
8180 PRINT TO 13;'para comenzar'
8190 A$=INKEY$(-1)
8192 AT 22,12
8194 FLASH 1
8196 PRINT 'POR FAVOR ESPERA'
8198 FLASH 0
8199 END DEFine prologue

8500 DEFine PROCedure fine_juego
8510 CLS
8520 REPeat inp
8530 AT 10,3
8540 PRINT 'Otro juego S/N?';
8550 INPUT A$
8560 IF A$(1)='S' OR A$(1)='N' THEN EXIT inp
8570 END REPeat inp
8580 IF A$(1)='S' THEN OTRA=CERTO
8599 END DEFine fine_juego

9500 DEFine FuNction tinte(R%,C%,X%,Y%)
9510 LOCal A%,P%,B%,A
9520 P%=C%*8+X%
9530 A=131072+(512*R%+Y%*64+P% DIV 4)*2
9540 B%=PEEK(A+1)
9550 A%=PEEK(A)
9560 P%=P% MOD 4
9570 IF P%=3 THEN RETurn ((2 && A%)*2)+(3 && B%)
9580 IF P%=2 THEN RETurn ((8 && A%)DIV 2)+
     ((12 && B%) DIV 4)
9590 IF P%=1 THEN RETurn ((32 && A%)DIV 8)+
     ((48 && B%) DIV 16)
9600 IF P%=0 THEN RETurn ((128 && A%) DIV 32)+
     ((192 && B%) DIV 64)
9699 END DEFine tinte

9800 DEFine PROCedure croe_poco
9810 croe_algo
9820 BEEP
9830 PAUSE 2
9840 croe_algo
9850 BEEP
9899 END DEFine croe_poco
```

```
9900 DEFine PROCedure croe_algo
9910 BEEP .1,20,5,0,7,2
9920 PAUSE 6
9999 END DEFine croe_algo
```

Capítulo Cinco

Culebreo

La animación de los objetos alargados es normalmente un problema muy difícil sin importar el lenguaje que uses. Si intentas usar BASIC, el problema radica en intentar alterar un gran número de cuadratínes lo suficientemente rápido como para dar la impresión de un movimiento armonizado, sin espasmos. Por otro lado, si usas lenguaje **ensamblador**, entonces el problema principal estriba en implementar los complicados cálculos que permiten mantener el rastro de todo lo que está expuesto en pantalla. Dicho simplemente, el problema con intentar hacer que se mueva un objeto grande y alargado es que habitualmente implica exponer, borrar y luego volver a exponer demasiados caracteres gráficos, cada vez que se da una ronda del bucle de animación.

Hay sin embargo, un motivo gráfico alargado que puede fácilmente animarse con cierta rapidez y que además tiene una manera complicada y fascinante de moverse -son los **reptiles.** Una culebra, por ejemplo, puede hacer que siga su curso por la pantalla de manera tal que su velocidad no depende en absoluto de lo larga que sea. Si la dirección del movimiento está controlada por las cuatro teclas de flechas, entonces hay algo atractivo sobre cómo conducir ese bicho por toda la pantalla y con el añadido de unas cuantas reglas sencillas -tal como la de no estar permitido que cruce su propio cuerpo formando un lazo- la tarea se convierte inmediatamente en un reto de habilidad. En este capítulo se presenta el método fundamental para la animación de esta clase de motivos gráficos, y quizás se desarrolla el juego que puede ser el más 'adictivo' del libro.

Animación de serpientes y similares

Una culebra consta de una retahíla de símbolos gráficos expuestos en lugares adjuntos formando una sucesión, en forma de **cordón.** Un extremo de ese cordón es el que se denomina **cabeza** y el otro obviamente **cola.** Lo que hay en medio es el **cuerpo.** La dirección en que se mueve una culebra está gobernada en esencia por la dirección del movimiento de la cabeza. Cualquier otro carácter que intervenga en la culebra finalmente seguirá la trayectoria de la cabeza. Por ejemplo, el segundo de los símbolos gráficos de la culebra sigue el curso de la cabeza, con un movimiento por detrás desplazado en el lapso de tiempo siguiente. En otras palabras, cuando la cabeza pasa a una nueva posición, el segundo carácter del motivo culebra ocupa la vieja posición de la cabeza. De la misma manera, el tercer carácter pasa a la vieja posición del segundo, y así se va pasando por todo el cuerpo de la culebra hasta la cola.

Esta descripción de cómo se mueve realmente una culebra es exacta, pero puede inferirse de ello que todos los caracteres componentes del motivo culebra realmente se desplazan cada vez que la cabeza lo hace. Si así fuera el caso, la animación de una culebra de un tamaño concreto se convertiría rápidamente en un tremendo problema para el lenguaje ensamblador. Sin embargo, si todos los caracteres que componen la culebra son el mismo motivo gráfico, sólo dos de ellos -la **cabeza** y la **cola**- son realmente los que necesitan moverse para dar la impresión de que la culebra va reptando por la pantalla. La razón creo que no será difícil de ver.

La cabeza tiene que moverse porque está desplazándose a un cuadratín que previamente estaba en 'blanco' (o más generalmente, simplemente no es uno de la propia culebra). Por otro lado, el segundo carácter pasa a la posición que la cabeza ocupaba en el lapso de tiempo anterior, pero ya que es un carácter del mismo 'estilo' no hay ningún cambio apreciable al efectuar este movimiento. Obviamente, si nos las arreglamos para no clarear la vieja posición de la cabeza cuando pasa a su nueva posición, no hay necesidad de volver a exponer el segundo carácter en su nueva posición. El mismo razonamiento puede aplicarse a todos los caracteres a lo largo de la culebra hasta que lleguemos a la cola. En este caso, el razonamiento todavía se mantiene y no tenemos que mover la cola a la posición ocupada anteriormente por el penúltimo carácter de la culebra, pero además de eso la vieja posición de la cola tiene que clarearse porque no hay ningún otro carácter de la culebra que tenga que pasar a la vieja posición de la cola.

Para resumir, si una culebra está compuesta de caracteres **idénticos,** puede hacerse que se mueva simplemente exponiendo la cabeza en su nueva posición y clareando la vieja posición de la cola.

Para ver una simple demostración de este método, ensaya el siguiente programa que anima un motivo culebra de corta longitud formado por caracteres gráficos "S" a lo ancho de la pantalla en una dirección prefijada.

```
 10 MODE 8:WINDOW 512,256,0,0
 15 CLS
 20 XH=10:YH=15
 30 XT=0:YT=15
 40 AT YH,XH:PRINT "S";
 50 AT YT,XT:PRINT " ";
 60 XH=XH+1
 70 XT=XT+1
 80 IF XH>40 THEN XH=1
 90 IF XT>40 THEN XT=1
100 PAUSE 5
110 GO TO 40
```

La línea 20 refleja la posición de la cabeza en XH,YH (por Head=Cabeza) y la línea 30 asigna la posición de la cola en XT,YT (por Tail=Cola). La línea 40 expone la cabeza en su nueva posición y la línea 50 cancela la vieja posición de la cola. Las líneas 60 a 90 actualizan las posiciones de la cabeza y la cola, y luego, después de una breve **demora** (línea 100), se repite otra ronda del bucle. Observa que lo que hace este sencillo ejemplo es que hay una manera fácil de actualizar las posiciones de la cabeza y la cola porque la culebra se está moviendo **horizontalmente.**

En la práctica es habitual que los caracteres que forman la cabeza y la cola de la serpiente sean diferentes de los que forman el resto del cuerpo, pero incluso esta enmienda provoca muy poco esfuerzo de trabajo adicional. Si la cabeza es diferente del resto del cuerpo, entonces su vieja posición tiene que cambiarse a un carácter que corresponda a los del cuerpo de la serpiente. Si la cola también es diferente, además de clarear su vieja posición, tendríamos que exponer el motivo cola en su nueva posición. Así, incluso una culebra bien parecida con una cabeza y cola diferente sólo necesita exponer cuatro caracteres para dar la impresión de movimiento, y sin importar lo larga que sea.

La serpiente singante - las 'listas de espera'

Aunque la animación de los motivos vermiformes -en forma de gusano- sólo requieren exponer un pequeño número de símbolos en cada ronda del bucle de animación, todavía existe el problema de mantener el rastro de las posiciones ocupadas por todos los caracteres que componen la culebra. Dado que el movimiento reptante sólo implica a la cabeza y a la cola, puede que te admires de por qué se necesita seguir el rastro de **todos** los caracteres que componen el motivo reptil. La razón de eso es la necesidad de saber dónde ha de exponerse la cola en cada movimiento. Cuando la culebra se mueve según una línea recta, como en el ejemplo de la sección anterior, es fácil de saber a dónde debe pasar la cola. Sin embargo, cuando la cabeza cambia su dirección todo resulta diferente. Cada vez que la serpiente se mueve, la cola pasa a ocupar el cuadratín que anteriormente ocupaba el penúltimo carácter de la serpiente, y de ahí que sea patente la necesidad de saber siempre la posición de dicho penúltimo carácter. Este razonamiento puede reiterarse porque cada vez que la serpiente se mueve, el antepenúltimo carácter se convierte en el penúltimo. En otras palabras, si todas las coordenadas del carácter I-ésimo de la serpiente se almacenan en las variables subindicadas X(I) e Y(I), entonces en cada ronda la actualización de las coordenadas será la siguiente:

```
XT=X(1)
YT=Y(1)
```

y luego:

```
FOR I=N TO 2 STEP -1
X(I-1)=X(I)
Y(I-1)=Y(I)
END FIR I
```

y finalmente:

```
X(N)=XH
Y(N)=YH
```

cuando la serpiente tiene una longitud de N cuadratínes, las coordenadas de la cabeza se reflejan en X(N) e Y(N); siendo XT e YT las coordenadas de la vieja posición de la cola, y XH e YH las nuevas coordenadas de la cabeza. Si examinas este bucle, serás capaz de apreciar que desplaza todas las coordenadas una posición; las coordenadas del primer carácter de la serpiente se convierten en las coordenadas del segundo carácter, y así sucesivamente.

Detrás de este desplazamiento, la animación de la culebra se consigue simplemente exponiendo la cabeza en su nueva posición:

```
porte_grafo X(N),Y(N),H%
```

luego cambiando la vieja posición de la cabeza para que sea ocupada por el primer carácter del cuerpo de la serpiente:

```
porte_grafo X(N-1),Y(N-1),S%
```

y finalmente, borrando la vieja cola y luego exponiéndola en su nueva posición:

```
porte_grafo XT,YT,3
porte_grafo X(1),Y(1),T%
```

donde H%, S% y T% son variables enteras que contienen el código del símbolo gráfico usado respectivamente para la cabeza, cuerpo y cola de la serpiente, y el motivo gráfico **3** corresponde a un cuadratín relleno con el color adecuado para borrar la cola.

El único problema con el método anterior es que cada vez que la serpiente se mueve, el contenido de todas las parejas de variables subindicadas X(I) e Y(I) tiene que desplazarse a la anterior. Ese desplazamiento regresivo representa mucho trabajo para que lo realice un programa en BASIC cada vez que se pasa por el bucle de animación y, lo que es peor, la cantidad de trabajo se incrementa en proporción a la longitud de la culebra. Usando este método en BASIC, una culebra se movería tan lentamente y acabaría reptando tan lentamente a medida que aumenta su longitud que virtualmente quedaría quieta.

La solución a esta dificultad ha de encontrarse aprovechándose de la existencia de otro **colectivo de datos** diferente a las tablas y listas habituales, y que está **estructurado** de manera similar a las 'colas de espera' en una ventanilla. Si desearas saber más sobre la teoría en que se basan las estructuras de datos en general y éstas en particular, puedes ver el libro "The Complete Programmer", por Mike James (Granada, 1983). La idea fundamental es la de evitar tener que mover todos los datos que componen las variables múltiples X e Y, usando en lugar de eso un par de **punteros**, uno para señalar hacia las coordenadas de la cabeza y otro hacia las coordenadas de la cola. Por ejemplo, si las coordenadas de cada carácter se guardan de nuevo otra vez en las tablas X e Y, y se elige Q como el **índice** del elemento de esa tabla que refleja las coordenadas de la cabeza y se elige Z como el índice del elemento de la tabla que refleja las coordenadas de la cola, entonces el procedimiento de actualización es simplemente:

```
Q=Q+1
X(Q)=XH
Y(Q)=YH

Z=Z+1
```

En otras palabras, la nueva posición de la cabeza de la culebra se almacena en el elemento situado en la ristra una posición por delante del que reflejaba la vieja posición de la cabeza. De esta manera, cada vez que la cabeza se mueve, va dejando detrás de ella una 'estela' con las coordenadas de sus viejas posiciones. Así por ejemplo, si la cabeza está actualmente en X(Q),Y(Q) entonces su posición previa fue X(Q-1),Y(Q-1), e inmediatamente antes de esa estuvo en X(Q-2),Y(Q-2) y así sucesivamente. Esta estela de coordenadas puede usarse para hacer que la cola de la culebra siga el curso de la cabeza alrededor de la pantalla simplemente haciendo que se mueva desde X(Z),Y(Z) hasta X(Z+1),Y(Z+1) cada vez que se da una pasada por el bucle de animación.

Técnicamente esa **estela** de coordenadas y las variables **puntero** Q y Z son tratadas de manera similar a lo que ocurre en una **lista de espera.** Para comprender cómo operan las coordenadas puede ser conveniente recurrir al símil de una **procesión** que va desplazándose sobre los elementos de la tabla de coordenadas, de manera que las coordenadas de la nueva posición de la cola de la culebra constituyen la parte frontal de dicha procesión. A medida que la culebra avanza, las coordenadas de la cabeza se desplazan a los elementos superiores de la tabla, y llegará en un momento dado a que la posición inicial de la cabeza esté ahora ocupada por la posición corriente de la cola (véase Fig. 5.1).

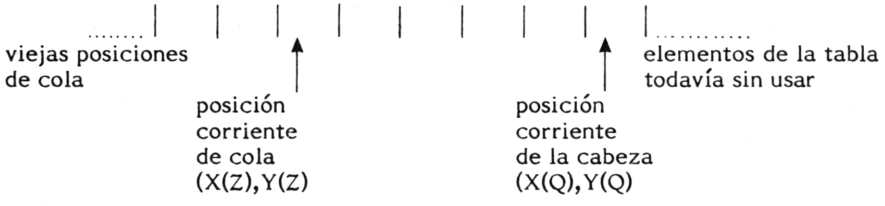


Fig. 5.1.

Este método de conservar las coordenadas de la cabeza de la culebra presenta un gran problema: a medida que la serpiente se mueve por la pantalla, se ocupan más y más elementos de la tabla de coordenadas. Sin embargo, en cualquier momento dado, sólo los elementos comprendidos entre el X(Z),Y(Z) y X(Q),Y(Q) son los necesarios. El resto de los elementos de la tabla o bien está todavía sin usar, o bien contienen viejas posiciones de la cola que ya no sirven para nada. La solución es simplemente hacer la tabla lo suficientemente grande como para que puedan reflejarse todas las coordenadas de la culebra más larga que vayamos a animar, y si uno u otro de los punteros Z o Q alcanza uno o otro límite de la tabla, entonces volver a colocar esos punteros al valor 1. La mejor manera de imaginar la forma en que esto trabaja es pensar como que es una tabla **circular** con su último elemento adosado a su primer elemento. En este sentido, los punteros de cabeza Q y de cola Z se moverían en un círculo y con todo el resto de coordenadas que definen las posiciones de la culebra reflejados entre esos dos elementos. Con esta pequeña adición tenemos ahora todas las ideas necesarias para implementar una variedad de juegos basados en objetos alargados como culebras y similares.

El diseño del juego

Hay muchas posibilidades diferentes para usar culebras en movimiento como parte de un juego, pero para el juego presentado en este capítulo la sencillez es el objetivo principal. La idea fundamental es que el jugador tiene que guiar una culebra en movimiento continuo por la pantalla con la meta de comerse tantas ranas como sea posible. A medida que come una rana, la serpiente incrementa su longitud en un carácter y así se hace un poco más difícil y más excitante el juego. Para aumentar la dificultad del juego todavía más, la serpiente no solamente debe evitar cruzarse sobre sí misma, sino también debe evitar los renacuajos venenosos que se esconden entre las ranas. La Fig. 5.2 muestra una imagen típica durante el juego. Otros detalles del juego se presentarán a medida que se describe el programa.

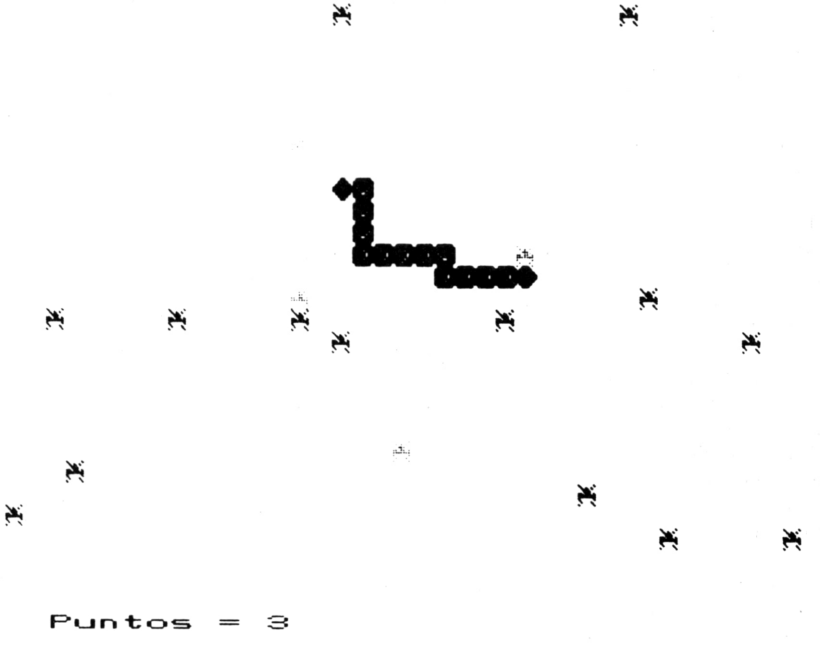


Fig. 5.2.

El programa principal

Se usan cuatro colores en este juego. Hemos elegido amarillo para el fondo, negro y amarillo para la culebra, verde para las ranas y rojo para los renacuajos venenosos. También aquí se define una ventana de 512 por 256 para dar suficiente espacio de maniobra a una culebra bastante larga.

El programa principal para este juego sigue las pautas usuales:

```
 10  REMark EXHIBICIONE
 20  WINDOW 512,256,0,0
 30  prepa_juego
 40  prologue
 50  REPeat manga
 60   prepa_manga
 70   trace_escena
 80   REPeat movida
 90    mueva_paso
100    IF NO_COMIDA OR NO_VIDAS THEN EXIT movida
110   END REPeat movida
120   fine_manga
130   IF NOT OTRA THEN EXIT manga
140 END REPeat manga
199 STOP
```

La línea 30 cita **prepa_juego** que implanta los motivos gráficos a usar y hace que la variable OTRA sea 0. Esta variable se usa para indicar si el programa está funcionando por primera vez o si es una manga sucesiva la que se va a jugar. La línea 130 comprueba OTRA y sale del bucle si no se ha solicitado jugar de nuevo. La línea 40 apela al procedimiento **prologue** para mostrar los títulos y reglas del juego. Las líneas 50 a 140 forman el bucle responsable principal del juego. La línea 60 recurre al procedimiento **prepa_manga** para establecer las condiciones iniciales de cada manga del juego y luego la línea 70 **trace_escena** presenta las ranas y renacuajos y la culebra en su posición inicial.

Las línea 80 a 110 constituyen el bucle de animación de los motivos gráficos. El procedimiento **mueva_paso** es responsable de desplazar una posición la culebra y comprobar si se ha comido algo. La línea 100 examina las condiciones para finalizar el juego. La variable NO_COMIDA es verdad si la culebra se las ha arreglado para comerse todas las ranas sin haber agotado sus cinco vidas, y la variable NO_VIDAS tiene el valor cierto si la culebra o bien se ha cruzado consigo misma o se ha tropezado con cinco renacuajos venenosos y por tanto ha agotado las vidas disponibles. Finalmente, el procedimiento **fine_manga** finaliza la jugada corriente con un mensaje adecuado y, si procede, permite al jugador que teclee su nombre en una tabla de 'mejores puntuaciones'.

Procedimientos prepa_juego y prepa_manga

Los caracteres definidos por el usuario que se emplean para formar el motivo gráfico culebra, son un poquito más complicados de lo que se pudiera esperar. Además de cuatro cabezas diferentes -símbolos gráficos 0, 3, 7 y 9 (uno para cada dirección de movimiento)- hay también cuatro diferentes 'anillos del cuerpo' que son los símbolos gráficos 1, 4, 8 y 10 (véase Fig. 5.3).

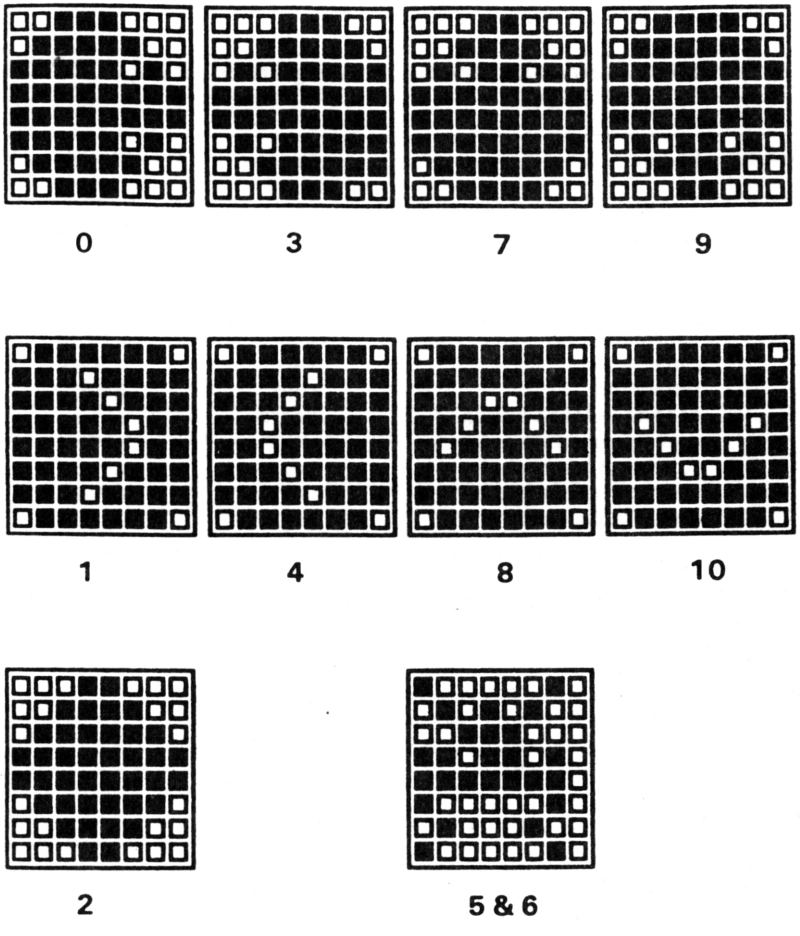


Fig. 5.3. Símbolos gráficos para la cabeza, los anillos del cuerpo y la cola de la culebra, y también para la rana

Eligiendo un símbolo para los anillos del cuerpo que señale en la misma dirección que la cabeza de la serpiente, de manera que se exponga encima de la cabeza en cada paso que se dé, la culebra puede hacerse que parezca incluso más immpresionante y llena de movimiento. Hay sólo un símbolo (2) usado para la cola, porque esa forma de rombo o diamante parece la misma cualquiera que sea la dirección del movimiento. La rana, que tiene como color de frente el verde, está definida como símbolo gráfico 5 y el renacuajo venenoso, que es de la misma forma pero con color frontal rojo, se define como símbolo gráfico 6. El símbolo 1 es un cuadratín de fondo amarillo que se usa para clarear la vieja posición de la cola de la serpiente.

La Tabla 5.1 resume los símbolos gráficos empleados:

Tabla 5.1. Símbolos para los motivos gráficos del juego

Número	Forma	Frente	Fondo
0	cabeza culebra derecha	negro	amarillo
1	cuerpo culebra derecha	negro	amarillo
2	cola culebra	negro	amarillo
3	cabeza culebra izquierda	negro	amarillo
4	cuerpo culebra izquierda	negro	amarillo
5	rana	verde	amarillo
6	renacuajo	rojo	amarillo
7	cabeza culebra arriba	negro	amarillo
8	cuerpo culebra arriba	negro	amarillo
9	cabeza culebra abajo	negro	amarillo
10	cuerpo culebra abajo	negro	amarillo
11	espacio clareador	amarillo	amarillo

El procedimiento **prepa_juego** sigue un esquema que ya debiera serte familiar:

```
350 DEFine PROCedure prepa_juego
360 DATA -5245,-10109,-7806,-11646
370 DATA -6783,1665,2,0,8769,8316
380 DATA 0,0,9264,18432,8834,-11524,128
390 DATA 22664,-20228,32,26350,28672,20085
400 dire_rus=RESPR(100)
410 RESTORE 360
420 FOR I=dire_rus TO dire_rus+22*2  STEP 2
430 READ B
440 POKE_W I,B
450 END FOR I
510 C_TAB=RESPR(32*12)
520 prepa_grafo 0,0,6,56,124,250,255,255,250,124,56
530 prepa_grafo 1,0,6,126,239,247,251,251,247,239,126
540 prepa_grafo 2,0,6,24,60,126,255,255,126,60,24
550 prepa_grafo 3,0,6,28,62,95,255,255,95,62,28
560 prepa_grafo 4,0,6,126,247,239,223,223,239,247,126
570 prepa_grafo 5,4,6,130,84,56,60,254,130,68,130
580 prepa_grafo 6,2,6,130,84,56,60,254,130,68,130
590 prepa_grafo 7,0,6,24,60,90,255,255,255,126,60
600 prepa_grafo 8,0,6,126,255,231,219,189,255,255,126
610 prepa_grafo 9,0,6,60,126,255,255,126,90,60,24
620 prepa_grafo 10,0,6,126,255,255,189,219,231,255,126
630 prepa_grafo 11,6,6,255,255,255,255,255,255,255,255
640 CERTO=1
650 FALSO=0
660 OTRA=CERTO
670 DIM X%(50)
```

```
680 DIM Y%(50)
690 MAX%=50
700 DIM T$(5,15)
710 DIM T(5)
720 FOR I=1 TO 5
730  T(I)=0
740 END FOR I
799 END DEFine prepa_juego
1000 DEFine PROCedure prepa_manga
1010 NO_COMIDA=FALSO
1020 NO_VIDAS=FALSO
1030 GOLPE=FALSO
1040 C%=0
1050 STOTAL%=0
1060 SC=0
1070 SN=0
1080 COMIDA=0
1099 END DEFine prepa_manga

1300 DEFine PROCedure prepa_grafo(S%,F%,B%,
     R0%,R1%,R2%,R3%,R4%,R5%,R6%,R7%)
1310 LOCal A,FH%,FL%,BH%,BL%
1320 A=C_TAB+32*S%
1330 FH%=(F% DIV 4)*2
1340 FL%=F% && 3
1350 BH%=(B% DIV 4)*2
1360 BL%=B% && 3
1370 prepa_raya(R0%)
1380 prepa_raya(R1%)
1390 prepa_raya(R2%)
1400 prepa_raya(R3%)
1410 prepa_raya(R4%)
1420 prepa_raya(R5%)
1430 prepa_raya(R6%)
1440 prepa_raya(R7%)
1499 END DEFine prepa_grafo
```

La primera parte de **prepa_juego** (líneas 360 a 450) sirve para implantar en memoria el código máquina usado por **porte_grafo** tal y como se describió en el Capítulo Uno. Los símbolos definidos por el usuario se describen usando el código máquina presentado en capítulos previos. Las líneas 510 a 630 corresponden a los doce símbolos gráficos ilustrados en la Fig. 5.3. Las líneas 640 y 650 prefijan los valores de CERTO y FALSO y la 660 hace que el valor inicial de OTRA sea CERTO -lo que significa que el juego funcionará como mínimo una vez, tal como se explicó en la sección previa. Las líneas 670 y 680 **ocupan** el espacio necesario para las dos tablas X% e Y% que serán usadas posteriormente en el programa para conservar las coordenadas de la culebra. MAX% (línea 690) se emplea para hacer que el resto del programa sepa que las tablas tienen 50 elementos, que es por tanto la limitación de la máxima longitud de una culebra. Las tablas T$ y T, cuyo espacio es **ocupado** por las líneas 700 y 710, se usan posteriormente en el programa para conservar las máximas puntuaciones alcanzadas, y las líneas 720 a 740 fijan los valores iniciales de los elementos de la tabla de puntos a 0.

El procedimiento **prepa_manga** fija los indicadores de fin de juego a FALSO (líneas 1010 a 1030) y **arredra** (pone a 0) los contadores usados en el programa. La variable GOLPE se hace cierta si la serpiente ha tropezado con algo y falsa en caso contrario. Las variables C% y STOTAL% se usan para reflejar los puntos a medida que el juego progresa. Cada jugador dispone de cinco manadas de ranas para comerse y cinco vidas para lograrlo. La variable C% registra el número de ranas comidas de la manada corriente en pantalla y STOTAL% registra el número de ranas comidas acumulando en las manadas anteriores. La variable SC es el número de orden de la manada corriente de ranas y SN es el número de la vida corriente de la culebra. La variable COMIDA se usa para llevar la cuenta del número de 'batracios' que quedan por comer.

El decorado inicial

El procedimiento **prologue** es bastante simple aparte de la utilización de los símbolos gráficos de la culebra para escribir la palabra 'SNAKE' en grandes letras a lo ancho de la pantalla para que practiques con tu inglés (i.e. es 'serpiente' y puedes cambiar el procedimiento para que salga la palabra española).

```
8000 DEFine PROCedure prologue
8010 PAPER 6
8020 CLS
8030 INK 0
8040 exhibicione
8045 CLS
8050 AT 4,2
8060 PRINT 'Debes guiar a tu culebra usando'
8065 PRINT '  las cuatro teclas de flechas y'
8070 PRINT '  comerte todas las ranas verdes';
8080 porte_grafo 19,8,5
8085 PRINT \\
8090 PRINT '  Si intentas comerte un sapo'
8100 porte_grafo 20,10,6
8110 PRINT '  o a ti mismo'
8120 PRINT '  entonces perderas una de'
8130 PRINT '  tus cinco vidas'\\
8140 PRINT '  tienes cinco manadas'
8145 PRINT '  de ranas para comerte antes de'
8150 PRINT '  convertirte en una SUPER CULEBRA'
8155 REPeat espera
8156  AT 23,7
8160  FLASH 1
8170  PRINT 'PULSA CUALQUIER TECLA'
8180  FLASH 0
8190  IF INKEY$<>'' THEN EXIT espera
8195 END REPeat espera
8199 END DEFine prologue
```

Las líneas 8080 y 8100 llevan a pantalla ejemplos de una rana y de un renacuajo para que el jugador aprecie la diferencia. La labor real de exhibir la palabra 'SNAKE' se hace mediante el procedimiento **exhibicione:**

```
8500 DEFine PROCedure exhibicione
8510 DATA 8,1,1,0,11,8,1,1,1,11,8,1,1,1,11
8520 DATA 2,11,8,0,11,8,1,1,0
8530 DATA 8,11,11,11,11,8,11,11,10,11,8,11,11,10
8540 DATA 11,10,8,1,11,11,8,11,11,11
8545 DATA 4,4,4,8,11,8,11,11,10,11,8,1,1
8550 DATA 10,11,10,8,11,11,11,4,4,8,11
8560 DATA 11,11,11,8,11,8,11,11,10,11,8,10,1,1
8570 DATA 11,10,4,8,11,11,8,1,1,11
8580 DATA 11,11,11,8,11,8,11,11,10,11,8,11,11
8590 DATA 10,11,10,11,4,8,11,8,11,11,11
8600 DATA 2,1,1,1,11,2,11,11,9,11,2,11,11,9
8610 DATA 11,9,11,11,2,11,4,4,4,2
8620 RESTORE 8500
8630 FOR I=9 TO 14
8640  FOR J=3 TO 26
8650   READ S
8660   porte_grafo J,I,S
8670  END FOR J
8680 END FOR I
8690 PAUSE 100
8699 END DEFine exhibicione
```

Las instrucciones DATA en las líneas 8510 a 8610 son simplemente **listas** que indican los caracteres que han de tomarse (línea 8650) sucesivamente como una lista para ser expuestos en pantalla. Las instrucciones DATA están organizadas en parejas y cada par define una única fila de 23 columnas. Seis filas de estos caracteres de tamaño standard se usan para construir la palabra de cinco letras SNAKE en gran tamaño. Hay otras maneras de generar letras de gran tamaño automáticamente, pero ésta es realmente la única manera conveniente de crearlas usando una gama de caracteres diferentes al tamaño standard.

Procedimientos trace_escena, trace_culebra y trace_comida

Aunque no es obvio en esta etapa del diseño del programa, **trace_escena** es de hecho, citado por otros procedimientos dentro del programa para volver a dibujar la escena después de que la serpiente haya tropezado con un renacuajo venenoso o consigo misma. En este sentido, no sólo es parte de la fase inicial del juego, sino también parte del bucle de animación. La única modificación que es necesaria para que este procedimiento pueda ser aprovechado para ambos propósitos es que debe exponer un cierto número de ranas dado por el valor alojado en la variable COMIDA, pero si la COMIDA es cero debe primero extraer un valor aleatorio, de manera que pueda generar una nueva manada de batracios comestibles.

```
2000 DEFine PROCedure trace_escena
2010 CLS
2020 trace_culebra
2030 trace_comida 6,4*SC
2040 IF COMIDA=0 THEN COMIDA=RND(1 TO 10)+15
2050 trace_comida 5,COMIDA
2060 AT 22,3
2070 PRINT "Puntos ";STOTAL%+C%
2099 END DEFine trace_escena
```

Recurriendo a **trace_culebra** se consigue una serpiente de longitud prefijada y en la misma posición inicial en cada ronda. Este procedimiento también es responsable de establecer los valores iniciales en las tablas de coordenadas X% e Y% y en los **punteros** Q% y Z%. El procedimiento **trace_comida** producirá tanto ranas como renacuajos dependiendo del valor del parámetro F que se le pase desde el procedimiento **trace_escena.** La línea 2030 lo cita para exponer renacuajos y la línea 2050 lo usa para exponer ranas. El número de renacuajos se incrementa a medida que se van comiendo manadas de ranas, haciendo así que el juego sea progresivamente más difícil. El número de ranas está dado por el valor de COMIDA que decrece a partir de su valor inicial a medida que se van comiendo ranas. Si llega al valor cero, la línea 2040 genera un número aleatorio de ranas para llenar otra pantalla. Finalmente, la línea 2070 expone la puntuación total alcanzada en cada momento.

El procedimiento **trace_culebra** parece un poco complicado porque no solamente dibuja la culebra, sino que también establece los valores iniciales de las tablas de coordenadas X% e Y%:

```
2600 DEFine PROCedure trace_culebra
2610 A%=2
2620 B%=10
2630 XH%=A%
2640 YH%=B%
2650 H%=0
2660 S%=1
2670 T%=2
2680 porte_grafo A%,B%,T%
2690 X%(1)=A%
2700 Y%(1)=B%
2710 FOR Q=2 TO 10
2730  A%=A%+1
2740  porte_grafo A%,B%,S%
2750  X%(Q)=A%
2760  Y%(Q)=B%
2770 END FOR Q
2780 A%=A%+1
2790 porte_grafo A%,B%,H%
2800 X%(11)=A%
2810 Y%(11)=B%
2820 XV%=1
2830 YV%=0
2840 Z%=1
2850 Q%=11
2899 END DEFine trace_culebra
```

Las variables A%,B% y XH%,YH% se usan para conservar la posición corriente de la cabeza (Head) de la culebra. La razón de usar dos parejas de variables sólo se hará patente cuando analicemos el procedimiento **actualice**, dado ulteriormente. H% y S% se usan para registrar los códigos de los símbolos gráficos de la cabeza corriente de la culebra y de los anillos del cuerpo, y T% corresponde a la cola (Tail). La línea 2680 expone el símbolo cola en la posición 2,10 y luego las líneas 2690 y 2700 almacenan esas coordenadas en los elementos X%(1) e Y%(1) de la tabla de coordenadas. El bucle FOR, líneas 2710 a 2770, expone los símbolos que forman el cuerpo de la culebra y almacena sus coordenadas en los correspondientes elementos de las tablas X% e Y%. Finalmente, las líneas 2780 y 2790 exponen el símbolo de la cabeza y luego las línea 2800 a 2850 colocan los valores iniciales en las variables que controlan la longitud de la culebra yel rumbo inicial del movimiento. La variable C% señala a las coordenadas corrientes de la cola en las tablas X% e Y%; y Q% apunta hacia las coordenadas corrientes de la cabeza en esas mismas tablas. Las variables XV% e YV% controlan la dirección o rumbo del movimiento y se explican con más detalle como parte de los procedimientos **mire_rumbo** y **actualice.**

El procedimiento **trace_comida** es muy sencillo:

```
2500 DEFine PROCedure trace_comida (F,MONTO)
2510 IF MONTO=0 THEN END DEFine trace_comida
2520 FOR K=1 TO MONTO
2525  REPeat pos
2530   U%=RND(1 TO 30)
2540   V%=RND(1 TO 21)
2550   IF tinte(V%,U%,1,1)=6 THEN EXIT pos
2560  END REPeat pos
2570  porte_grafo U%,V%,F
2580 END FOR K
2599 END DEFine trace_comida
```

El bucle preconfinado, líneas 2520 a 2580 expondrá una manada de ranas o renacuajos en posiciones aleatorias de la pantalla, que tenga tantos 'batracios' como indique la variable MONTO. Que el procedimiento produzca ranas o renacuajos depende del valor de F. El bucle de la línea 2525 hasta la línea 2560 genera números aleatorios que han de ser usados como posibles coordenadas del batracio. Si no hay ningún símbolo en V%,U%, entonces la línea 2550 hace que se salga del bucle. En los demás casos, se generan otros dos números aleatorios. La comprobación de si existe un símbolo en una posición dada se efectúa aplicando la función **tinte** que da como resultado el color de la mota situada en la esquina superior izquierda del cuadratín colocado en V%,U%. (Esta función **tinte** se describió en el Capítulo Uno y el listado se presentó entonces).

Procedimiento mueva_paso y los procedimientos que cita

El procedimiento **mueva_paso** simplemente apela a otros dos procedimientos para que hagan todo el trabajo necesario para desplazar a la culebra.

```
3000 DEFine PROCedure mueva_paso
3010 mire_rumbo
3020 actualice
3099 END DEFine mueva_paso
```

El procedimiento **mire_rumbo** inspecciona el teclado para ver si se ha pulsado una tecla de flecha. Si realmente **está** pulsada, actualiza el valor de las variables que indica la dirección del movimiento de la culebra. El movimiento real de la culebra se efectúa mediante el procedimiento **actualice.** Este procedimiento no sólo da valores actuales a las tablas de coordenadas y expone los símbolos necesarios, sino también comprueba si la culebra ha tropezado con algún batracio.

El procedimiento **mire_rumbo** es muy similar a las otras **rutinas** usadas para inspeccionar el teclado en programas anteriores.

```
4000 DEFine PROCedure mire_rumbo
4002 BEEP .1,-20
4004 PAUSE 2
4006 BEEP
4010 A=KEYROW(1)
4020 IF A=0 THEN END DEFine mire_rumbo
4030 SELect ON A
4040  ON A=4
4050   XV%=0
4060   YV%=-1
4070   H%=7
4080   S%=8
4090  ON A=128
4100   XV%=0
4110   YV%=1
4120   H%=9
4130   S%=10
4140  ON A=2
4150   XV%=-1
4160   YV%=0
4170   H%=3
4180   S%=4
4190  ON A=16
4200   XV%=1
4210   YV%=0
4220   H%=0
4230   S%=1
4240 END SELect
4299 END DEFine mire_rumbo
```

Las líneas 4002 a 4006 hacen una suerte de ruido sibilante usando el comando BEEP. La función KEYROW se usa luego para inspeccionar cada **fila de teclas** sucesivamente (líneas 4010 a 4240). Si una flecha concreta de tecla está pulsada, entonces XV% e YV% se fijan a valores que corresponden a una velocidad en esa dirección.

Por ejemplo, si está pulsada la tecla de flecha a derechas, XV% se pone a 1 e YV% se pone a 0 y cuando XV% e YV% se suman a A% y B% (la posición corriente de la cabeza) obviamente se produce un movimiento hacia la derecha. Además de fijar la velocidad como respuesta a las pulsaciones de las teclas de flecha, también es necesario elegir el símbolo apropiado para usar como cabeza de culebra y como cuerpo de culebra. Por ejemplo, si se ha pulsado la tecla de flecha a derecha, la línea 4220 hace corresponder a H% una cabeza de culebra mirando hacia la derecha, y la línea 4230 hace corresponder a S% un símbolo de cuerpo de culebra apuntando también hacia la derecha.

El procedimiento **actualice** es probablemente el procedimiento más complicado en todo el programa en cuestión:

```
4500 DEFine PROCedure actualice
4510 XH%=A%:YH%=B%
4520 A%=A%+XV%:B%=B%+YV%
4530 IF A%>30 THEN A%=1
4540 IF A%<1 THEN A%=30
4550 IF B%>25 THEN B%=0
4560 IF B%<0 THEN B%=25
4570 IF tinte(B%,A%,4,4)<>6 THEN golpalgo
4580 porte_grafo XH%,YH%,S%
4590 porte_grafo A%,B%,H%
4600 Q%=Q%+1
4610 IF Q%>MAX% THEN Q%=1
4620 X%(Q%)=A%:Y%(Q%)=B%
4630 porte_grafo X%(Z%),Y%(Z%),11
4640 Z%=Z%+1
4650 IF Z%>MAX% THEN Z%=1
4660 porte_grafo X%(Z%),Y%(Z%),T%
4699 END DEFine actualice
```

La línea 4510 preserva la posición corriente de la cabeza dada por A%,B% en XH%,YH%. Luego la línea 4520 actualiza la posición de la cabeza al añadirle las velocidades XV% e YV%. Detrás de eso, las líneas 4530 a 4560 se aseguran que la nueva posición todavía cae dentro de la pantalla. Hay dos maneras de tratar con esta situación cuando la culebra se va a salir por uno de los bordes de la pantalla. Lo podríamos tratar como una equivocación y hacer que perdiera una de sus vidas, o podríamos tratarla como si la pantalla tuviera sus bordes empalmados extrañamente, de manera que si por ejemplo se fuera a salir por la parte superior de la pantalla reaparecería por la parte inferior. Eso es lo que se denomina a menudo como una cancha de juego de la clase 'universo esférico' y es la estrategia usada en este juego.

La línea 4570 aplica la función **tinte** para comprobar si la culebra ha golpeado con algo. Si así ha sido, se recurre a **golpalgo** para manejar esta situación. En caso contrario, la línea 4580 expone un símbolo del cuerpo en la vieja posición de la cabeza con lo que se borra ésta y la línea 4590 expone la cabeza en su nueva posición. Observa que el procedimiento **actualice** no tiene que preocuparse sobre cuál de los muchos símbolos de cabeza o de cuerpo ha de usar, porque los correctos ya han sido elegidos mediante el procedimiento **mire_rumbo.** A continuación de esto, la nueva posición de la cabeza queda reflejada en las tablas de coordenadas X% e Y% (línea 4620) después de que se haya actualizado el **puntero de cabeza** Q% en las líneas 4600 y 4610.

La tarea final del procedimiento **actualice** es mover la cola de la serpiente. La línea 4630 borra la cola en su vieja posición. La línea 4640 y 4650 actualizan luego el **puntero de cola** Z%, y la línea 4660 expone el símbolo de la cola en su nueva posición.

Procedimiento golpalgo y los procedimientos que cita

Este procedimiento es responsable de llevar a cabo las acciones requeridas siempre que la culebra tropieza con algún motivo gráfico:

```
5000 DEFine PROCedure golpalgo
5010 COL=tinte(B%,A%,4,4)
5020 SELect ON COL
5030  ON COL=4
5040   crece_lebra
5050  ON COL=REMAINDER
5060   muere_lebra
5070  END SELect
5099 END DEFine golpalgo
```

Cuando se recurre a **golpalgo,** todo lo que se sabe es que la serpiente ha tropezado con algo y la primera cosa que este procedimiento tiene que hacer es descubrir con qué. La línea 5010 aplica la función **tinte** para descubrir el color de la mota situada hacia la mitad del carácter que la culebra está a punto de ocupar. Si es rojo o negro indica que la culebra ha tropezado con un renacuajo o consigo misma, respectivamente, y se apela al procedimiento **muere_lebra** (línea 5060) para quitar una de las vidas disponibles. Sin embargo, si el color resulta ser el verde, se recurre al procedimiento **crece_lebra** para añadir 1 a la puntuación del jugador y hacer que la culebra incremente su longitud en un símbolo (línea 5040).

El procedimiento **muere_lebra** es bastante corto, pero la manera en que trabaja e interactúa en el resto del programa es bastante complicada.

```
5500 DEFine PROCedure muere_lebra
5510 BEEP .1,-10
5512 PAUSE 2
5515 BEEP
5520 BEEP .1,-20
5524 PAUSE 2
5526 BEEP
5530 SN=SN+1
5540 IF SN<4 THEN
5560  trace_escena
5570 ELSE
5580  STOTAL%=STOTAL%+C%
5590  NO_VIDAS=CERTO
5600 END IF
5699 END DEFine muere_lebra
```

Las líneas 5510 a 5526 emiten un ruido para que el jugador sepa que ha perdido una de las vidas. Luego, la línea 5530 añade 1 al contador de culebras SN. Si este contador marca más de 4, es que todas las vidas ya se han empleado y el juego está terminado. La línea 5540 comprueba eso y consecuentemente fija NO_VIDAS al valor CERTO para informar al programa principal que eso ha ocurrido. En los demás casos, se cita **trace_escena** para volver a dibujar la culebra y las ranas y renacuajos que quedan, preparándose para que continúe el juego.

El procedimiento **crece_lebra** tiene dos tareas que llevar a cabo - preocuparse de la puntuación y hacer que la culebra aumente su longitud en un símbolo.

```
5700 DEFine PROCedure crece_lebra
5710 BEEP .1,10
5715 PAUSE 2
5720 BEEP
5730 C%=C%+1
5740 COMIDA=COMIDA-1
5750 AT 22,3
5760 PRINT 'Puntos';STOTAL%+C%
5770 Z%=Z%-1
5780 IF Z%<1 THEN Z%=MAX%
5790 IF COMIDA<>0 THEN END DEFine crece_lebra
5800 STOTAL%=STOTAL%+C%
5810 C%=0
5820 IF SC<4 THEN
5830  SC=SC+1
5840  trace_escena
5850 ELSE
5860  NO_COMIDA=CERTO
5870 END IF
5899 END DEFine crece_lebra
```

Las líneas 5710 a 5720 emiten el ruido que indica al jugador que se acaba de tragar una rana. Luego, la línea 5730 añade 1 a la puntuación corriente y la línea 5740 resta 1 de la COMIDA disponible. La razón de por qué ha de reducirse en 1 es que esta variable se usa para comprobar si la culebra se ha comido ya todas las ranas, y también como un indicador de cuántas ranas hay que volver a trazar cuando la culebra pierde una de sus vidas y el juego se reanuda. Las líneas 5750 y 5760 exponen los puntos alcanzados.

Aumentar la longitud de la culebra en un símbolo es la pura sencillez. Las líneas 5770 y 5780 ajustan el puntero de cola Z%, haciendo que se mueva la posición corriente de la cola una posición hacia atrás. Cuando termina **crece_lebra** el control se devuelve al procedimiento **actualice** que es el que efectúa la tarea de hacer avanzar la culebra. A causa de eso, el efecto de ajustar Z% es dejar la cola en su corriente posición durante un paso, incluso aunque la cabeza avance. De esta manera la longitud de la culebra se incrementa automáticamente amedida que se desplaza.

La parte final del procedimiento concierne a la detección de si se han comido ya todas las ranas. En ese caso, la línea 5800 añade los puntos obtenidos al total acumulado. La línea 5880 comprueba luego si ya se han comido todas las cinco manadas de ranas disponibles y la línea 5860 coloca NO_COMIDA al valor CERTO si así ha sucedido.

Si todavía quedan más manadas de ranas por comerse, se incrementa el 'número de escena' SC, mediante la línea 5830, y se cita el procedimiento **trace_escena** (línea 5840) para sacar una nueva manada de batracios. Observa que **trace_escena** detectará el hecho de ser cero el valor de COMIDA y por tanto generará un nuevo número aleatorio para la cantidad de ranas a sacar en esta escena.

Procedimiento fine_manga

La finalización del juego es poco espectacular, ya que simplemente expone un mensaje adecuado a la situación diciéndote lo bien que has jugado, basándose en el número de manadas completas de ranas que la culebra se haya comido antes de perder las cinco vidas que tiene disponibles.

```
7000 DEFine PROCedure fine_manga
7010 CLS
7015 AT 3,10
7020 SELect ON SC
7030  ON SC=0
7040   PRINT 'Degradado a simple gusano'
7050  ON SC=1
7060   PRINT 'Las ranas no tienen que preocuparse'
7070  ON SC=2
7080   PRINT 'No esta mal para una culebra de cesped'
7090  ON SC=3
7100   PRINT 'Bien aprovechado el paseo'
7110  ON SC=4
7140   IF NO_COMIDA THEN
7150    expo_super
7160    exhibicione
7170   ELSE
7180    PRINT 'Un paseo virulento y maligno'
7190   END IF
7200 END SELect
7210 FOR I= 5 TO 1 STEP -1
7220  IF STOTAL%>=T(I) THEN N=I
7230 END FOR I
7235 AT 15,10
7240 IF N=0 THEN
7250  PRINT 'Tus puntos';STOTAL%
7260 ELSE
7270  PRINT 'Ocupas ahora el rango';N
7280  PRINT TO 10;'en la liga culebrera'
7290 IF N<5 THEN
7300  FOR I=5 TO N+1 STEP -1
7310   T(I)=T(I-1)
7320   T$(I)=T$(I-1)
7330  END FOR I
7340 END IF
```

```
7350 T(N)=STOTAL%
7355 PRINT TO 10;
7360 INPUT 'Dime tu nombre';T$(N)
7370 CLS
7380 FOR I=1 TO 5
7390  AT I*2+5,5
7400  PRINT T$(I,1 TO 15);
7410  PRINT TO 21; T(I)
7420 END FOR I
7440 REPeat otro_juego
7445  AT 21,10
7450  INPUT 'Otro paseo';A$
7455  IF A$='S' OR A$='N' THEN EXIT otro_juego
7460  END REPeat otro_juego
7470 IF A$='N' THEN
7480  OTRA=FALSO
7490 ELSE
7500  OTRA=CERTO
7510 END IF
7599 END DEFine fine_manga
```

Las líneas 7015 a 7200 exponen los mensajes de felicitación. Si has logrado comerte las cinco manadas de ranas presentadas, entonces se generará el mensaje 'Super Snake' en grandes letras hecha con los símbolos del motivo gráfico culebra al igual que al principio del juego. Eso se lleva a cabo recurriendo a los procedimientos **expo_super** y **exhibicione.** El procedimiento **expo_super** es muy similar al que ya vímos. Para sacar la palabra 'Snake' en mayúsculas (y que suponemos ya habrás cambiado).

```
6000 DEFine PROCedure expo_super
6010 RESTORE 6010
6020 DATA 8,1,1,0,11,2,11,11,7,11,8,1,1,1
6030 DATA 11,8,1,1,0,11,8,1,1,1
6040 DATA 8,11,11,11,11,10,11,11,8,11,8,11,11,10
6050 DATA 11,8,11,11,11,11,8,11,11,10
6060 DATA 4,4,4,8,11,10,11,11,8,11,8,3,4,10
6070 DATA 11,8,4,8,11,11,8,4,4,10
6080 DATA 11,11,11,8,11,10,11,11,8,11,8,11,11,11
6090 DATA 11,8,1,1,11,11,8,10,1,11
6092 DATA 11,11,11,8,11,10,11,11,8,11,8,11,11,11
6094 DATA 11,8,11,11,11,11,8,11,10,1
6100 DATA 2,1,1,1,11,10,1,1,1,11,2,11,11,11,11
6110 DATA 4,4,4,2,11,2,11,11,9
6120 PAUSE 20
6130 CLS
6140 FOR I=2 TO 7
6150  FOR J=3 TO 26
6160   READ S%
6170   porte_grafo J,I,S%
6180  END FOR J
6190 END FOR I
6199 END DEFine expo_super
```

Una faceta adicional del final de este juego es que mantiene una tabla con las puntuaciones logradas (líneas 7200 a 7240). Si estás dentro de los cinco primeros se te pide que escribas tu nombre para incluirlo en la posición correcta de la tabla. Las líneas 7210 a 7230 determinan dónde encaja tu puntuación dentro de la liga de culebras. Eso funciona buscando la primera puntuación de la tabla a la que tu puntuación supera o iguala. Si mereces ser incluido entre los máximos jugadores, todas las puntuaciones inferiores o iguales a la tuya se desplazan hacia abajo un lugar (líneas 7290 a 7330) y se inserta entonces tu puntuación y nombre (líneas 7350 a 7360). Luego se expone la tabla representativa de la liga y se le pide al jugador si desea jugar otra vez (líneas 7440 a 7460). Observa que si se requiere otro juego no basta simplemente usar RUN para volver a ejecutar ése, ya que así se borrarían los contenidos de las tablas que reflejan las puntuaciones alcanzadas. En lugar de eso, se usa la variable OTRA para indicar al resto del programa que va a ser ejecutado de nuevo.

Conclusión - practicando con el juego

Incluso este sencillo juego que implica una sola culebra es bastante atractivo. Eso no quiere decir que no haya manera de mejorarlo y renovarlo, sino que globalmente el juego es un gran éxito y tiene una velocidad e interés que normalmente sólo se encuentra en programas para juegos escritos en lenguaje ensamblador. En el siguiente capítulo exploraremos el potencial para aprovechar nuestra culebra en otro juego.

La versión final - un listado completo

```
  10  REMark EXHIBICIONE
  20  WINDOW 512,256,0,0
  30  prepa_juego
  40  prologue
  50  REPeat manga
  60   prepa_manga
  70   trace_escena
  80   REPeat movida
  90    mueva_paso
 100   IF NO_COMIDA OR NO_VIDAS THEN EXIT movida
 110  END REPeat movida
 120  fine_manga
 130  IF NOT OTRA THEN EXIT manga
 140 END REPeat manga
 199 STOP

 350 DEFine PROCedure prepa_juego
 360 DATA -5245,-10109,-7806,-11646
 370 DATA -6783,1665,2,0,8769,8316
 380 DATA 0,0,9264,18432,8834,-11524,128
 390 DATA 22664,-20228,32,26350,28672,20085
```

```
400 dire_rus=RESPR(100)
410 RESTORE 360
420 FOR I=dire_rus TO dire_rus+22*2  STEP 2
430 READ B
440 POKE_W I,B
450 END FOR I
510 C_TAB=RESPR(32*12)
520 prepa_grafo 0,0,6,56,124,250,255,255,250,124,56
530 prepa_grafo 1,0,6,126,239,247,251,251,247,239,126
540 prepa_grafo 2,0,6,24,60,126,255,255,126,60,24
550 prepa_grafo 3,0,6,28,62,95,255,255,95,62,28
560 prepa_grafo 4,0,6,126,247,239,223,223,239,247,126
570 prepa_grafo 5,4,6,130,84,56,60,254,130,68,130
580 prepa_grafo 6,2,6,130,84,56,60,254,130,68,130
590 prepa_grafo 7,0,6,24,60,90,255,255,255,126,60
600 prepa_grafo 8,0,6,60,255,231,219,189,255,255,126
610 prepa_grafo 9,0,6,60,126,255,255,126,90,60,24
620 prepa_grafo 10,0,6,126,255,255,189,219,231,255,126
630 prepa_grafo 11,6,6,255,255,255,255,255,255,255,255
640 CERTO=1
650 FALSO=0
660 OTRA=CERTO
670 DIM X%(50)
680 DIM Y%(50)
690 MAX%=50
700 DIM T$(5,15)
710 DIM T(5)
720 FOR I=1 TO 5
730  T(I)=0
740 END FOR I
799 END DEFine prepa_juego

1000 DEFine PROCedure prepa_manga
1010 NO_COMIDA=FALSO
1020 NO_VIDAS=FALSO
1030 GOLPE=FALSO
1040 C%=0
1050 STOTAL%=0
1060 SC=0
1070 SN=0
1080 COMIDA=0
1099 END DEFine prepa_manga

1300 DEFine PROCedure prepa_grafo(S%,F%,B%,
     R0%,R1%,R2%,R3%,R4%,R5%,R6%,R7%)
1310 LOCal A,FH%,FL%,BH%,BL%
1320 A=C_TAB+32*S%
1330 FH%=(F% DIV 4)*2
1340 FL%=F% && 3
1350 BH%=(B% DIV 4)*2
1360 BL%=B% && 3
1370 prepa_raya(R0%)
1380 prepa_raya(R1%)
```

```
1390 prepa_raya(R2%)
1400 prepa_raya(R3%)
1410 prepa_raya(R4%)
1420 prepa_raya(R5%)
1430 prepa_raya(R6%)
1440 prepa_raya(R7%)
1499 END DEFine prepa_grafo

1500 DEFine PROCedure prepa_raya(R%)
1510 LOCal M%,J,I,DL%,DH%
1520 M%=128
1530 FOR J=1 TO 2
1540  DL%=0:DH%=0
1550  FOR I=1 TO 4
1560   IF (R% && M%)=M% THEN
1570    DL%=FH%+DL%*4
1580    DH%=FL%+DH%*4
1590   ELSE
1600    DL%=BH%+DL%*4
1610    DH%=BL%+DH%*4
1620   END IF
1630   M%=M% DIV 2
1640  END FOR I
1650  POKE A,DL%
1660  POKE A+1,DH%
1670  A=A+2
1680 END FOR J
1699 END DEFine prepa_raya

1700 DEFine PROCedure porte_grafo(X%,Y%,S%)
1710 CALL dire_rus,X%,Y%,S%,C_TAB
1799 END DEFine porte_grafo

2000 DEFine PROCedure trace_escena
2010 CLS
2020 trace_culebra
2030 trace_comida 6,4*SC
2040 IF COMIDA=0 THEN COMIDA=RND(1 TO 10)+15
2050 trace_comida 5,COMIDA
2060 AT 22,3
2070 PRINT 'Puntos ';STOTAL%+C%
2099 END DEFine trace_escena

2500 DEFine PROCedure trace_comida (F,MONTO)
2510 IF MONTO=0 THEN END DEFine trace_comida
2520 FOR K=1 TO MONTO
2525  REPeat pos
2530   U%=RND(1 TO 30)
2540   V%=RND(1 TO 21)
2550   IF tinte(V%,U%,1,1)=6 THEN EXIT pos
2560  END REPeat pos
2570  porte_grafo U%,V%,F
2580 END FOR K
```

```
2599 END DEFine trace_comida

2600 DEFine PROCedure trace_culebra
2610 A%=2
2620 B%=10
2630 XH%=A%
2640 YH%=B%
2650 H%=0
2660 S%=1
2670 T%=2
2680 porte_grafo A%,B%,T%
2690 X%(1)=A%
2700 Y%(1)=B%
2710 FOR Q=2 TO 10
2730  A%=A%+1
2740  porte_grafo A%,B%,S%
2750  X%(Q)=A%
2760  Y%(Q)=B%
2770 END FOR Q
2780 A%=A%+1
2790 porte_grafo A%,B%,H%
2800 X%(11)=A%
2810 Y%(11)=B%
2820 XV%=1
2830 YV%=0
2840 Z%=1
2850 Q%=11
2899 END DEFine trace_culebra

3000 DEFine PROCedure mueva_paso
3010 mire_rumbo
3020 actualice
3099 END DEFine mueva_paso

4000 DEFine PROCedure mire_rumbo
4002 BEEP .1,-20
4004 PAUSE 2
4006 BEEP
4010 A=KEYROW(1)
4020 IF A=0 THEN END DEFine mire_rumbo
4030 SELect ON A
4040  ON A=4
4050   XV%=0
4060   YV%=-1
4070   H%=7
4080   S%=8
4090  ON A=128
4100   XV%=0
4110   YV%=1
4120   H%=9
4130   S%=10
4140  ON A=2
4150   XV%=-1
```

```
4160    YV%=0
4170    H%=3
4180    S%=4
4190   ON A=16
4200    XV%=1
4210    YV%=0
4220    H%=0
4230    S%=1
4240 END SELect
4299 END DEFine mire_rumbo

4500 DEFine PROCedure actualice
4510 XH%=A%:YH%=B%
4520 A%=A%+XV%:B%=B%+YV%
4530 IF A%>30 THEN A%=1
4540 IF A%<1 THEN A%=30
4550 IF B%>25 THEN B%=0
4560 IF B%<0 THEN B%=25
4570 IF tinte(B%,A%,4,4)<>6 THEN golpalgo
4580 porte_grafo XH%,YH%,S%
4590 porte_grafo A%,B%,H%
4600 Q%=Q%+1
4610 IF Q%>MAX% THEN Q%=1
4620 X%(Q%)=A%:Y%(Q%)=B%
4630 porte_grafo X%(Z%),Y%(Z%),11
4640 Z%=Z%+1
4650 IF Z%>MAX% THEN Z%=1
4660 porte_grafo X%(Z%),Y%(Z%),T%
4699 END DEFine actualice

5000 DEFine PROCedure golpalgo
5010 COL=tinte(B%,A%,4,4)
5020 SELect ON COL
5030  ON COL=4
5040   crece_lebra
5050  ON COL=REMAINDER
5060   muere_lebra
5070  END SELect
5099 END DEFine golpalgo

5500 DEFine PROCedure muere_lebra
5510 BEEP .1,-10
5512 PAUSE 2
5515 BEEP
5520 BEEP .1,-20
5524 PAUSE 2
5526 BEEP
5530 SN=SN+1
5540 IF SN<4 THEN
5560  trace_escena
5570 ELSE
5580  STOTAL%=STOTAL%+C%
5590  NO_VIDAS=CERTO
```

```
5600 END IF
5699 END DEFine muere_lebra

5700 DEFine PROCedure crece_lebra
5710 BEEP .1,10
5715 PAUSE 2
5720 BEEP
5730 C%=C%+1
5740 COMIDA=COMIDA-1
5750 AT 22,3
5760 PRINT 'Puntos= ';STOTAL%+C%
5770 Z%=Z%-1
5780 IF Z%<1 THEN Z%=MAX%
5790 IF COMIDA<>0 THEN END DEFine crece_lebra
5800 STOTAL%=STOTAL%+C%
5810 C%=0
5820 IF SC<4 THEN
5830  SC=SC+1
5840  trace_escena
5850 ELSE
5860  NO_COMIDA=CERTO
5870 END IF
5899 END DEFine crece_lebra

6000 DEFine PROCedure expo_super
6010 RESTORE 6010
6020 DATA 8,1,1,0,11,2,11,11,7,11,8,1,1,1
6030 DATA 11,8,1,1,0,11,8,1,1,1
6040 DATA 8,11,11,11,11,10,11,11,8,11,8,11,11,10
6050 DATA 11,8,11,11,11,11,8,11,11,10
6060 DATA 4,4,4,8,11,10,11,11,8,11,8,3,4,10
6070 DATA 11,8,4,8,11,11,8,4,4,10
6080 DATA 11,11,11,8,11,10,11,11,8,11,8,11,11,11
6090 DATA 11,8,1,1,11,11,8,10,1,11
6092 DATA 11,11,11,8,11,10,11,11,8,11,8,11,11,11
6094 DATA 11,8,11,11,11,11,8,11,10,1
6100 DATA 2,1,1,1,11,10,1,1,1,11,2,11,11,11,11
6110 DATA 4,4,4,2,11,2,11,11,9
6120 PAUSE 20
6130 CLS
6140 FOR I=2 TO 7
6150  FOR J=3 TO 26
6160   READ S%
6170   porte_grafo J,I,S%
6180  END FOR J
6190 END FOR I
6199 END DEFine expo_super

7000 DEFine PROCedure fine_manga
7010 CLS
7015 AT 3,10
7020 SELect ON SC
7030  ON SC=0
```

```
7040   PRINT 'Degradado a simple gusano'
7050  ON SC=1
7060   PRINT 'Las ranas no tienen que preocuparse'
7070  ON SC=2
7080   PRINT 'No esta mal para una culebra de cesped'
7090  ON SC=3
7100   PRINT 'Bien aprovechado el paseo'
7110  ON SC=4
7140   IF NO_COMIDA THEN
7150    expo_super
7160    exhibicione
7170   ELSE
7180    PRINT 'Un paseo virulento y maligno'
7190   END IF
7200 END SELect
7210 FOR I= 5 TO 1 STEP -1
7220  IF STOTAL%>=T(I) THEN N=I
7230 END FOR I
7235 AT 15,10
7240 IF N=0 THEN
7250  PRINT 'Tus puntos';STOTAL%
7260 ELSE
7270  PRINT 'Ocupas ahora el rango ';N
7280  PRINT TO 10;'en la liga culebrera'
7290 IF N<5 THEN
7300  FOR I=5 TO N+1 STEP -1
7310   T(I)=T(I-1)
7320   T$(I)=T$(I-1)
7330  END FOR I
7340 END IF
7350 T(N)=STOTAL%
7355 PRINT TO 10;
7360 INPUT 'Dime tu nombre';T$(N)
7370 CLS
7380 FOR I=1 TO 5
7390  AT I*2+5,5
7400  PRINT T$(I,1 TO 15);
7410  PRINT TO 21; T(I)
7420 END FOR I          7430 - END IF
7440 REPeat otro_juego
7445  AT 21,10
7450  INPUT 'Otro paseo ';A$
7455  IF A$='S' OR A$='N' THEN EXIT otro_juego
7460  END REPeat otro_juego
7470 IF A$='N' THEN
7480  OTRA=FALSO
7490 ELSE
7500  OTRA=CERTO
7510 END IF
7599 END DEFine fine_manga

8000 DEFine PROCedure prologue
8010 PAPER 6
```

```
8020 CLS
8030 INK 0
8040 exhibicione
8045 CLS
8050 AT 4,2
8060 PRINT 'Debes guiar a tu culebra usando'
8065 PRINT '  las cuatro teclas de flechas y'
8070 PRINT '  comerte todas las ranas verdes';
8080 porte_grafo 19,8,5
8085 PRINT \\
8090 PRINT '  Si intentas comerte un sapo'
8100 porte_grafo 20,10,6
8110 PRINT '  o a ti mismo'
8120 PRINT '  entonces perderas una de'
8130 PRINT '  tus cinco vidas'\\
8140 PRINT '  tienes cinco manadas'
8145 PRINT '  de ranas para comerte antes de'
8150 PRINT '  convertirte en una SUPER CULEBRA'
8155 REPeat espera
8156  AT 23,7
8160  FLASH 1
8170  PRINT 'PULSA CUALQUIER TECLA'
8180  FLASH 0
8190  IF INKEY$<>'' THEN EXIT espera
8195 END REPeat espera
8199 END DEFine prologue

8500 DEFine PROCedure exhibicione
8510 DATA 8,1,1,0,11,8,1,1,1,11,8,1,1,1,11
8520 DATA 2,11,8,0,11,8,1,1,0
8530 DATA 8,11,11,11,11,8,11,11,10,11,8,11,11,10
8540 DATA 11,10,8,1,11,11,8,11,11,11
8545 DATA 4,4,4,8,11,8,11,11,10,11,8,1,1
8550 DATA 10,11,10,8,11,11,11,4,4,8,11
8560 DATA 11,11,11,8,11,8,11,11,10,11,8,10,1,1
8570 DATA 11,10,4,8,11,11,8,1,1,11
8580 DATA 11,11,11,8,11,8,11,11,10,11,8,11,11
8590 DATA 10,11,10,11,4,8,11,8,11,11,11
8600 DATA 2,1,1,1,11,2,11,11,9,11,2,11,11,9
8610 DATA 11,9,11,11,2,11,4,4,4,2
8620 RESTORE 8500
8630 FOR I=9 TO 14
8640  FOR J=3 TO 26
8650   READ S
8660   porte_grafo J,I,S
8670  END FOR J
8680 END FOR I
8690 PAUSE 100
8699 END DEFine exhibicione

9500 DEFine FuNction tinte(R%,C%,X%,Y%)
9510 LOCal A%,P%,B%,A
9520 P%=C%*8+X%
```

```
9530 A=131072+(512*R%+Y%*64+P% DIV 4)*2
9540 B%=PEEK(A+1)
9550 A%=PEEK(A)
9560 P%=P% MOD 4
9570 IF P%=3 THEN RETurn ((2 && A%)*2)+(3 && B%)
9580 IF P%=2 THEN RETurn ((8 && A%)DIV 2)+
     ((12 && B%) DIV 4)
9590 IF P%=1 THEN RETurn ((32 && A%)DIV 8)+
     ((48 && B%) DIV 16)
9600 IF P%=0 THEN RETurn ((128 && A%)DIV 32)+
     ((192 && B%) DIV 64)
9699 END DEFine tinte
```

Capítulo Seis

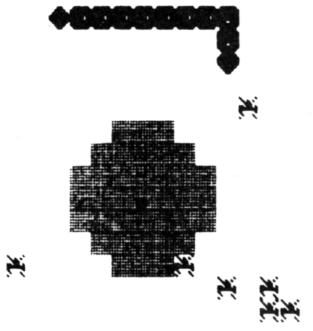
Charcajo

La culebra y la animación de la misma desarrollada en el último capítulo puede usarse para crear un amplio surtido de juegos excitantes. Charcajo con sus charcas de renacuajos, es justamente uno de esos juegos y este capítulo muestra cómo se puede modificar el programa anterior para producir un nuevo juego.

El diseño del juego

La idea fundamental de este juego es similar al desarrollado en el capítulo anterior, en cuanto que una culebra animada tiene que comerse ranas para ganar puntos. La diferencia es que ahora las ranas también se mueven. Al comienzo del juego las ranas son todavía renacuajos inmóviles colocados en una charca que aparece en medio de la pantalla. Uno a uno se van convirtiendo en ranas y salen disparados para alcanzar el borde de la pantalla. La culebra no se puede comer los renacuajos hasta que se hayan convertido en ranas, y desde luego pierde una de sus vidas si cae dentro de la charca. El objeto del juego es simplemente comerse tantas ranas como se pueda antes de que logren la escapada por el borde de la pantalla.

Además de añadir la charca y de dar movimiento a las ranas, el resto de los elementos de este juego continúa sin cambio con respecto al juego que nos sirve de referencia. En concreto, la culebra dispone de cinco vidas y hay cinco escenas con ranas para comerse. Si la culebra pierde una de sus vidas, entonces las ranas restantes milagrosamente vuelven a renacer como renacuajos y la culebra dispone de otra oportunidad para descansar en espera de su conversión a ranas. Aparte del motivo gráfico renacuajo y de un símbolo que es un cuadratín en ciano para la charca, todos los símbolos gráficos usados en el nuevo juego ya han sido presentados. La imagen en pantalla (véase Fig. 6.1) es muy similar a la producida en Culebreo, siendo el único añadido la charca que hay en el medio, pero deberá recordarse que en el anterior las ranas eran estáticas, mientras que aquí se moverán.



Culebra = 1 Tanda = 1 Puntos = 0

Fig. 6.1.

Preparando el escenario

Las modificaciones que haremos caen en dos clases: primeramente los cambios y añadidos necesarios para preparar el juego, y en segundo lugar, los cambios y añadidos necesarios para animar los motivos ranas. Esta sección trata con la preparación del juego.

La mayoría de las modificaciones necesarias para preparar el juego se han de hacer en el procedimiento **trace_escena.** Ahora, en lugar de exponer ranas y renacuajos, tiene que exponer una charca en medio de la pantalla que contiene a esos renacuajos. La charca es mejor producirla recurriendo a un nuevo procedimiento, **trace_charca** y los renacuajos por una modificación del procedimiento **trace_comida.**

La nueva versión del procedimiento empleado para trazar la escena es:

```
2000 DEFine PROCedure trace_escena
2010 CLS
2020 trace_culebra
2030 trace_charca
2040 IF COMIDA=0 THEN COMIDA=NUM%
2050 trace_comida (COMIDA)
2060 AT 24,3
2064 PRINT 'Culebra=';SN;
2068 PRINT ' Tanda=';SC;
2070 PRINT ' Puntos=';STOTAL%+C%
2080 LINE 0,10 TO 256,10
2099 END DEFine trace_escena
```

El nuevo procedimiento para dibujar la charca se cita en la línea 2030. El procedimiento **trace_comida** se cita ahora sólo una vez (línea 2050) para exponer un número prefijado de renacuajos dentro de la charca. Los únicos dos cambios son la inclusión de las líneas 2064 a 2070 para exponer en pantalla el número ordinal de culebra y 'escena', y la inclusión de un comando para gráficos de alta resolución (línea 2080) para trazar una línea horizontal que muestre el límite de la cancha de juego.

El procedimiento **trace_charca** simplemente recurre a **porte_grafo** para colocar los cuadratínes ciano en la pantalla de manera que tengan forma de 'charca'.

```
2200 DEFine PROCedure trace_charca
2205 LOCal T
2206 T=1
2210 FOR Y=12 TO 14
2220  FOR X=14-T TO 14+T
2230   porte_grafo X,Y,12
2240   porte_grafo X,Y+(8-(T*2)),12
2250  END FOR X
2255  T=T+1
2260 END FOR Y
2270 FOR X=11 TO 17
2280  porte_grafo X,15,12
2290 END FOR X
2299 END DEFine trace_charca
```

Todo lo que queda ahora es definir esos símbolos cuadrados ciano como motivo gráfico 12 dentro del procedimiento **prepa_juego.** Es decir, incluir la línea:

```
635 prepa_grafo 12,5,5,255,255,255,255,255,255,
    255,255
```

Se usa el ciano en lugar del azul porque produce un mejor contraste con los otros colores en un televisor en blanco y negro.

El procedimiento **trace_comida** tiene ahora que colocar renacuajos en la charca en lugar de ranas por cualquier parte de la pantalla. Eso se procura fácilmente si se restringe la banda de números aleatorios generados en las líneas 2530 y 2540. Más adelante en el programa, los renacuajos y las ranas en que se convierten tienen que ser dotados de movimiento y eso implica que sus posiciones sean conocidas. Para seguir el rastro de dónde están los renacuajos y las ranas, es necesario usar una pareja de tablas para conservar las coordenadas X e Y de cada renacuajo/rana.

```
2500 DEFine PROCedure trace_comida (MONTO)
2510 IF MONTO=0 THEN END DEFine trace_comida
2520 FOR K=1 TO MONTO
2525  REPeat pos
2530   U%=RND(1 TO 7)+10
2540   V%=RND(1 TO 7)+9
2550   IF tinte(V%,U%,3,3)=5 THEN EXIT pos
2560  END REPeat pos
2570  porte_grafo U%,V%,6
2575  FX%(K)=U%:FY%(K)=V%
```

```
2580 END FOR K
2582 IF MONTO<NUM% THEN
2584  FOR K=MONTO+1 TO NUM%
2586   FX%(K)=-1
2588  END FOR K
2590 END IF
2599 END DEFine trace_comida
```

La línea 2575 preserva las coordenadas en FX% y FY% para su uso posterior. El número máximo de renacuajos/ranas es de ocho, pero en etapas posteriores del juego puede que haya menos, debido a que se hayan escapado las ranas o hayan sido comidas. Para indicar que un renacuajo/rana no está siendo empleado en el juego, su coordenada X se fija a -1. El bucle preconfinado, líneas 2584 a 2588 coloca los elementos no usados de la tabla FX% a dicho valor -1. Desde luego, la nueva pareja de tablas tiene que tener sus dimensiones establecidas y el símbolo renacuajo haber sido descrito, así que hay que añadir al procedimiento **prepa_juego**, las líneas:

```
580 prepa_grafo 6,0,5,0,24,56,24,16,16,32,64

750 NUM%=8
760 DIM FX%(NUM%),FY%(NUM%)
```

Intentando la escapada

Una vez que se ha preparado la imagen inicial y las coordenadas de todos los renacuajos están reflejadas en la pareja de tablas FX% y FY%, puede comenzar la animación de la culebra y de las ranas. El único cambio en el bucle de animación es la incluisón de una cita al procedimiento **mueva_rana** que aleatoriamente desplaza una sola rana. Es decir, el procedimiento **mueva_paso** se convierte en:

```
3000 DEFine PROCedure mueva_paso
3005 IF RND>.5 THEN mueva_rana
3010 mire_rumbo
3020 actualice
3099 END DEFine mueva_paso
```

El procedimiento **mueva_rana** es un poco más complicado de lo que se pudiera esperar. El problema principal está en asegurarse que la rana no se entrega a la culebra, en usar el color de fondo correcto para exponer y borrar el motivo rana, y en detectar cuándo una rana ha conseguido escapar. De hecho, la mayoría de las soluciones a esos problemas ya se han presentado en capítulos anteriores.

```
3500 DEFine PROCedure mueva_rana
3510 K%=RND(1 TO NUM%)
3520 IF FX%(K%)=-1 THEN  END DEFine mueva_rana
3530 N%=tinte(FY%(K%),FX%(K%),4,4)
3540 IF N%=5 THEN
3550  porte_grafo FX%(K%),FY%(K%),12
3560 ELSE
3570  IF N%<>0 THEN porte_grafo FX%(K%),FY%(K%),11
3580 END IF
3585 REPeat saltada
3590  FX%(K%)=FX%(K%)-sgn(13.5-FX%(K%))
3600  FY%(K%)=FY%(K%)-sgn(14.5-FY%(K%))
3610  IF FX%(K%)<0 OR FX%(K%)>31 THEN huye_rana:
      END DEFine mueva_rana
3620  IF FY%(K%)<0 OR FY%(K%)>24 THEN huye_rana:
      END DEFine mueva_rana
3630  N%=tinte(FY%(K%),FX%(K%),4,4)
3640  IF N%=6 THEN EXIT saltada
3650 END REPeat saltada
3660 porte_grafo FX%(K%),FY%(K%),5
3699 END DEFine mueva_rana

3700 DEFine FuNction sgn(Q)
3710 IF Q=0 THEN RETurn 0
3720 IF Q>0 THEN
3730  RETurn 1
3740 ELSE
3750  RETurn -1
3760 END IF
3799 END DEFine sgn
```

La línea 3510 elige un renacuajo/rana para moverlo al azar. Si la coordenada X de ese renacuajo/rana vale -1, indica que o bien ha sido comido o bien ya ha logrado escapar, y por lo tanto la línea 3520 devuelve el control al procedimiento **mueva_paso.** La línea 3530 halla el color del fondo en la vieja posición de la rana y preserva dicho color en N%. Mientras que ese color no sea negro, las líneas 3540 a 3580 eliminan la rana de su vieja posición usando para ello un cuadratín del color apropiado. Las líneas 3585 a 3600 actualizan las coordenadas de la rana de manera que se mueva al borde de la pantalla. La manera en que esto funciona es fácil de comprender una vez que sabes que aplicando la función signo **sgn**(13.5-FX%(K%)) se obtiene +1 si la rana está a la izquierda del centro de la charca, y se obtiene -1 si esá a la derecha. (La función **sgn** se presentó en el Capítulo Dos). Por lo tanto, restando **sgn**(13.5-FX%(K%)) de la coordenada X corriente de la rana siempre da como resultado alejarse del centro de la charca. Un razonamiento similar muestra que restando **sgn**(14.5-FY%(K%)) de la coordenada Y corriente de la rana, da también como resultado un alejamiento del centro de la charca. Después de eso, en las líneas 3610 a 3620 se comprueba si la rana ya ha huido fuera de la pantalla y se recurre al procedimiento **huye_rana** si así ha sido. La línea final del procedimiento simplemente expone la rana en su nueva posición.

El procedimiento **mueva_rana** recurre el procedimiento **huye_rana** para marcar adecuadamente el hecho de que una rana se haya escapado.

```
3900 DEFine PROCedure huye_rana
3910 FX%(K%)=-1
3920 COMIDA=COMIDA-1
3930 IF COMIDA<>0 THEN END DEFine huye_rana
3940 STOTAL%=STOTAL%+C%
3950 C%=0
3960 IF SC=4 THEN
3962  NO_COMIDA=CERTO
3964 ELSE
3970  SC=SC+1
3980  trace_escena
3990 END IF
3999 END DEFine huye_rana
```

La línea 3910 fija la coordenada horizontal de la rana al valor -1 para indicar al resto del programa que esa rana ya ha escapado. El resto del procedimiento simplemente resta 1 de la COMIDA corriente y vuelve a reanudar el juego con una nueva escena completa de renacuajos/ranas, a no ser que ya se hayan comido -o se hayan escapado- las cinco tandas de ranas disponibles.

Atrapando ranas

Los cambios que quedan por hacer en el programa son primordialmente cambios que han de hacerse al procedimiento **crece_lebra.** Este procedimiento se lleva a cabo siempre que la culebra se topa con un objeto verde, i.e. ¡una rana! En la versión original de Culebreo, todo lo que este procedimiento tenía que hacer era añadir 1 a la puntuación y comprobar si ya se habían comido todas las ranas. Ahora, tiene que quitar la rana cazada de la lista de coordenadas en las tablas FX% y FY%. Eso se efectúa apelando a un nuevo procedimiento, **marq_rana** mediante la cita:

```
5745 marq_rana
```

Además de eso, se aprovecha la oportunidad para añadir las líneas 5750 a 5760 que expondrán los números de culebra y escena y los puntos alcanzados:

```
5750 AT 24,3
5752 PRINT 'Culebra=';SN;
5754 PRINT ' Tanda=';SC;
5760 PRINT ' Puntos=';STOTAL%+C%
```

La posición de la rana que la culebra acaba de cazar queda reflejada en A% y B%. El procedimiento **marq_rana** la quita de la lista de ranas disponibles, escrutando para ello las tablas FX% y FY% en busca de una rana con las mismas coordenadas, y luego haciendo que su coordenada X valga -1.

```
5900 DEFine PROCedure marq_rana
5910 K%=0
5920 REPeat busqueda
5930  K%=K%+1
5940  IF A%=FX%(K%) AND B%=FY%(K%) THEN EXIT busqueda
5960 END REPeat busqueda
5970 FX%(K%)=-1
5999 END DEFine marq_rana
```

Los toques finales al programa que estamos desarrollando implican alterar las puntuaciones por las que se eligen los mensajes lanzados mediante el procedimiento **fine_manga.**

```
7010 CLS
7015 AT 3,10
7016 STOTAL=STOTAL%
7020 SELect ON STOTAL
7030  ON STOTAL=0 TO 9
7040   PRINT 'Degradado a simple gusano'
7050  ON STOTAL=10 TO 19
7060   PRINT 'Las ranas no tienen que preocuparse'
7070  ON STOTAL=20 TO 29
7080   PRINT 'No esta mal para una culebra de cesped'
7090  ON STOTAL=30 TO 34
7100   PRINT 'Bien aprovechado el paseo'
7110  ON STOTAL=REMAINDER
7140   IF NO_COMIDA THEN
7150    expo_super
7160    exhibicione
7170   ELSE
7180    PRINT 'Un paseo virulento y maligno'
7190   END IF
7200 END SELect
```

y las reglas dadas en el procedimiento **prologue:**

```
8085 PRINT \\
8090 PRINT ' No te caigas al agua ni intentes'
8110 PRINT '  comerte a ti mismo o perderas una'
8130 PRINT '  de tus cinco vidas.'\\
8140 PRINT '  Dispones de cinco manadas'
8145 PRINT '  de ranas para comerte antes de'
8150 PRINT '  convertirte en SUPER CULEBRA'
```

Conclusión

La conversión de Culebreo a Charcajo ha sido fácil y sirve para demostrar la ventaja de usar **procedimientos** y **funciones** al confeccionar un programa. Todavía hay muchas maneras en que puede mejorarse este programa, pero la presente versión del juego ya es divertida y los desarrollos adicionales quedan a tu imaginación y exploración. Podrías por ejemplo, permitir que algunas de las ranas sean venenosas o permitirlas que se muevan para esquivar a la culebra...

El programa final - un listado completo

```
 10 REMark TADPOLE
 20 WINDOW 512,256,0,0
 30 prepa_juego
 40 prologue
 50 REPeat manga
 60  prepa_manga
 70  trace_escena
 80  REPeat movida
 90   mueva_paso
100   IF NO_COMIDA OR NO_VIDAS THEN EXIT movida
110  END REPeat movida
120  fine_manga
130  IF NOT OTRA THEN EXIT manga
140 END REPeat manga
199 STOP

350 DEFine PROCedure prepa_juego
360 DATA -5245,-10109,-7806,-11646
370 DATA -6783,1665,2,0,8769,8316
380 DATA 0,0,9264,18432,8834,-11524,128
390 DATA 22664,-20228,32,26350,28672,20085
400 dire_rus=RESPR(100)
410 RESTORE 360
420 FOR I=dire_rus TO dire_rus+22*2  STEP 2
430  READ B
440  POKE_W I,B
450 END FOR I
510 C_TAB=RESPR(32*13)
520 prepa_grafo 0,0,6,56,124,250,255,255,250,124,56
530 prepa_grafo 1,0,6,126,239,247,251,251,247,239,126
540 prepa_grafo 2,0,6,24,60,126,255,255,126,60,24
550 prepa_grafo 3,0,6,28,62,95,255,255,95,62,28
560 prepa_grafo 4,0,6,126,247,239,223,223,239,247,126
570 prepa_grafo 5,4,6,130,84,56,60,254,130,68,130
580 prepa_grafo 6,0,5,0,24,56,24,16,16,32,64
590 prepa_grafo 7,0,6,24,60,90,255,255,255,126,60
600 prepa_grafo 8,0,6,60,255,231,219,189,255,255,126
610 prepa_grafo 9,0,6,60,126,255,255,126,90,60,24
620 prepa_grafo 10,0,6,126,255,255,189,219,231,255,126
```

```
630 prepa_grafo 11,6,6,255,255,255,255,255,255,255,255
635 prepa_grafo 12,5,5,255,255,255,255,255,255,255,255

640 CERTO=1
650 FALSO=0
660 OTRA=CERTO
670 DIM X%(50)
680 DIM Y%(50)
690 MAX%=50
700 DIM T$(5,15)
710 DIM T(5)
720 FOR I=1 TO 5
730  T(I)=0
740 END FOR I
750 NUM%=8
760 DIM FX%(NUM%),FY%(NUM%)
799 END DEFine prepa_juego

1000 DEFine PROCedure prepa_manga
1010 NO_COMIDA=FALSO
1020 NO_VIDAS=FALSO
1030 GOLPE=FALSO
1040 C%=0
1050 STOTAL%=0
1060 SC=0
1070 SN=0
1080 COMIDA=0
1099 END DEFine prepa_manga

1300 DEFine PROCedure prepa_grafo(S%,F%,B%,
     R0%,R1%,R2%,R3%,R4%,R5%,R6%,R7%)
1310 LOCal A,FH%,FL%,BH%,BL%
1320 A=C_TAB+32*S%
1330 FH%=(F% DIV 4)*2
1340 FL%=F% && 3
1350 BH%=(B% DIV 4)*2
1360 BL%=B% && 3
1370 prepa_raya(R0%)
1380 prepa_raya(R1%)
1390 prepa_raya(R2%)
1400 prepa_raya(R3%)
1410 prepa_raya(R4%)
1420 prepa_raya(R5%)
1430 prepa_raya(R6%)
1440 prepa_raya(R7%)
1499 END DEFine prepa_grafo

1500 DEFine PROCedure prepa_raya(R%)
1510 LOCal M%,J,I,DL%,DH%
1520 M%=128
1530 FOR J=1 TO 2
1540  DL%=0:DH%=0
1550  FOR I=1 TO 4
```

```
1560    IF (R% && M%)=M% THEN
1570     DL%=FH%+DL%*4
1580     DH%=FL%+DH%*4
1590    ELSE
1600     DL%=BH%+DL%*4
1610     DH%=BL%+DH%*4
1620    END IF
1630    M%=M% DIV 2
1640   END FOR I
1650   POKE A,DL%
1660   POKE A+1,DH%
1670   A=A+2
1680 END FOR J
1699 END DEFine prepa_raya

1700 DEFine PROCedure porte_grafo(X%,Y%,S%)
1710 CALL dire_rus,X%,Y%,S%,C_TAB
1799 END DEFine porte_grafo

2000 DEFine PROCedure trace_escena
2010 CLS
2020 trace_culebra
2030 trace_charca
2040 IF COMIDA=0 THEN COMIDA=NUM%
2050 trace_comida COMIDA
2060 AT 24,3
2064 PRINT 'Culebra=';SN;
2068 PRINT ' Tanda=';SC;
2070 PRINT ' Puntos=';STOTAL%+C%
2080 LINE 0,10 TO 256,10
2099 END DEFine trace_escena

2200 DEFine PROCedure trace_charca
2205 LOCal T
2206 T=1
2210 FOR Y=12 TO 14
2220  FOR X=14-T TO 14+T
2230   porte_grafo X,Y,12
2240   porte_grafo X,Y+(8-(T*2)),12
2250  END FOR X
2255  T=T+1
2260 END FOR Y
2270 FOR X=11 TO 17
2280  porte_grafo X,15,12
2290 END FOR X
2299 END DEFine trace_charca

2500 DEFine PROCedure trace_comida (MONTO)
2510 IF MONTO=0 THEN END DEFine trace_comida
2520 FOR K=1 TO MONTO
2525  REPeat pos
2530   U%=RND(1 TO 7)+10
2540   V%=RND(1 TO 7)+9
```

```
2550   IF tinte(V%,U%,3,3)=5 THEN EXIT pos
2560  END REPeat pos
2570  porte_grafo U%,V%,6
2575  FX%(K)=U%:FY%(K)=V%
2580 END FOR K
2582 IF MONTO<NUM% THEN
2584  FOR K=MONTO+1 TO NUM%
2586   FX%(K)=-1
2588  END FOR K
2590 END IF
2599 END DEFine trace_comida

2600 DEFine PROCedure trace_culebra
2610 A%=2
2620 B%=10
2630 XH%=A%
2640 YH%=B%
2650 H%=0
2660 S%=1
2670 T%=2
2680 porte_grafo A%,B%,T%
2690 X%(1)=A%
2700 Y%(1)=B%
2710 FOR Q=2 TO 10
2730  A%=A%+1
2740  porte_grafo A%,B%,S%
2750  X%(Q)=A%
2760  Y%(Q)=B%
2770 END FOR Q
2780 A%=A%+1
2790 porte_grafo A%,B%,H%
2800 X%(11)=A%
2810 Y%(11)=B%
2820 XV%=1
2830 YV%=0
2840 Z%=1
2850 Q%=11
2899 END DEFine trace_culebra

3000 DEFine PROCedure mueva_paso
3005 IF RND>.5 THEN mueva_rana
3010 mire_rumbo
3020 actualice
3099 END DEFine mueva_paso

3500 DEFine PROCedure mueva_rana
3505 IF RND<.5 THEN END DEFine mueva_rana
3510 K%=RND(1 TO NUM%)
3520 IF FX%(K%)=-1 THEN  END DEFine mueva_rana
3530 N%=tinte(FY%(K%),FX%(K%),4,4)
3540 IF N%=5 THEN
3550  porte_grafo FX%(K%),FY%(K%),12
3560 ELSE
```

```
3570  IF N%<>0 THEN porte_grafo FX%(K%),FY%(K%),11
3580 END IF
3585 REPeat saltada
3590  FX%(K%)=FX%(K%)-sgn(13.5-FX%(K%))
3600  FY%(K%)=FY%(K%)-sgn(14.5-FY%(K%))
3610  IF FX%(K%)<0 OR FX%(K%)>31 THEN huye_rana:
      END DEFine mueva_rana
3620  IF FY%(K%)<0 OR FY%(K%)>24 THEN huye_rana:
      END DEFine mueva_rana
3630  N%=tinte(FY%(K%),FX%(K%),4,4)
3640  IF N%=6 THEN EXIT saltada
3650 END REPeat saltada
3660 porte_grafo FX%(K%),FY%(K%),5
3699 END DEFine mueva_rana

3700 DEFine FuNction sgn(Q)
3710 IF Q=0 THEN RETurn 0
3720 IF Q>0 THEN
3730  RETurn 1
3740 ELSE
3750  RETurn -1
3760 END IF
3799 END DEFine sgn

3900 DEFine PROCedure huye_rana
3910 FX%(K%)=-1
3920 COMIDA=COMIDA-1
3930 IF COMIDA<>0 THEN END DEFine huye_rana
3940 STOTAL%=STOTAL%+C%
3950 C%=0
3960 IF SC=4 THEN
3962  NO_COMIDA=CERTO
3964 ELSE
3970  SC=SC+1
3980  trace_escena
3990 END IF
3999 END DEFine huye_rana

4000 DEFine PROCedure mire_rumbo
4010 A=KEYROW(1)
4020 IF A=0 THEN END DEFine mire_rumbo
4030 SELect ON A
4040  ON A=4
4050   XV%=0
4060   YV%=-1
4070   H%=7
4080   S%=8
4090  ON A=128
4100   XV%=0
4110   YV%=1
4120   H%=9
4130   S%=10
4140  ON A=2
```

```
4150    XV%=-1
4160    YV%=0
4170    H%=3
4180    S%=4
4190   ON A=16
4200    XV%=1
4210    YV%=0
4220    H%=0
4230    S%=1
4240 END SELect
4299 END DEFine mire_rumbo

4500 DEFine PROCedure actualice
4510 XH%=A%:YH%=B%
4520 A%=A%+XV%:B%=B%+YV%
4530 IF A%>30 THEN A%=1
4540 IF A%<1 THEN A%=30
4550 IF B%>25 THEN B%=0
4560 IF B%<0 THEN B%=25
4570 IF tinte(B%,A%,4,4)<>6 THEN golpalgo
4580 porte_grafo XH%,YH%,S%
4590 porte_grafo A%,B%,H%
4600 Q%=Q%+1
4610 IF Q%>MAX% THEN Q%=1
4620 X%(Q%)=A%:Y%(Q%)=B%
4630 porte_grafo X%(Z%),Y%(Z%),11
4640 Z%=Z%+1
4650 IF Z%>MAX% THEN Z%=1
4660 porte_grafo X%(Z%),Y%(Z%),T%
4699 END DEFine actualice

5000 DEFine PROCedure golpalgo
5010 COL=tinte(B%,A%,4,4)
5020 SELect ON COL
5030  ON COL=4
5040   crece_lebra
5050  ON COL=REMAINDER
5060   muere_lebra
5070  END SELect
5099 END DEFine golpalgo

5500 DEFine PROCedure muere_lebra
5510 BEEP .1,-10
5512 PAUSE 2
5515 BEEP
5520 BEEP .1,-20
5524 PAUSE 2
5526 BEEP
5530 SN=SN+1
5540 IF SN<4 THEN
5560  trace_escena
5570 ELSE
5580  STOTAL%=STOTAL%+C%
```

```
5590  NO_VIDAS=CERTO
5600 END IF
5699 END DEFine muere_lebra

5700 DEFine PROCedure crece_lebra
5710 BEEP .1,10
5715 PAUSE 2
5720 BEEP
5730 C%=C%+1
5740 COMIDA=COMIDA-1
5745 marq_rana
5750 AT 24,3
5752 PRINT 'Culebra=';SN;
5754 PRINT ' Tanda=';SC;
5760 PRINT ' Puntos=';STOTAL%+C%
5770 Z%=Z%-1
5780 IF Z%<1 THEN Z%=MAX%
5790 IF COMIDA<>0 THEN END DEFine crece_lebra
5800 STOTAL%=STOTAL%+C%
5810 K%=0
5820 IF SC<4 THEN
5830  SC=SC+1
5840  trace_escena
5850 ELSE
5860  NO_COMIDA=CERTO
5870 END IF
5899 END DEFine crece_lebra

5900 DEFine PROCedure marq_rana
5910 K%=0
5920 REPeat busqueda
5930  K%=K%+1
5940  IF A%=FX%(K%) AND B%=FY%(K%) THEN EXIT busqueda
5950 END REPeat busqueda
5960 FX%(K%)=-1
5999 END DEFine marq_rana

6000 DEFine PROCedure expo_super
6010 RESTORE 6010
6020 DATA 8,1,1,0,11,2,11,11,7,11,8,1,1,1
6030 DATA 11,8,1,1,0,11,8,1,1,1
6040 DATA 8,11,11,11,11,10,11,11,8,11,8,11,11,10
6050 DATA 11,8,11,11,11,11,8,11,11,10
6060 DATA 4,4,4,8,11,10,11,11,8,11,8,3,4,10
6070 DATA 11,8,4,8,11,11,8,4,4,10
6080 DATA 11,11,11,8,11,10,11,11,8,11,8,11,11,11
6090 DATA 11,8,1,1,11,11,8,10,1,11
6092 DATA 11,11,11,8,11,10,11,11,8,11,8,11,11,11
6094 DATA 11,8,11,11,11,11,8,11,10,1
6100 DATA 2,1,1,1,11,10,1,1,1,11,2,11,11,11,11
6110 DATA 4,4,4,2,11,2,11,11,9
6120 PAUSE 20
6130 CLS
```

```
6140 FOR I=2 TO 7
6150  FOR J=3 TO 26
6160   READ S%
6170   porte_grafo J,I,S%
6180  END FOR J
6190 END FOR I
6199 END DEFine expo_super

7000 DEFine PROCedure fine_manga
7010 CLS
7015 AT 3,10
7016 STOTAL=STOTAL%
7020 SELect ON STOTAL
7030  ON STOTAL=0 TO 9
7040   PRINT 'Degradado a simple gusano'
7050  ON STOTAL=10 TO 19
7060   PRINT 'Las ranas no tienen que preocuparse'
7070  ON STOTAL=20 TO 29
7080   PRINT 'No esta mal para una culebra de cesped'
7090  ON STOTAL=30 TO 34
7100   PRINT 'Bien aprovechado el paseo'
7110  ON STOTAL=REMAINDER
7140   IF NO_COMIDA THEN
7150    expo_super
7160    exhibicione
7170   ELSE
7180    PRINT 'Un paseo virulento y maligno'
7190   END IF
7200 END SELect
7210 FOR I= 5 TO 1 STEP -1
7220  IF STOTAL%>=T(I) THEN N=I
7230 END FOR I
7235 AT 15,10
7240 IF N=0 THEN
7250  PRINT 'Tus puntos ';STOTAL%
7260 ELSE
7270  PRINT 'Ocupas ahora el rango ';N
7280  PRINT TO 10;'en la liga culebrera'
7290 IF N<5 THEN
7300  FOR I=5 TO N+1 STEP -1
7310   T(I)=T(I-1)
7320   T$(I)=T$(I-1)
7330  END FOR I
7340 END IF
7350 T(N)=STOTAL%
7355 PRINT TO 10;
7360 INPUT 'Dime tu nombre ';T$(N)
7370 CLS
7380 FOR I=1 TO 5
7390  AT I*2+5,5
7400  PRINT T$(I,1 TO 15);
7410  PRINT TO 21; T(I)
7420 END FOR I
```

```
7440 REPeat otro_juego
7445  AT 21,10
7450  INPUT 'Otro paseo ';A$
7455  IF A$='S' OR A$='N' THEN EXIT otro_juego
7460  END REPeat otro_juego
7470 IF A$='N' THEN
7480  OTRA=FALSO
7490 ELSE
7500  OTRA=CERTO
7510 END IF
7599 END DEFine fine_manga

8000 DEFine PROCedure prologue
8010 PAPER 6
8020 CLS
8030 INK 0
8040 exhibicione
8045 CLS
8050 AT 4,2
8060 PRINT 'Debes guiar a tu culebra usando'
8065 PRINT '  las cuatro teclas de flechas y'
8070 PRINT '  comerte todas las ranas verdes';
8080 porte_grafo 19,8,5
8085 PRINT \\
8090 PRINT ' No te caigas al agua ni intentes'
8110 PRINT '  comerte a ti mismo o perderas una'
8130 PRINT '  de tus cinco vidas.'\\
8140 PRINT '  Dispones de cinco manadas'
8145 PRINT '  de ranas para comerte antes de'
8150 PRINT '  convertirte en SUPER CULEBRA'
8155 REPeat espera
8156  AT 23,7
8160  FLASH 1
8170  PRINT 'PULSA CUALQUIER TECLA'
8180  FLASH 0
8190  IF INKEY$<>'' THEN EXIT espera
8195 END REPeat espera
8199 END DEFine prologue

8500 DEFine PROCedure exhibicione
8510 DATA 8,1,1,0,11,8,1,1,1,11,8,1,1,1,11
8520 DATA 2,11,8,0,11,8,1,1,0
8530 DATA 8,11,11,11,11,8,11,11,10,11,8,11,11,10
8540 DATA 11,10,8,1,11,11,8,11,11,11
8545 DATA 4,4,4,8,11,8,11,11,10,11,8,1,1
8550 DATA 10,11,10,8,11,11,11,4,4,8,11
8560 DATA 11,11,11,8,11,8,11,11,10,11,8,10,1,1
8570 DATA 11,10,4,8,11,11,8,1,1,11
8580 DATA 11,11,11,8,11,8,11,11,10,11,8,11,11
8590 DATA 10,11,10,11,4,8,11,8,11,11,11
8600 DATA 2,1,1,1,11,2,11,11,9,11,2,11,11,9
8610 DATA 11,9,11,11,2,11,4,4,4,2
8620 RESTORE 8500
```

```
8630 FOR I=9 TO 14
8640  FOR J=3 TO 26
8650   READ S
8660   porte_grafo J,I,S
8670  END FOR J
8680 END FOR I
8690 PAUSE 100
8699 END DEFine exhibicione

9500 DEFine FuNction tinte(R%,C%,X%,Y%)
9510 LOCal A%,P%,B%,A
9520 P%=C%*8+X%
9530 A=131072+(512*R%+Y%*64+P% DIV 4)*2
9540 B%=PEEK(A+1)
9550 A%=PEEK(A)
9560 P%=P% MOD 4
9570 IF P%=3 THEN RETurn ((2 && A%)*2)+(3 && B%)
9580 IF P%=2 THEN RETurn ((8 && A%)DIV 2)+
     ((12 && B%) DIV 4)
9590 IF P%=1 THEN RETurn ((32 && A%)DIV 8)+
     ((48 && B%) DIV 16)
9600 IF P%=0 THEN RETurn ((128 && A%) DIV 32)+
     ((192 && B%) DIV 64)
9699 END DEFine tinte
```

Capítulo Siete

Scalebra

En este capítulo un juego tradicional y muy conocido en Inglaterra, que trata de **culebras** y **escaleras,** recibe nuevo vigor al ser implementado en ordenador. Los juegos tradicionales son una fuente obvia para cualquiera que esté buscando nuevas ideas en juegos por computadora. No sólo pueden implementarse de acuerdo con las reglas primitivas, sino que también constituyen el punto de arranque para juegos que únicamente pueden llevarse a cabo con la ayuda de un ordenador. Como ejemplo de esta idea, al final de este capítulo se modifica el juego tradicional de manera que es un juego para ordenador de acción rápida que nunca pudiera haberse jugado usando un **tablero** y **dados.**

Scalebra es diferente a todos los otros juegos de este libro en que aparentemente no involucra la animación de motivos gráficos. Es cierto que la principal dificultad en implementar este juego está en la construcción de los gráficos y del tablero; sin embargo, muchas de las técnicas de animación presentadas en capítulos anteriores demostrarán también aquí que son totalmente provechosas.

El diseño del juego

La forma y reglas del juego de Scalebra son conocidas por ¡todo el mundo que ya las conozca! pero nos centraremos en cómo traspasarlas al ordenador. La versión tradicional del juego (para los que no saben las reglas) consiste en un tablero dividido en 'escaques' o cuadros, con culebras y escaleras conectando cuadrados en diferentes filas. El juego habitualmente es para más de una persona a la vez, y su objetivo es ser el primero en mover un contador desde el cuadro de la esquina inferior izquierda del tablero hasta el cuadro final en la esquina superior izquierda. Los jugadores mueven alternativamente y el número de cuadros movidos está gobernado por el resultado de un dado. Si el movimiento de un jugador lo coloca en la cabeza de una culebra, entonces el jugador retrocede hasta la posición ocupada por la cola de la culebra. Igualmente, si el movimiento cae en el pie de una escalera, entonces el jugador avanza hasta la posición ocupada por la cima de la escalera. En otras palabras, si caes en la cabeza de una culebra, te deslizas hacia abajo, pero si caes al pie de una escalera subes por ella hasta el final.

En la versión para ordenador del juego, añadiríamos interés si creáramos el tablero de nuevo con una disposición aleatoria de culebras y escaleras cada vez que se comienza el juego.

También, la posición corriente de cada jugador podría marcarse usando un motivo gráfico con forma de hombre y ocupando un solo carácter, y así surgen diversas posibilidades para animar ese motivo cuando se desliza por las culebras o escala los peldaños. Otra tarea obvia para el programa es la de registrar el turno del jugador e incluso la de generar números aleatorios para eliminar la necesidad de los dados.

Como ya he mencionado, el reto principal en la implementación de este juego es la de diseñar y manejar los motivos gráficos de culebras y escaleras y los símbolos que los conforman. Un ejemplo del tablero generado por el programa, con sus culebras y escaleras puede verse en la Fig. 7.1. Las serpientes y las escaleras se producen usando un número variado de símbolos gráficos de un cuadratín, expuestos combinadamente. No sólo las posiciones de comienzo de las culebras y escaleras son aleatorias, sino que también lo son sus longitudes, y por ende la posición de terminación.

El único problema real en la posición del tablero es asegurarse que las culebras y escaleras no se cruzan unas con otras. Hay dos maneras de conseguir esta separación. Primeramente, la posición de comienzo y terminación de cada culebra y cada escalera podría registrarse en una tabla, y cada vez que se fuera a incluir algo en el tablero podría examinarse dicha tabla para ver si coincidía con algo que ya estuviera presente en el tablero. El segundo método es simplemente examinar cada cuadratín de la pantalla donde se va a llevar un nuevo motivo gráfico para asegurarse que los cuadratínes correspondientes están en 'claro'.

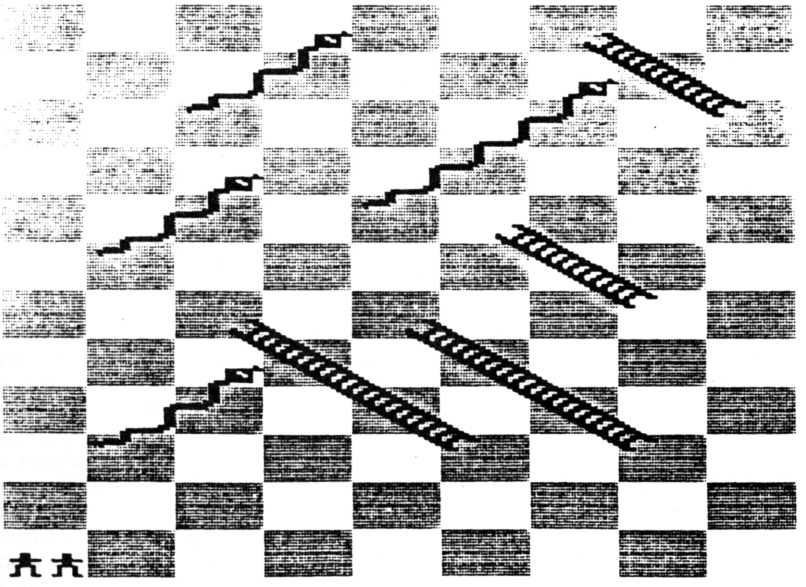


Fig. 7.1.

En la práctica, y a no ser que la velocidad sea de importancia extrema, es siempre más simple examinar lo que ya está presente en pantalla. Sin embargo, aunque este método es simple en teoría, hallar el símbolo que está presente en un cuadratín dado de la pantalla del QL es bastante difícil, y por lo tanto para este programa usaremos el método de almacenar los datos en tablas y consultar dichas tablas. En otras palabras, cada símbolo que se lleve a pantalla para formar el tablero, queda también reflejado en una tabla que posteriormente se usará para examinar el símbolo que hay colocado en una posición concreta del tablero.

Programa principal

Aunque este programa no es un juego con motivos gráficos **dinámicos**, la parte principal del programa tiene por ahora la forma ya familiar:

```
 10 REMark Culebras y escaleras
 15 PAPER 1
 20 WINDOW 512,256,0,0
 30 prepa_juego
 40 REPeat manga
 50  prologue
 60  decore
 70  empiece
 80  I%=0
 90  REPeat tirada
100   lance I%
110   IF YA_META THEN EXIT tirada
120   I%=NOT(I%)
130  END REPeat tirada
140  fine_manga
150  IF OTRA=FALSO THEN EXIT manga
160 END REPeat manga
199 STOP
```

Los primeros dos procedimientos **prepa_juego** y **prologue** ya te serán familiares si has leído los capítulos anteriores, pero observa que en este caso **prologue** he tenido que situarlo después de **prepa_juego** porque utiliza algunos de los símbolos gráficos implantados mediante **prepa_juego.** Después de eso, la línea 60 recurre al procedimiento **decore** que expone el tablero con sus culebras y escaleras en pantalla. Detrás de eso, se apela al procedimiento **empiece** para establecer los valores iniciales de las variables necesarias. El bucle condicionado, líneas 90 a 130, es el responsable de la marcha del juego. Cada vez que se da una ronda del bucle ambos jugadores efectúan un movimiento. La línea 100 recurre al procedimiento **lance** para permitir que el jugador I% tenga su turno. La variable YA_META se usa para indicar que uno de los jugadores ha alcanzado el cuadro situado en la esquina superior izquierda del tablero. Finalmente, cuando se termina un juego, la línea 140 cita el procedimiento **fine_manga** y la línea 150 vuelve a reanudar el juego o lo concluye definitivamente, dependiendo del estado de la variable OTRA.

Procedimientos prepa_juego y prologue

El procedimiento **prepa_juego** simplemente prepara los símbolos gráficos a usar en los motivos culebra y escalera, y los valores iniciales de las tablas empleadas en el programa.

```
350 DEFine PROCedure prepa_juego
360 DATA -5245,-10109,-7806,-11646
370 DATA -6783,1665,2,0,8769,8316
380 DATA 0,0,9264,18432,8834,-11524,128
390 DATA 22664,-20228,32,26350,28672,20085
400 dire_rus=RESPR(100)
410 RESTORE 360
420 FOR I=dire_rus TO dire_rus+22*2  STEP 2
430 READ B
440  POKE_W I,B
450 END FOR I
460 C_TAB=RESPR(32*23)
465 FOR Q=0 TO 1
470  prepa_grafo 11*Q+0,6,1+Q,0,2,127,108,[illegible],124,252,
    192
480  prepa_grafo 11*Q+1,6,1+Q,3,3,3,3,3,7,254,252
490  prepa_grafo 11*Q+2,6,1+Q,0,0,0,0,0,0,1,3
500  prepa_grafo 11*Q+3,6,1+Q,3,3,31,62,32,0,0,0
510  prepa_grafo 11*Q+4,6,1+Q,0,0,6,3,1,35,54,28
520  prepa_grafo 11*Q+5,6,1+Q,152,60,102,207,217,115,
    54,28
530  prepa_grafo 11*Q+6,6,1+Q,0,0,0,0,128,192,96,240
540  prepa_grafo 11*Q+7,6,1+Q,13,7,3,1,0,0,0,0
550  prepa_grafo 11*Q+8,6,1+Q,152,60,102,192,192,96,
    0,0
560  prepa_grafo 11*Q+9,3,1+Q,24,24,126,24,60,36,36,
    102
570  prepa_grafo 11*Q+10,1+Q,1+Q,255,255,255,255,255,
    255,255,255
580 END FOR Q
590 prepa_grafo 22,6,1,0,0,0,24,24,0,0,0
600 CERTO=1
610 FALSO=0
620 DIM ringl%(20,26)
630 DIM VX%(1)
640 DIM N$(1,20)
650 DIM X%(1)
660 DIM Y%(1)
699 END DEFine prepa_juego
```

La manera en que los símbolos gráficos definidos por el usuario se combinan para formar el motivo culebra o el motivo escalera, es bastante complicado y puede comprenderse mejor a partir de la Fig. 7.2. El juego demanda un total de veintitres combinaciones de símbolos según los colores de frente/fondo, y la Tabla 7.1 lo resume.

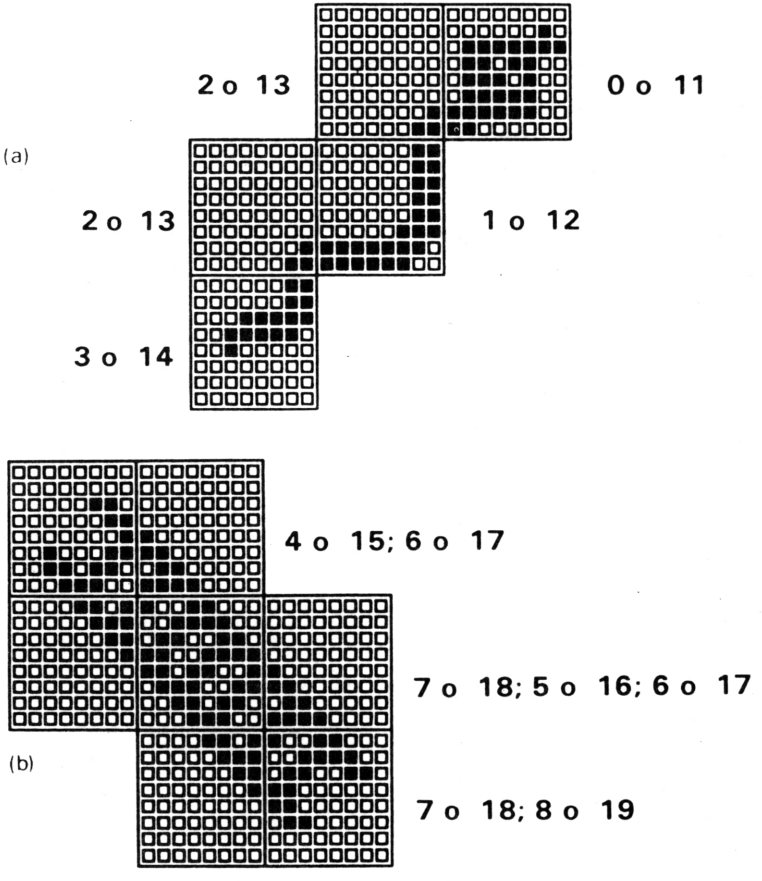


Fig. 7.2. Motivos gráficos (a) culebra y (b) escalera.

Tabla 7.1. Símbolos para los motivos gráficos

Número	Forma	Frente	Fondo
0	cabeza culebra	amarillo	azul
1	cuerpo culebra	amarillo	azul
2	esquina	amarillo	azul
3	cola culebra	amarillo	azul
4	cima escalera	amarillo	azul
5	medio escalera	amarillo	azul
6	derecha escalera	amarillo	azul
7	izquierda escalera	amarillo	azul
8	pie escalera	amarillo	azul
9	jugador	magenta	azul
10	espacio	azul	azul
11	cabeza culebra	amarillo	rojo
12	cuerpo culebra	amarillo	rojo
13	esquina	amarillo	rojo
14	cola culebra	amarillo	rojo
15	cima escalera	amarillo	rojo
16	medio escalera	amarillo	rojo
17	derecha escalera	amarillo	rojo
18	izquierda escalera	amarillo	rojo
19	pie escalera	amarillo	rojo
20	jugador	magenta	rojo
21	espacio	rojo	rojo
22	punto para dado	amarillo	azul

Observa que cada símbolo tiene que estar disponible con dos colores de fondo de manera que pueda colocarse sobre una y otra clase sobre una y otra clase de cuadros del tablero. En general, si S es el número de un símbolo gráfico con un fondo azul, entonces S+11 es un símbolo con la misma forma pero sobre un fondo rojo. Las tablas X% e Y% se usan para reflejar las coordenadas del motivo jugador (símbolo gráfico 9) para cada jugador. El uso de la tabla VX% será descrito más adelante, junto con el método de hacer cada movimiento. La tabla literal N$ se usa para reflejar los nombres de cada uno de los jugadores; N$(0) contiene el nombre del jugador 0 y N$(1) el del jugador 1.

El procedimiento **prologue** expone las instrucciones para el juego y también pregunta a los jugadores sus nombres y si desean usar un dado real o el simulado por ordenador.

```
8000 DEFine PROCedure prologue
8005 CLS
8006 RANDOMISE
8010 AT 4,14:PRINT 'y'
8020 exhibicione
8030 AT 12,3
8040 PRINT 'Este es un juego para dos jugadores.'
8050 PRINT '  El primer jugador en alcanzar la esquina'
```

```
8060 PRINT '  superior izquierda del tablero es el ganador'
8070 AT 21,2
8080 REPeat picado
8090  INPUT 'Quieres usar un dado real? ';A$
8100  IF A$='S' OR A$='N' THEN EXIT picado
8110 END REPeat picado
8120 IF A$='S' THEN
8130  DADOS=CERTO
8140 ELSE
8150  DADOS=FALSO
8160 END IF
8170 AT 22,3
8180 INPUT 'Dime el nombre del primer jugador ?';N$(0)
8190 INPUT '   Dime el nombre del segundo jugador ?';
     N$(1)
8199 END DEFine prologue
```

Las líneas 8080 a 8160 fijan la variable DADOS según que los jugadores deseen usar un dado real o prefieran que el ordenador genere números aleatorios, en cuyo caso se consideran los dados falsos. Las líneas 8180 y 8190 preguntan los nombres de los jugadores y los imponen como valores de N$(0) y N$(1). La línea 8020 apela al procedimiento **exhibicione** que aprovecha alguno de los símbolos gráficos implantados previamente por **prepa_juego** para generar unos titulares que implican a culebras y escaleras.

```
8200 DEFine PROCedure exhibicione
8210 X1%=1
8220 X2%=16
8230 FOR Y=5 TO 2 STEP -1
8240  porte_grafo X1%,Y,2
8250  porte_grafo X1%+1,Y,1
8260  porte_grafo X1%+3,Y,2
8270  porte_grafo X1%+4,Y,1
8280  porte_grafo X2%,Y,7
8290  porte_grafo X2%+1,Y,
8300  porte_grafo X2%+2,Y,6
8310  porte_grafo X2%+4,Y,7
8320  porte_grafo X2%+5,Y,5
8330  porte_grafo X2%+6,Y,6
8340  X2%=X2%-1
8350  X1%=X1%+1
8355 END FOR Y
8360 porte_grafo 5,1,2
8370 porte_grafo 6,1,0
8380 porte_grafo 8,1,2
8390 porte_grafo 9,1,0
8400 porte_grafo 13,1,4
8410 porte_grafo 14,1,6
8420 porte_grafo 17,1,4
```

```
8430 porte_grafo 18,1,6
8440 porte_grafo 1,6,3
8450 porte_grafo 4,6,3
8460 porte_grafo 17,6,7
8470 porte_grafo 18,6,8
8480 porte_grafo 21,6,7
8490 porte_grafo 22,6,8
8499 END DEFine exhibicione
```

Trazando el decorado

El propósito del procedimiento **decore** es el de exponer el tablero completo con las culebras y escaleras. Lo hace primordialmente recurriendo a otros procedimientos.

```
2000 DEFine PROCedure decore
2010 expo_caques
2020 FOR S=1 TO RND(4 TO 6)
2030  expo_lebra
2040  expo_calera
2050 END FOR S
2099 END DEFine decore
```

El procedimiento **expo_caques** expone la cuadrícula de 'escaques' azules y rojos que constituyen el tablero de juego. Los procedimientos **expo_lebra** y **expo_calera** exponen en pantalla una única culebra y una única escalera respectivamente. El bucle preconfinado, líneas 2020 a 2050, recurre a dichos procedimientos un número de veces que se determina aleatoriamente y comprendido entre 4 y 6.

El procedimiento **expo_caques** es bastante directo:

```
2500 DEFine PROCedure expo_caques
2505 PAPER 6
2510 CLS
2530 FOR Y=2 TO 24 STEP 2
2540  FOR X=2 TO 19 STEP 2
2560   ex_cuatin X,Y,10
2570   ex_cuatin X+1,Y,10
2580   ex_cuatin X,Y+1,10
2590   ex_cuatin X+1,Y+1,10
2600  END FOR X
2610 END FOR Y
2699 END DEFine expo_caques
```

Cada uno de los cuadros que forman el tablero está compuesto de cuatro cuadratínes adyacentes de los que forman la pantalla del QL, y que alternativamente se llenan de azul y rojo recurriendo al proce ·niento **ex_cuatín:**

```
1800 DEFine PROCedure ex_cuatin(X%,Y%,S%)
1810 IF ( (X%+1)/2 MOD 2)=( (Y%+1)/2 MOD 2) THEN
     S%=S%+11
1820 porte_grafo X%,Y%,S%
1830 ringl%(X%,Y%)=S%
1899 END DEFine ex_cuatin
```

Este procedimiento simplemente expone un **cuadratín** pleno de azul o de rojo, dependiendo si su posición en la pantalla corresponde a una fila o columna par o impar. El procedimiento **ex_cuatín** también asigna a los elementos de la tabla **ringl%** el número identificativo (10 ó 21) del símbolo gráfico correspondiente. Es decir, si todos los símbolos que se exponen usando **ex_cuatín,** entonces la tabla **ringl%**(X%,Y%) contendrá el código asignado al símbolo que ocupa la posición X%,Y% de la cuadrícula de pantalla.

El procedimiento **expo_lebra** parece un poco complicado pero sólo es debido al gran número de apelaciones al procedimiento **ex_cuatín** necesario para exponer todos los símbolos que componen el motivo "culebra".

```
3000 DEFine PROCedure expo_lebra
3005 LOCal A%,B%
3010 SAYO=0
3015 REPeat pos
3020  A%=RND(3 TO 8):B%=RND(1 TO 8)
3030  IF A%-1<(10-B%) THEN
3040   L%=RND(2 TO A%-1)+1
3050  ELSE
3060   L%=RND(2 TO 10-B%)+1
3070  END IF
3080  IF SAYO>20 THEN
3090   END DEFine expo_lebra
3100  ELSE
3110   SAYO=SAYO+1
3120  END IF
3130  IF cruzando(A%,B%,L%,-1)=FALSO THEN EXIT pos
3140 END REPeat pos
3150 ex_cuatin A%*2+1,B%*2+1,0
3160 ex_cuatin A%*2,B%*2+1,2
3170 FOR Z=A%-1 TO A%-L%+2 STEP -1
3180  B%=B%+1
3190  ex_cuatin Z*2+2,B%*2,1
3200  ex_cuatin Z*2+1,B%*2,2
3210  ex_cuatin Z*2+1,B%*2+1,1
3220  ex_cuatin Z*2,B%*2+1,2
3230 END FOR Z
3240 B%=B%+1
3250 ex_cuatin (A%-L%+1)*2+2,B%*2,3
3299 END DEFine expo_lebra
```

La línea 3020 genera aleatoriamente dos números que especifican la posición de comienzo de una culebra. Las líneas 3030 a 3070 generan luego otro número aleatorio, reflejado en L%, que determina la longitud de la culebra. La instrucción condicional se asegura que L% no es tan largo que obligue a la culebra a salirse del tablero. La línea 3130 aplica la función **cruzando** para comprobar si la culebra especificada mediante A%, B% y L% cruza cualquier otra culebra o escalera que hubiera en pantalla. Si así es, entonces se genera al azar otro conjunto de números definitorios de culebras y se repite el proceso. De esta manera, se efectúan enSAYOs con diversas posiciones de comienzo y diversas longitudes de culebra hasta que se encuentre un conjunto que encaje en el tablero -es decir, que pueda exponerse sin que caiga encima de cualquier otra culebra o escalera que ya hubiera en el tablero.

Dado que no hay ninguna garantía de encontrar en un lapso razonable de tiempo una culebra que encaje, aprovechamos la variable SAYO para llevar el recuento del número de ensayos hechos. La línea 3080 a 3120 hace que el procedimiento **expo_lebra** se abandone después de 20 ensayos fallidos. Una vez que se haya encontrado una culebra que encaje mediante las líneas 3150 a 3250 se expone en pantalla la combinación de símbolos que le corresponde. El procedimiento **ex_cuatín** es citado en lugar del procedimiento **porte_grafo** directamente, dada la necesidad de elegir el color correcto del fondo para que concuerde con el color ya presente en esa posición de pantalla. Es decir, cuando va a colocarse una parte de culebra sobre un fondo rojo, **ex_cuatín** usará el símbolo gráfico que tiene como fondo el color rojo, y cuando esté colocada sobre un cuadratín azul, mediante ese mismo procedimiento sacaremos un símbolo gráfico sobre un cuadratín azul.

El procedimiento **expo_calera** trabaja aproximadamente de la misma manera que el procedimiento para exponer las culebras.

```
3500 DEFine PROCedure expo_calera
3505 LOCal A%,B%
3510 SAYO=0
3515 REPeat pos
3520  A%=RND(1 TO 6)
3530  B%=RND(1 TO 8)
3540  IF SAYO>20 THEN
3550   END DEFine expo_calera
3560  ELSE
3570   SAYO=SAYO+1
3580  END IF
3590  IF 9-A%<11-B% THEN
3600   L%=RND (2 TO 9-A%)+1
3610  ELSE
3620   L%=RND(2 TO 11-B%)+1
3630  END IF
3640  IF cruzando(A%,B%,L%,1)=FALSO THEN EXIT pos
3650 END REPeat pos
3660 ex_cuatin A%*2+1,B%*2+1,4
3670 ex_cuatin A%*2+2,B%*2+1,6
3680 ex_cuatin A%*2+1,B%*2+2,7
3690 B%=B%+1
3700 FOR Z=A%+1 TO A%+L%-2
3710  ex_cuatin Z*2,B%*2,5
```

```
3720  ex_cuatin Z*2+1,B%*2+1,5
3730  ex_cuatin Z*2+2,B%*2+1,6
3740  ex_cuatin Z*2+1,B%*2+2,7
3750  ex_cuatin Z*2+1,B%*2,6
3760  ex_cuatin Z*2,B%*2+1,7
3770  B%=B%+1
3780 END FOR Z
3790 ex_cuatin (A%+L%-1)*2,B%*2,8
3799 END DEFine expo_calera
```

Las líneas 3515 a 3560 buscan valores para A%, B% y L% que produzcan una escalera que no interfiera con ninguna otra escalera o culebra que ya hubiera expuesta en pantalla. Las líneas 3660 a 3790 sacan la combinación de símbolos que forman el motivo gráfico escalera.

La función **cruzando** definida por el usuario, examina la tabla descriptiva de la situación del tablero para buscar lo que ya está expuesto en pantalla en cada posición concreta, que se vería afectada al exponer encima una escalera o una culebra.

```
9000 DEFine FuNction cruzando(X%,Y%,L%,D%)
9010 LOCal T%,A%,B%
9015 T%=FALSO
9020 E%=2*(X%+D%*L%)+D%
9030 A%=X%*2-D%
9040 B%=Y%*2-1
9050 REPeat sq
9060  FOR Q=-1 TO 1
9070   IF ringl%(A%+Q+1,B%+1)<>10 AND
       ringl%(A%+Q+1,B%+1)<>21 THEN T%=CERTO
9090  END FOR Q
9100  A%=A%+D%
9110  B%=B%+1
9120  IF A%=E% OR T%=CERTO THEN EXIT sq
9130 END REPeat sq
9140 RETurn T%
9199 END DEFine cruzando
```

Las líneas 9020 a 9040 determinan las coordenadas del cuadratín de pantalla que ha de examinarse. Las líneas 9050 a 9130 proceden luego a examinar cada uno de ellos sucesivamente, bien hasta que se haya examinado la última posición, o bien hasta que se haya encontrado un símbolo distinto del 'blanco'. La línea 9070 es la responsable de comprobar realmente que una posición en pantalla contiene un espacio en blanco. Observa que hay dos símbolos gráficos con el carácter de 'blanco', que son el cuadratín azul y el cuadratín rojo, y ambos tienen que comprobarse.

Procedimiento empiece

El procedimiento **empiece** establece los valores iniciales de las tablas que se usarán ulteriormente en el programa y expone los dos motivos gráficos jugadores en sus posiciones iniciales:

```
3800 DEFine PROCedure empiece
3810 FOR I=0 TO 1
3820  X%(I)=1:Y%(I)=12
3840  VX%(I)=1
3850  porte_grafo X%(I)*2-I+1,Y%(I)*2+1,9
3860 END FOR I
3870 YA_META=FALSO
3899 END DEFine empiece
```

Como ya se ha mencionado, X% e Y% se usan para conservar las posiciones de los dos motivos gráficos que representan a los dos jugadores. Aunque cada cuadro del tablero está compuesto de cuatro cuadratínes de pantalla, las posiciones quedan registradas de manera que X%(I%)*2+1,Y%(I%)*2+1 corresponde al cuadratín de la esquina inferior derecha perteneciente al cuadro de tablero situado en la X%(I%)-ésima columna y la Y%(I%)-ésima fila del tablero. La tabla VX% se usa para reflejar la dirección de movimiento de cada motivo gráfico jugador. En cada actualización la posición del jugador I% se cambia para que sea:

```
X%(I%)=X%(I%)+VX%(I%)
```

En todos los juegos anteriores, cualquier motivo gráfico que se movía lo hacía sobre un fondo claro y así en cada ronda dada del bucle de animación podría eliminarse ese motivo usando un 'cuadratín clareador' del color apropiado. Sin embargo, los hombrecillos de este juego se mueven sobre un fondo que consta de una amplia variedad de símbolos gráficos diferentes. Eso complica la estrategia lógica del movimiento un poquito, pero como se explicará en la siguiente sección, el uso de una tabla **ringl%** que remeda las posiciones del tablero, produce un método muy simple de permitir que el hombre se mueva sin destruir la imagen del tablero presente en pantalla.

Movimiento de hombrecillos en cada lance

Una vez que se expone el tablero y los motivos humanos están colocados en su posición inicial, se recurre repetidas veces al procedimiento **lance** para poder moverlos por el tablero:

```
5000 DEFine PROCedure lance(I%)
5010 IF DADOS THEN
5020  emplee_dado
5030 ELSE
5040  simule_dado
```

```
5050 END IF
5060 REPeat avance
5070  ande_paso I%
5080  BEEP .1,-10*I%
5090  PAUSE 2
5100  BEEP
5110  M%=M%-1
5120  IF M%=0 THEN EXIT avance
5130  YA_META=(X%(I%)=1 AND Y%(I%)=1)
5140  IF YA_META THEN EXIT avance
5150 END REPeat avance
5160 IF LEBRA AND CALERA THEN elija
5170 IF LEBRA THEN subaje I%,1,3
5180 IF CALERA THEN subaje I%,-1,4
5190 YA_META=(X%(I%)=1 AND Y%(I%)=1)
5199 END DEFine lance
```

Dependiendo del valor de DADOS, las líneas 5010 a 5050 apelan a uno de los dos procedimientos de dados empleados o simulados; bien sea pidiendo al jugador I% el resultado de su lanzamiento con dado real, o bien sea para generar un número aleatorio que sirva como resultado del lanzamiento simulado. No importa a cuál de los dos procedimiento se recurra, porque el resultado siempre es entregado en M% y luego mediante las líneas 5060 a 5150 se avanza el jugador I% con tantos pasos como indique esa cantidad M%.

Un desplazamiento de un solo cuadro se produce recurriendo a **ande_paso** (línea 5070) y luego en la línea 5130 se comprueba si el avance ha dado como resultado que el jugador pertinente haya llegado a la meta en el cuadro de la esquina superior izquierda. Después de cada paso andado, se fija la variable LEBRA al valor CERTO si el jugador ha caído sobre una cabeza de culebra, y la variable CALERA también al valor CERTO si el cuadro corresponde con el pie de una escalera. Las líneas 5160 a 5180 eligen la tarea pertinente según haya sido el avance del jugador. La línea 5160 resuelve el problema provocado por la caída del jugador sobre un cuadro que contiene a la vez una cabeza de culebra y un pie de escalera, recurriendo al procedimiento **elija**, que aleatoriamente se decidirá por la escalera o por la culebra, pero no por ambas a la vez. El procedimiento **subaje** hace que el hombre o bien se deslice hacia abajo por la culebra, o escale subiendo los peldaños de la escalera (líneas 5170 y 5180).

El procedimiento **ande_paso** tiene tres tareas diferentes por realizar. Tiene que preocuparse de clarear el cuadratín que el jugador ocupaba en su vieja posición, tiene que actualizar las coordenadas teniendo en cuenta lo que sucede cuando el jugador alcanza el borde del tablero, y tiene que comprobar la presencia de una cabeza de culebra o de un pie de escalera dentro del nuevo cuadro al que lleva al jugador.

```
4000 DEFine PROCedure ande_paso(I%)
4010 porte_grafo X%(I%)*2-I%+1,Y%(I%)*2+1,
     ringl%(X%(I%)*2-I%+1,Y%(I%)*2+1)
4020 X%(I%)=X%(I%)+VX%(I%)
4030 IF X%(I%)>9 THEN
4040  X%(I%)=9
4050  Y%(I%)=Y%(I%)-1
4060  VX%(I%)=-VX%(I%)
```

```
4070 END IF
4080 IF X%(I%)<1 THEN
4090  X%(I%)=1
4100  Y%(I%)=Y%(I%)-1
4110  VX%(I%)=-VX%(I%)
4120 END IF
4130 IF ringl%(X%(I%)*2+1,Y%(I%)*2+1)=0 OR
     ringl%(X%(I%)*2+1,Y%(I%)*2+1)=11 THEN
4140  LEBRA=CERTO
4150 ELSE
4160  LEBRA=FALSO
4170 END IF
4180 IF ringl%(X%(I%)*2,Y%(I%)*2)=8 OR
     ringl%(X%(I%)*2,Y%(I%)*2)=19 THEN
4190  CALERA=CERTO
4200 ELSE
4210  CALERA=FALSO
4220 END IF
4230 expo_mbre X%(I%)*2-I%+1,Y%(I%)*2+1
4299 END DEFine ande_paso
```

El 'clareado' del motivo gráfico que representa al jugador en su vieja posición, es llevado a cabo por la línea 4010 que expone el símbolo almacenado en la tabla **ringl%** de nuevo en la posición correcta del tablero. La actualización de las coordenadas del jugador se implementa mediante las líneas 4030 a 4120. Debieras ser capaz de reconocer estas líneas como la realización de una clase de 'rebote' contra el borde de un tablero. Es decir, al alcanzar el borde del tablero el jugador se mueve hacia arriba una fila (i.e. Y%(I%)=Y%(I%)-1) y se invierte el rumbo del movimiento (i.e. VX%(I%)=-VX%(I%)). Finalmente, las líneas 4130 a 4220 comprueban respectivamente si el cuadro está ocupado por una cabeza de culebra y por un pie de escalera, y luego la línea 4230 expone el motivo gráfico jugador en su nueva posición apelando al procedimiento **expo_mbre.**

Dicho procedimiento es muy similar al procedimiento **ex_cuatín** en el aspecto de que elige el motivo gráfico que representa al jugador, que también ocupa un solo cuadratín de pantalla, con el color de fondo correcto de acuerdo con el cuadro del tablero sobre el que va a colocarlo, pero no conserva el número identificativo del símbolo gráfico dentro de la tabla **ringl%.** La razón de esta diferencia debiera ser obvia ya que el propósito de esa tabla es la de conservar el símbolo que está presente en una posición dada del tablero, precisamente antes de exponer sobre esa misma posición el símbolo representativo del jugador.

```
4500 DEFine PROCedure expo_mbre(X%,Y%)
4505 LOCal S%
4506 S%=9
4510 IF ((X%+1)/2 MOD 2)=((Y%+1)/2 MOD 2) THEN
     S%=S%+11
4520 porte_grafo X%,Y%,S%
4599 END DEFine expo_mbre
```

Con eso ya sólo nos quedan unos cuantos procedimientos pequeños y un procedimiento grande, **subaje**, para confeccionar. Antes de pasar al más largo, es mejor quitarse de en medio los más pequeños.

El procedimiento **emplee_dado** pregunta el resultado del lanzamiento real de un dado:

```
5500 DEFine PROCedure emplee_dado
5510 AT 21,2
5520 PRINT 'Es tu turno ';N$(I%);'     '
5530 PRINT '-';N$(I%)
5550 REPeat TIRA_DADOS
5555  AT 22,2
5560  INPUT 'Que sacaste al tirar ';M%
5570  IF M%>0 AND M%<7 THEN EXIT TIRA_DADOS
5580 END REPeat TIRA_DADOS
5599 END DEFine emplee_dado
```

El procedimiento **simule_dado** aplica la función de azar RND para obtener un número aleatorio como resultado del lanzamiento. Sin embargo, en lugar de simplemente exponer el resultado, presenta además un dado animado con los puntos de sus caras cambiando aleatoriamente. Es un intento de crear algo de suspense en el lanzamiento y la espera de ver el resultado de un dado real. Este procedimiento también recurre a unos cuantos procedimientos para exponer las diversas caras del dado:

```
5600 DEFine PROCedure simule_dado
5610 ROL%=RND(5 TO 15)
5620 FOR Q=25 TO 27
5630  FOR R=25 TO 27
5640   porte_grafo Q,R,10
5650  END FOR R
5660 END FOR Q
5670 porte_grafo 26,26,22
5680 M=1
5690 AT 21,3:PRINT 'Es tu turno'
5700 PRINT '  -';N$(I%);'       '
5710 PRINT '   PULSA CUALQUIER TECLA'
5720 PAUSE
5725 porte_grafo 26,26,10
5730 FOR Q=1 TO ROL%
5735 FOR C=22,10
5736 IF C=10 THEN PAUSE Q*2
5737 IF C=10 AND Q=ROL% THEN EXIT Q
5745  C%=C
5750  SELect ON M
5760   ON M=1
5770    sac_uno C%
5780   ON M=2
5790    sac_dos C%
5800   ON M=3
5810    sac_uno C%
5820    sac_dos C%
```

```
5840    ON M=4
5850     sac_dos C%
5860     sac_cuatro C%
5870    ON M=5
5880     sac_uno C%
5890     sac_dos C%
5900     sac_cuatro C%
5910    ON M=6
5920     sac_dos C%
5930     sac_cuatro C%
5940     sac_seis C%
5950   END SELect
5955 END FOR C
5960   M=RND(1 TO 6)
5980 END FOR Q
5990 M%=M
5999 END DEFine simule_dado

6000 DEFine PROCedure sac_uno(C%)
6010 porte_grafo 26,26,C%
6099 END DEFine sac_uno

6100 DEFine PROCedure sac_dos(C%)
6110 porte_grafo 25,25,C%
6120 porte_grafo 27,27,C%
6199 END DEFine sac_dos

6200 DEFine PROCedure sac_cuatro(C%)
6210 porte_grafo 25,27,C%
6220 porte_grafo 27,25,C%
6299 END DEFine sac_cuatro

6300 DEFine PROCedure sac_seis(C%)
6310 porte_grafo 25,26,C%
6320 porte_grafo 27,26,C%
6399 END DEFine sac_seis
```

Observa la manera poco habitual de usar un bucle preconfinado en las líneas 5735 que repite la exposición del dado por dos veces, una usando el símbolo de puntos de la cara del dado (código 22) y otra usando un cuadratín de fondo azul (código 10) para clarear la figura de puntos.

El procedimiento **elija** coloca aleatoriamente al valor falso una u otra de las dos variables LEBRA o CALERA, decidiendo por tanto entre las dos posibilidades:

```
5200 DEFine PROCedure elija
5210 IF RND>.5 THEN
5220   CALERA=FALSO
5230 ELSE
5240   LEBRA=FALSO
5299 END DEFine elija
```

El procedimiento final concerniente al movimiento de los símbolos representativos de los jugadores es **subaje.** Con ese se consigue que el hombre suba por la escalera o baje por la culebra:

```
6500 DEFine PROCedure subaje(I%,D%,C%)
6510 LOCal A%,B%
6520 porte_grafo X%(I%)*2-I%+1,Y%(I%)*2+1,
     ringl%(X%(I%)*2-I%+1,Y%(I%)*2+1)
6530 A%=X%(I%)*2+1
6540 B%=Y%(I%)*2+1
6550 REPeat lizada
6560  expo_mbre A%,B%
6570  BEEP .1,2*B%
6580  PAUSE 2
6590  BEEP
6600  porte_grafo A%,B%,ringl%(A%,B%)
6605 A%=A%-1:B%=B%+D%
6610  IF ringl%(A%,B%)=C% OR
      ringl%(A%,B%)=C%+11 THEN EXIT lizada
6620 END REPeat lizada
6630 A%=A%+1:B%=B%-D%
6640 porte_grafo A%,B%,ringl%(A%,B%)
6650 X%(I%)=A% DIV 2-1
6660 Y%(I%)=B% DIV 2-1
6670 IF D%=1 THEN
6680  Y%(I%)=Y%(I%)+2
6690  X%(I%)=X%(I%)+1
6700 END IF
6705 FOR I=0 TO 1
6710  expo_mbre X%(I)*2-I+1,Y%(I)*2+1
6715 END FOR I
6720 IF (Y%(I%) DIV 2)*2=Y%(I%) THEN
6730  VX%(I%)=1
6740 ELSE
6750  VX%(I%)=-1
6760 END IF
6799 END DEFine subaje
```

La manera en que el procedimiento **subaje** funciona es simplemente exponiendo la figura humana en cada cuadratín de la culebra o de la escalera aprovechando la tabla **ringl%** para clarear la vieja posición del jugador (líneas 6520, 6600 y 6640); y emitiendo los sonidos adecuados durante la 'deslizada' hasta que se encuentra el símbolo que marca el extremo de la culebra o de la escalera (línea 6610). El valor del parámetro D% controla si el hombre se desliza hacia arriba por la escalera (D%=1) o hacia abajo por la culebra (D%=-1), y C% es el número identificativo del símbolo final de una escalera o de una culebra.

Con esto se concluye la descripción de la estrategia lógica de los movimientos, y tengo que admitir que gran parte de ella se desarrolló como resultado de los problemas que surgieron durante las pruebas. Por ejemplo, la situación en que aparecen dentro del mismo cuadro del tablero una cabeza de culebra y el pie de una escalera, no había sido previsto de antemano.

Eso es, de hecho, lo que ocurre muy a menudo cuando se desarrolla un programa en que pueden suceder muchas cosas diferentes de manera aleatoria. Es casi imposible ser consciente de cada evento raro que pueda devenir antes de haber escrito el programa, pero -cuando estés acostumbrado a la **estructura modular**- te será relativamente fácil incluir nuevos procedimientos para que afronten los problemas a medida que brotan.

Procedimiento fine_manga

Este procedimiento simplemente presenta las felicitaciones al jugador que ha llegado ya a la meta y pregunta si desea jugar otra vez:

```
8500 DEFine PROCedure fine_manga
8510 AT 21,2
8520 PRINT 'Tu ganas ';N$(I%);'
8530 REPeat pidotra
8540  INPUT '  Otro juego (S/N)?';A$
8550  IF A$='S' OR A$='N' THEN EXIT pidotra
8560 END REPeat pidotra
8570 IF A$='S' THEN
8580  OTRA=CERTO
8590 ELSE
8600  OTRA=FALSO
8610 END IF
8699 END DEFine fine_manga
```

Evaluación

El programa final funcionó razonablemente rápido y para una pantalla de 32 por 32 cuadratines tiene unos gráficos buenos y efectivos. Todavía hay multitud de cosas por mejorar y montón de memoria libre para aprovecharla. Por ejemplo, el juego parece más atractivo usando personalmente un dado real en lugar del generador de números aleatorios. El problema más grande es que el resultado del juego es enteramente un asunto de azar y aunque es divertido, e incluso interesante durante un rato, no presentan realmente ningún reto ni exige ninguna habilidad para jugarlo. Este problema sólo puede abordarse cambiando bastante la naturaleza del juego. La parte final de este capítulo da las modificaciones necesarias para acercarnos a un juego de esa categoría. Mientras tanto, sin embargo, la versión para ordenador del juego tradicional de "Serpientes y Escalas" está preparado para que juegues.

El listado completo

```
 10 REMark Culebras y     leras
 15 PAPER 1
 20 WINDOW 512,256,0,0
 30 prepa_juego
 40 REPeat manga
 50  prologue
 60  decore
 70  empiece
 80  I%=0
 90  REPeat tirada
100   lance I%
110   IF YA_META THEN EXIT tirada
120   I%=NOT(I%)
130  END REPeat tirada
140  fine_manga
150  IF OTRA=FALSO THEN EXIT manga
160 END REPeat manga
199 STOP

350 DEFine PROCedure prepa_juego
360 DATA -5245,-10109,-7806,-11646
370 DATA -6783,1665,2,0,8769,8316
380 DATA 0,0,9264,18432,8834,-11524,128
390 DATA 22664,-20228,32,26350,28672,20085
400 dire_rus=RESPR(100)
410 RESTORE 360
420 FOR I=dire_rus TO dire_rus+22*2  STEP 2
430 READ B
440  POKE_W I,B
450 END FOR I
460 C_TAB=RESPR(32*23)
465 FOR Q=0 TO 1
470  prepa_grafo 11*Q+0,6,1+Q,0,2,127,102,16,124,252,
     192
480  prepa_grafo 11*Q+1,6,1+Q,3,3,3,3,3,7,254,252
490  prepa_grafo 11*Q+2,6,1+Q,0,0,0,0,0,0,1,3
500  prepa_grafo 11*Q+3,6,1+Q,3,3,31,62,32,0,0,0
510  prepa_grafo 11*Q+4,6,1+Q,0,0,6,3,1,35,54,28
520  prepa_grafo 11*Q+5,6,1+Q,152,60,102,207,217,115,
     54,28
530  prepa_grafo 11*Q+6,6,1+Q,0,0,0,0,128,192,96 ?40
540  prepa_grafo 11*Q+7,6,1+Q,13,7,3,1,0,0,0,0
550  prepa_grafo 11*Q+8,6,1+Q,152,60,102,192,192,?6,
     0,0
560  prepa_grafo 11*Q+9,3,1+Q,24,24,126,24,60,36,36,
     102
570  prepa_grafo 11*Q+10,1+Q,1+Q,255,255,255,255,255,
     255,255,255
580 END FOR Q
590 prepa_grafo 22,6,1,0,0,0,24,24,0,0,0
```

```
600 CERTO=1
610 FALSO=0
620 DIM ringl%(20,26)
630 DIM VX%(1)
640 DIM N$(1,20)
650 DIM X%(1)
660 DIM Y%(1)
699 END DEFine prepa_juego

1300 DEFine PROCedure prepa_grafo(S%,F%,B%,
     R0%,R1%,R2%,R3%,R4%,R5%,R6%,R7%)
1310 LOCal A,FH%,FL%,BH%,BL%
1320 A=C_TAB+32*S%
1330 FH%=(F% DIV 4)*2
1340 FL%=F% && 3
1350 BH%=(B% DIV 4)*2
1360 BL%=B% && 3
1370 prepa_raya(R0%)
1380 prepa_raya(R1%)
1390 prepa_raya(R2%)
1400 prepa_raya(R3%)
1410 prepa_raya(R4%)
1420 prepa_raya(R5%)
1430 prepa_raya(R6%)
1440 prepa_raya(R7%)
1499 END DEFine prepa_grafo

1500 DEFine PROCedure prepa_raya(R%)
1502 LOCal M%,J,I,DL%,DH%
1503 M%=128
1504 FOR J=1 TO 2
1505  DL%=0:DH%=0
1510  FOR I=1 TO 4
1520   IF (R% && M%)=M% THEN
1530    DL%=FH%+DL%*4
1540    DH%=FL%+DH%*4
1550   ELSE
1560    DL%=BH%+DL%*4
1570    DH%=BL%+DH%*4
1580   END IF
1590   M%=M% DIV 2
1600  END FOR I
1610  POKE A,DL%
1620  POKE A+1,DH%
1630  A=A+2
1640 END FOR J
1699 END DEFine prepa_raya

1700 DEFine PROCedure porte_grafo(X%,Y%,S%)
1710 CALL dire_rus,X%,Y%,S%,C_TAB
1799 END DEFine porte_grafo

1800 DEFine PROCedure ex_cuatin(X%,Y%,S%)
```

```
1810 IF ( (X%+1)/2 MOD 2)=( (Y%+1)/2 MOD 2) THEN
     S%=S%+11
1820 porte_grafo X%,Y%,S%
1830 ringl%(X%,Y%)=S%
1899 END DEFine ex_cuatin

2000 DEFine PROCedure decore
2010 expo_caques
2020 FOR S=1 TO RND(4 TO 6)
2030  expo_lebra
2040  expo_calera
2050 END FOR S
2099 END DEFine decore

2500 DEFine PROCedure expo_caques
2505 PAPER 6
2510 CLS
2530 FOR Y=2 TO 24 STEP 2
2540  FOR X=2 TO 19 STEP 2
2560   ex_cuatin X,Y,10
2570   ex_cuatin X+1,Y,10
2580   ex_cuatin X,Y+1,10
2590   ex_cuatin X+1,Y+1,10
2600  END FOR X
2610 END FOR Y
2699 END DEFine expo_caques

3000 DEFine PROCedure expo_lebra
3005 LOCal A%,B%
3010 SAYO=0
3015 REPeat pos
3020  A%=RND(3 TO 8):B%=RND(1 TO 8)
3030  IF A%-1<(10-B%) THEN
3040   L%=RND(2 TO A%-1)+1
3050  ELSE
3060   L%=RND(2 TO 10-B%)+1
3070  END IF
3080  IF SAYO>20 THEN
3090   END DEFine expo_lebra
3100  ELSE
3110   SAYO=SAYO+1
3120  END IF
3130  IF cruzando(A%,B%,L%,-1)=FALSO THEN EXIT pos
3140 END REPeat pos
3150 ex_cuatin A%*2+1,B%*2+1,0
3160 ex_cuatin A%*2,B%*2+1,2
3170 FOR Z=A%-1 TO A%-L%+2 STEP -1
3180  B%=B%+1
3190  ex_cuatin Z*2+2,B%*2,1
3200  ex_cuatin Z*2+1,B%*2,2
3210  ex_cuatin Z*2+1,B%*2+1,1
3220  ex_cuatin Z*2,B%*2+1,2
3230 END FOR Z
```

```
3240 B%=B%+1
3250 ex_cuatin (A%-L%+1)*2+2,B%*2,3
3299 END DEFine expo_lebra

3500 DEFine PROCedure expo_calera
3505 LOCal A%,B%
3510 SAYO=0
3515 REPeat pos
3520  A%=RND(1 TO 6)
3530  B%=RND(1 TO 8)
3540  IF SAYO>20 THEN
3550   END DEFine expo_calera
3560  ELSE
3570   SAYO=SAYO+1
3580  END IF
3590  IF 9-A%<11-B% THEN
3600   L%=RND (2 TO 9-A%)+1
3610  ELSE
3620   L%=RND(2 TO 11-B%)+1
3630  END IF
3640  IF cruzando(A%,B%,L%,1)=FALSO THEN EXIT pos
3650 END REPeat pos
3660 ex_cuatin A%*2+1,B%*2+1,4
3670 ex_cuatin A%*2+2,B%*2+1,6
3680 ex_cuatin A%*2+1,B%*2+2,7
3690 B%=B%+1
3700 FOR Z=A%+1 TO A%+L%-2
3710  ex_cuatin Z*2,B%*2,5
3720  ex_cuatin Z*2+1,B%*2+1,5
3730  ex_cuatin Z*2+2,B%*2+1,6
3740  ex_cuatin Z*2+1,B%*2+2,7
3750  ex_cuatin Z*2+1,B%*2,6
3760  ex_cuatin Z*2,B%*2+1,7
3770  B%=B%+1
3780 END FOR Z
3790 ex_cuatin (A%+L%-1)*2,B%*2,8
3799 END DEFine expo_calera

3800 DEFine PROCedure empiece
3810 FOR I=0 TO 1
3820  X%(I)=1:Y%(I)=12
3840  VX%(I)=1
3850  porte_grafo X%(I)*2-I+1,Y%(I)*2+1,9
3860 END FOR I
3870 YA_META=FALSO
3899 END DEFine empiece

4000 DEFine PROCedure ande_paso(I%)
4010 porte_grafo X%(I%)*2-I%+1,Y%(I%)*2+1,
     ring1%(X%(I%)*2-I%+1,Y%(I%)*2+1)
4020 X%(I%)=X%(I%)+VX%(I%)
4030 IF X%(I%)>9 THEN
4040  X%(I%)=9
```

```
4050  Y%(I%)=Y%(I%)-1
4060  VX%(I%)=-VX%(I%)
4070 END IF
4080 IF X%(I%)<1 THEN
4090  X%(I%)=1
4100  Y%(I%)=Y%(I%)-1
4110  VX%(I%)=-VX%(I%)
4120 END IF
4130 IF ringl%(X%(I%)*2+1,Y%(I%)*2+1)=0 OR
     ringl%(X%(I%)*2+1,Y%(I%)*2+1)=11 THEN
4140  LEBRA=CERTO
4150 ELSE
4160  LEBRA=FALSO
4170 END IF
4180 IF ringl%(X%(I%)*2,Y%(I%)*2)=8 OR
     ringl%(X%(I%)*2,Y%(I%)*2)=19 THEN
4190  CALERA=CERTO
4200 ELSE
4210  CALERA=FALSO
4220 END IF
4230 expo_mbre X%(I%)*2-I%+1,Y%(I%)*2+1
4299 END DEFine ande_paso

4500 DEFine PROCedure expo_mbre(X%,Y%)
4505 LOCal S%
4506 S%=9
4510 IF ((X%+1)/2 MOD 2)=((Y%+1)/2 MOD 2) THEN
     S%=S%+11
4520 porte_grafo X%,Y%,S%
4599 END DEFine expo_mbre

5000 DEFine PROCedure lance(I%)
5010 IF DADOS THEN
5020  emplee_dado
5030 ELSE
5040  simule_dado
5050 END IF
5060 REPeat avance
5070  ande_paso I%
5080  BEEP .1,-10*I%
5090  PAUSE 2
5100  BEEP
5110  M%=M%-1
5120  IF M%=0 THEN EXIT avance
5130  YA_META=(X%(I%)=1 AND Y%(I%)=1)
5140  IF YA_META THEN EXIT avance
5150 END REPeat avance
5160 IF LEBRA AND CALERA THEN elija
5170 IF LEBRA THEN subaje I%,1,3
5180 IF CALERA THEN subaje I%,-1,4
5190 YA_META=(X%(I%)=1 AND Y%(I%)=1)
5199 END DEFine lance
```

```
5200 DEFine PROCedure elija
5210 IF RND>.5 THEN
5220  CALERA=FALSO
5230 ELSE
5240  LEBRA=FALSO
5299 END DEFine elija

5500 DEFine PROCedure emple_dado
5510 AT 21,2
5520 PRINT 'Es tu turno ';N$(I%);'     '
5530 PRINT '-';N$(I%)
5550 REPeat TIRA_DADOS
5555  AT 22,2
5560  INPUT 'Que sacaste al tirar ';M%
5570  IF M%>0 AND M%<7 THEN EXIT TIRA_DADOS
5580 END REPeat TIRA_DADOS
5599 END DEFine emple_dado

5600 DEFine PROCedure simule_dado
5610 ROL%=RND(5 TO 15)
5620 FOR Q=25 TO 27
5630  FOR R=25 TO 27
5640   porte_grafo Q,R,10
5650  END FOR R
5660 END FOR Q
5670 porte_grafo 26,26,22
5680 M=1
5690 AT 21,3:PRINT 'Es tu turno'
5700 PRINT '  -';N$(I%);'         '
5710 PRINT '  PULSA CUALQUIER TECLA'
5720 PAUSE
5725 porte_grafo 26,26,10
5730 FOR Q=1 TO ROL%
5735 FOR C=22,10
5736 IF C=10 THEN PAUSE Q*2
5737 IF C=10 AND Q=ROL% THEN EXIT Q
5745  C%=C
5750  SELect ON M
5760   ON M=1
5770    sac_uno C%
5780   ON M=2
5790    sac_dos C%
5800   ON M=3
5810    sac_uno C%
5820    sac_dos C%
5840   ON M=4
5850    sac_dos C%
5860    sac_cuatro C%
5870   ON M=5
5880    sac_uno C%
5890    sac_dos C%
5900    sac_cuatro C%
5910   ON M=6
```

```
5920     sac_dos C%
5930     sac_cuatro C%
5940     sac_seis C%
5950   END SELect
5955 END FOR C
5960   M=RND(1 TO 6)
5980 END FOR Q
5990 M%=M
5999 END DEFine simule_dado

6000 DEFine PROCedure sac_uno(C%)
6010 porte_grafo 26,26,C%
6099 END DEFine sac_uno

6100 DEFine PROCedure sac_dos(C%)
6110 porte_grafo 25,25,C%
6120 porte_grafo 27,27,C%
6199 END DEFine sac_dos

6200 DEFine PROCedure sac_cuatro(C%)
6210 porte_grafo 25,27,C%
6220 porte_grafo 27,25,C%
6299 END DEFine sac_cuatro

6300 DEFine PROCedure sac_seis(C%)
6310 porte_grafo 25,26,C%
6320 porte_grafo 27,26,C%
6399 END DEFine sac_seis

6500 DEFine PROCedure subaje(I%,D%,C%)
6510 LOCal A%,B%
6520 porte_grafo X%(I%)*2-I%+1,Y%(I%)*2+1,
     ring1%(X%(I%)*2-I%+1,Y%(I%)*2+1)
6530 A%=X%(I%)*2+1
6540 B%=Y%(I%)*2+1
6550 REPeat lizada
6560   expo_mbre A%,B%
6570   BEEP .1,2*B%
6580   PAUSE 2
6590   BEEP
6600   porte_grafo A%,B%,ring1%(A%,B%)
6605 A%=A%-1:B%=B%+D%
6610   IF ring1%(A%,B%)=C% OR
       ring1%(A%,B%)=C%+11 THEN EXIT lizada
6620 END REPeat lizada
6630 A%=A%+1:B%=B%-D%
6640 porte_grafo A%,B%,ring1%(A%,B%)
6650 X%(I%)=A% DIV 2-1
6660 Y%(I%)=B% DIV 2-1
6670 IF D%=1 THEN
6680   Y%(I%)=Y%(I%)+2
6690   X%(I%)=X%(I%)+1
6700 END IF
```

```
6705 FOR I=0 TO 1
6710  expo_mbre X%(I)*2-I+1,Y%(I)*2+1
6715 END FOR I
6720 IF (Y%(I%) DIV 2)*2=Y%(I%) THEN
6730  VX%(I%)=1
6740 ELSE
6750  VX%(I%)=-1
6760 END IF
6799 END DEFine subaje

8000 DEFine PROCedure prologue
8005 CLS
8006 RANDOMISE
8010 AT 4,14:PRINT 'y'
8020 exhibicione
8030 AT 12,3
8040 PRINT 'Este es un juego para dos jugadores.'
8050 PRINT '  El primer jugador en alcanzar la esquina'
8060 PRINT '  superior izquierda del tablero es el ganador'
8070 AT 21,2
8080 REPeat picado
8090  INPUT 'Quieres usar un dado real?';A$
8100  IF A$='S' OR A$='N' THEN EXIT picado
8110 END REPeat picado
8120 IF A$='S' THEN
8130  DADOS=CERTO
8140 ELSE
8150  DADOS=FALSO
8160 END IF
8170 AT 22,3
8180 INPUT 'Dime el nombre del primer jugador ?';N$(0)
8190 INPUT '    Dime el nombre del segundo jugador ?';
     N$(1)
8199 END DEFine prologue

8200 DEFine PROCedure exhibicione
8210 X1%=1
8220 X2%=16
8230 FOR Y=5 TO 2 STEP -1
8240  porte_grafo X1%,Y,2
8250  porte_grafo X1%+1,Y,1
8260  porte_grafo X1%+3,Y,2
8270  porte_grafo X1%+4,Y,1
8280  porte_grafo X2%,Y,7
8290  porte_grafo X2%+1,Y,
8300  porte_grafo X2%+2,Y,6
8310  porte_grafo X2%+4,Y,7
8320  porte_grafo X2%+5,Y,5
8330  porte_grafo X2%+6,Y,6
8340  X2%=X2%-1
8350  X1%=X1%+1
8355 END FOR Y
8360 porte_grafo 5,1,2
```

```
8370 porte_grafo 6,1,0
8380 porte_grafo 8,1,2
8390 porte_grafo 9,1,0
8400 porte_grafo 13,1,4
8410 porte_grafo 14,1,6
8420 porte_grafo 17,1,4
8430 porte_grafo 18,1,6
8440 porte_grafo 1,6,3
8450 porte_grafo 4,6,3
8460 porte_grafo 17,6,7
8470 porte_grafo 18,6,8
8480 porte_grafo 21,6,7
8490 porte_grafo 22,6,8
8499 END DEFine exhibicione

8500 DEFine PROCedure fine_manga
8510 AT 21,2
8520 PRINT 'Tu ganas ';N$(I%);'          '
8530 REPeat pidotra
8540  INPUT '  Otro juego (S/N)?';A$
8550  IF A$='S' OR A$='N' THEN EXIT pidotra
8560 END REPeat pidotra
8570 IF A$='S' THEN
8580  OTRA=CERTO
8590 ELSE
8600  OTRA=FALSO
8610 END IF
8699 END DEFine fine_manga

9000 DEFine FuNction cruzando(X%,Y%,L%,D%)
9010 LOCal T%,A%,B%
9015 T%=FALSO
9020 E%=2*(X%+D%*L%)+D%
9030 A%=X%*2-D%
9040 B%=Y%*2-1
9050 REPeat sq
9060  FOR Q=-1 TO 1
9070   IF ring1%(A%+Q+1,B%+1)<>10 AND
       ring1%(A%+Q+1,B%+1)<>21 THEN T%=CERTO
9090  END FOR Q
9100  A%=A%+D%
9110  B%=B%+1
9120  IF A%=E% OR T%=CERTO THEN EXIT sq
9130 END REPeat sq
9140 RETurn T%
9199 END DEFine cruzando
```

Serpientes y Escalas en acción

La forma básica del juego lleva por sí mismo a una conversión en un juego con motivos gráficos realmente dinámicos. La idea es que en lugar de mover alternativamente cada jugador en una cantidad de pasos aleatoria, una única figura humana sea continuamente animada, moviendo un cuadro cada vez que pase a través del bucle de animación. Desde luego, en este esquema el hombre automáticamente se deslizará hacia abajo con cada culebra y hacia arriba con cada escalera que encuentre a lo largo de su trayecto. Algunas veces, la disposición de escaleras y culebras puede ser tal que el hombre alcance la posición final y otras veces se mantendrá dando vueltas sin cesar subiendo las mismas escaleras y bajando por las mismas culebras. Para hacer que esto se convierta en un juego de habilidad, todo lo que se necesita es añadir la condición extra de obligar a que el jugador pulse la tecla L cuando desee subir por una escalera exactamente al encontrarse en el pie de ella, y evitar que se deslice hacia abajo por una culebra pulsando la tecla S justamente cuando la figura llega a la cabeza de la culebra. Manteniendo pulsadas una u otra tecla en cualquier momento, hace que el motivo gráfico cese de moverse y, ya que la meta del juego es llegar a la esquina superior izquierda del tablero lo más rápidamente posible, eso no sería una buena manera de jugar.

Las modificaciones al programa vistas hasta este momento son bastante simples. El programa principal se convierte en:

```
 10 REMark culebras y escaleras
 15 PAPER 1
 20 WINDOW 512,256,0,0
 30 prepa_juego
 40 REPeat manga
 50  prologue
 60  decore
 70  empiece
 90  REPeat tirada
100   lance 1
110   IF YA_META THEN EXIT tirada
130  END REPeat tirada
140  fine_manga
150  IF OTRA=FALSO THEN EXIT manga
160 END REPeat manga
199 STOP
```

El procedimiento **empiece** tiene que modificarse para que exponga sólo un motivo gráfico que represente al jugador y para poner a cero el cronómetro. Eso implica cambiar la línea 3810 y añadir la línea 3880 en la forma:

```
3810 FOR I=1 TO 1

3880 SDATE 2000,1,1,0,0,0
```

También ha de cambiarse una línea en el procedimiento **subaje,** en la forma:

```
6705 FOR I=1 TO 1
```

Las modificaciones y añadidos al procedimiento **lance** son suficientes como para que merezca la pena dar el listado completo de la nueva versión.

```
5000 DEFine PROCedure lance(I%)
5010 expo_ra
5020 ande_paso I%
5030 BEEP .1,-10*I%
5040 PAUSE 2
5050 BEEP
5060 YA_META=(X%(I%)=1 AND Y%(I%)=1)
5070 REPeat indago
5080 expo_ra
5090 IF NOT(pulsada) OR LEBRA OR CALERA THEN
     EXIT indago
5100 END REPeat indago
5110 IF LEBRA AND KEYROW(3)<>8 THEN subaje I%,1,3
5120 IF CALERA AND KEYROW(4)=1 THEN subaje I%,-1,4
5130 YA_META=(X%(I%)=1 AND Y%(I%)=1)
5199 END DEFine lance
```

Ahora, el procedimiento **lance** sólo recurre al procedimiento **ande_paso** una vez en cada ronda del bucle de animación y apela al procedimiento **expo_ra** para exponer la hora corriente del cronómetro. Las únicas otras modificaciones son el añadido de las líneas 5070 a 5100 que indagan si se ha pulsado la tecla 'L' o la 'S' para detener la animación cuando el hombre no está situado en la cabeza de una culebra o en el pie de una escalera; y los cambios en las líneas 5110 y 5120 que dan como resultado que el hombre se deslice hacia abajo por una culebra si no está pulsada la tecla 'S', y que suba por la escalera sólo si está pulsada la tecla 'L'. Observa que debido a los cambios en los números de línea es mejor teclear este procedimiento completamente de nuevo en lugar de intentar modificar la versión existente de **lance.**

El procedimiento **expo_ra** es nuevo pero muy corto, ya que sólo expone la hora en la parte inferior de la pantalla:

```
4600 DEFine PROCedure expo_ra
4610 time$=DATE$
4620 seg=time$(17)*60
4630 seg=seg+time$(19 TO 20)
4635 AT 22,5
4650 PRINT 'TIEMPO';seg
4699 END DEFine expo_ra
```

Los cambios para el procedimiento **prologue** son muy simples pero amplios y eso hace que merezca la pena listar la nueva versión completa del procedimiento.

```
8000 DEFine PROCedure prologue
8005 CLS
8006 RANDOMISE
8010 AT 4,14:PRINT 'y'
8020 exhibicione
8030 AT 12,3
8040 PRINT 'Debes intentar llevar tu hombrecillo a'
8050 PRINT '   la esquina superior izquierda del'
8060 PRINT '   tablero lo mas rapidamente posible,'
8062 PRINT '   debes evitar las cabezas de las'
8064 PRINT '   culebras pulsando S y saltar las '
8066 PRINT '   escaleras pulsando L'
8070 AT 21,2
8080 PRINT 'Pulsa cualquier tecla para comenzar'
8090 PAUSE
8199 END DEFine prologue
```

Lo mismo es cierto para el procedimiento **fine_manga:**

```
8500 DEFine PROCedure fine_manga
8510 AT 22,2
8520 PRINT 'tardaste ';seg;' segundos'
8530 REPeat pidotra
8540  INPUT 'Otro juego (S/N)?';A$
8550  IF A$='S' OR A$='N' THEN EXIT pidotra
8560 END REPeat pidotra
8570 IF A$='S' THEN
8580  OTRA=CERTO
8590 ELSE
8600  OTRA=FALSO
8610 END IF
8699 END DEFine fine_manga
```

Con estos cambios en el programa primitivo te encontrarás con que tienes un juego rápido y muy excitante. El hombrecillo se mueve alrededor del tablero, baja por las culebras y sube por las escaleras de una manera fascinante. Hay un montón de mejoras que pueden hacerse a este rápido juego de serpientes y escalas siguiendo las mismas directrices para posibilidades como las incluidas en juegos anteriores. Por ejemplo, la rutina de final de juego puede emitir los mensajes apropiados; puedes obligar a un límite de tiempo; producir una tabla con los mejores tiempos; mejorar los efectos sonoros... Puedes incluso introducir nuevos elementos en el juego tales como "agujeros negros" sobre los que hay que saltar, etc. La lección que ha de aprenderse es que hay todavía un gran trabajo que puede desarrollarse con los juegos tradicionales para actualizarlos a los tiempos modernos y algunas veces los resultados pueden ser muy impresionantes.

Capítulo Ocho

La Conversión en Programador Magistral

La graduación desde ser un programador novato hasta convertirse en un maestro de esta artesanía es un tema de desarrollo de las habilidades que ya has adquirido y de consecución de confianza para aplicarlas a proyectos más y más ambiciosos. Si has tecleado y usado los programas en los capítulos primeros, entonces debieras comenzar a apreciar algunas de las maneras en que se escriben los programas largos. Sin embargo, para conseguir realmente el máximo provecho de lo que has aprendido es esencial que obtengas experiencia práctica, primero modificando y ampliando los juegos dados en este libro y luego implementando tus propias ideas. No puedes llegar a ser un buen programador sin **hacer el esfuerzo** de escribir algunos programas y aprender de tus propias equivocaciones y éxitos. En este capítulo final echamos una mirada a algunos de los métodos y actitudes que debieras ensayar y acostumbrarte a llevar en tu propio trabajo.

El Capítulo Uno reforzó la necesidad de usar un **método de programación** y todos los programas en este libro han usado una forma de **programación estructurada** combinada con el refinamiento gradual, lo que se describe con mayor profundidad en el libro "The Complete Programmer" por Mike James (Granada, 1983) -pero eso es sólo parte de la historia. Podrías decir que la programación estructurada y el desglose gradual son la 'cara científica' en el arte de programar. Sin aplicarlas, la programación es un trabajo duro. Sin embargo, incluso con el uso de un método de programación, todavía queda una gran cantidad de 'arte' dentro de la buena programación y eso sólo puede aprenderse mediante la práctica.

Eso no quiere decir que la buena programación sea la programación **artesana** en el sentido de descansar en trucos y métodos imaginativos. Un buen programa debe ser tan claro como sea posible y los trucos y los métodos inteligentes tienden a aumentar la confusión dentro del programa -y un programa confuso es un programa propenso a pifias. Como ya he mencionado, la única ruta para convertirse en un programador magistral es aprender de tus propias experiencias al escribir programas, pero para hacer eso debes primero conocer cómo evaluarlos y cómo tienes que acabar los programas que has comenzado.

Juegos y solución de problemas

Si repasas los primeros capítulos te encontrarás que la mayoría de los programas comenzaron a partir de una sencilla idea que, **implementada en BASIC**, no producía un juego satisfactorio. La verdad es que la mayoría de las ideas para los juegos tienden a ser frustrantes cuando se implementan por primera vez, pero eso no es ninguna razón para abandonar la idea. El éxito de cualquier juego para ordenador depende de lo bien que esté implementado e incluso enormes buenas ideas que estén malamente implementadas, dan como resultado juegos horribles. Por ejemplo, uno de los juegos para ordenador más viejos y más atractivo es el de los 'invasores espaciales', pero es posible hacer que incluso este juego sea totalmente **infumable** si no se presta atención al detalle. Si el juego marcha demasiado lentamente o la respuesta a las pulsaciones de las teclas es muy perezosa, será frustrante en lugar de excitante.

Si un juego no resulta de acuerdo con tus esperanzas, no desprecies inmediatamente la idea y busca por algo nuevo. Hazte a ti mismo la pregunta ¿qué está mal? Algunas veces la respuesta será sencilla de descubrir y no difícil de corregir; por ejemplo, si el juego funciona demasiado lentamente, cambiando algunas de las rutinas y pasándolas a lenguaje máquina consigue el objeto. Por otro lado, la razón de las respuestas lentas a las pulsaciones del teclado puede que algunas veces sean difíciles de descubrir el fallo. Pueden deberse al intervalo de tiempo que el bucle de animación tarda en ejecutarse, pero pueden igualmente ser debidas a que no se inspecciona el teclado lo suficientemente a menudo cada vez que se pasa por el bucle. Por ejemplo, si el bucle de animación va demasiado rápido, entonces la manera más obvia de hacerlo más lento es insertar un **bucle de demora.** Si el tiempo empleado en el bucle de demora es una porción significativa del tiempo que se tarda en el bucle de animación, el teclado será examinado solamente durante un corto período cada vez que se da una ronda del bucle. La solución a este problema no es acelerar el bucle de animación, sino examinar el teclado más de una vez en cada pasada a través del bucle.

Algunas veces, la razón de por qué el primer intento de un juego no tiene éxito, no radica en su implementación. Por ejemplo, en el Capítulo Dos la primera versión del Hormiguero no era muy divertida de jugar como debiera haber sido, porque era demasiado fácil. Si un juego es demasiado fácil, la tendencia a menudo es hacerlo más difícil de las maneras más obvias. Por ejemplo, puedes hacer cualquier juego más difícil si fijas un límite de tiempo más corto, pero a no ser que el tiempo que se tarde en completar el juego realmente dependa de la habilidad del jugador, la prefijación de un límite de tiempo corto o bien no tiene ningún efecto, o bien hace que el juego sea imposible. Lo que tienes que hacer es examinar el juego cuidadosamente y descubrir qué hay en él que lo hace fácil o difícil. En el caso de los nidos de huevos de las hormigas el problema se rastreó hasta encontrar que las hormigas no constituían demasiado problema para el hombre que intentaba llegar al nidal del hormiguero. En ese caso particular, la solución fue hacer que la dispersión de las hormigas al comienzo del programa fuera más aleatoria y además confinarlas a un área más pequeña. Si después de examinar el juego la razón de que sea demasiado fácil o demasiado difícil, resulta que es algo virtualmente imposible de corregir, entonces no tienes otra elección, sino la la de abandonar y ensayar alguna otra cosa -pero eso es un caso muy raro.

Lo cierto es que muy pocos juegos, o cualquier otra clase de programa para este tema son tan fáciles que la primera versión que produces satisfaga tus esperanzas -y eso no es una buena razón para abandonar y comenzar otra cosa distinta. Si quieres mejorar tu programación, debes hacer el esfuerzo de convertir **lo que tienes** en forma de un programa en **lo que deseas,** y ese es un proceso analítico de rastreo y solución de cada pequeño problema que hace que tu programa acabe cumpliendo sus especificaciones.

Finalización

Una vez que tengas un juego divertido para jugar, el siguiente paso es añadir los procedimientos necesarios para hacerlo **jugable.** En otras palabras, tienes que hacer un juego adecuado para ser usado por otras personas, y la única manera de descubrir si has tenido éxito es vigilar a las otras personas que lo juegan. No es de recibo reclamar que los jugadores hacen que tu juego **se la pegue** porque ellos no lo comprenden bien, y por tanto el fallo es suyo; si ellos no lo comprenden, no les dijiste lo suficiente sobre él o es demasiado complicado y en ambos casos un programa bien escrito nunca debe pegársela.

Todos los juegos en este libro han sido **comprobados ante fallos** en el sentido de ser imposible introducir valores o controlar los juegos de manera que le obliguen a parar de funcionar y haga que el jugador caiga de nuevo en las manos de BASIC. El método principal para estas comprobaciones se basa en la vieja y bien conocida idea de "basura adentro, basura afuera" (a menudo abreviada a BIBO, y en inglés GIGO). Si puedes evitar que la basura se introduzca en tu programa estarás razonablemente seguro de que no se interrumpirá la ejecución. Por ejemplo, si pides al jugador que ingrese el nivel de dificultad, debes siempre comprobar que está dentro de la banda correcta antes de pasar al resto del programa. Sin embargo, incluso aunque compruebes cada valor introducido por teclado todavía es posible que un programa se interrumpa o funcione mal a causa de una combinación de valores para los que no estaba preparado. Por ejemplo, en las primeras etapas de prueba de las serpientes y escalas se generó un tablero que incluía un cuadro que contenía a la vez la cabeza de una culebra y el pie de una escalera. El resultado fue que el motivo gráfico que representaba al hombre subía a la vez la escalera y bajaba por la culebra, con consecuencias obvias y desastrosas para el resto del programa.

Para asegurarte que tu programa no se interrumpe debido a circunstancias inesperadas debes comprobarlo a conciencia. La teoría de la **comprobación** y **depuración** de un programa se cubre con detalle en el libro "The Complete Programmer", pero merece la pena decir aquí que jugar durante algún tiempo con un juego no es lo mismo que probarlo. Si juegas lo que tú estás desarrollando hay gran oportunidad de llegar a un momento en que estás cercano a la versión casi definitiva y eres bastante bueno jugándolo. Eso significa que estás probando el juego con un nivel concreto de habilidad y el problema que produce el fallo en el programa puede surgir con un nivel más bajo de habilidad. Por ejemplo, supongamos que estás probando una primera versión de las Ranas Saltarinas (Capítulo Tres) y luego puedes jugar muchas mangas sucesivas sólo para descubrir que tan pronto como un principiante comienza a jugar, el programa deja de funcionar porque el jugador falló en el rebote de cualquiera de las diez ranas y su puntuación fue por tanto cero.

Cuando pruebas un juego debes ensayar todas las posibilidades extremas, puntuaciones altas y puntuaciones bajas y jugarlo de la manera más boba posible que puedas pensar - en suma, **intentar que se corte.** Si no puedes conseguirlo cuando lo estás intentando lo más duramente posible, entonces con suerte, cualquier problema que contenga será pequeño.

Hay un problema particular asociado con la comprobación de programas de juegos que generalmente no ocurre con otras clases de programas y es debido a la **aleatoriedad.** Por ejemplo, el tablero expuesto como parte del juego de Serpientes y Escalas se genera aleatoriamente y puede que tengas que esperar un montón de tiempo antes de que un tablero problemático (v.g. uno que contenga dentro del mismo cuadro la cabeza de culebra y el pie de escalera) sea generado.

Hay dos maneras de abordar el problema de probar secciones del programa con aleatoriedad. Primeramente, puedes cambiar cualquier instrucción que contenga funciones de azar RND dentro del programa fijando las variables a valores conocidos que puedan causar problemas. La cuestión con este método es que tienes que prever los valores que son propicios para provocar los problemas, y en general, si puedes conseguirlo es que el problema es tan obvio que incluso no necesitarás poner en marcha el programa. Por ejemplo, si detectas el hecho que algunos valores de los números aleatorios en los procedimientos que exponen una culebra y exponen una escalera en el programa mencionado, pueden dar como resultado que se exponga la cabeza de la culebra y el pie de la escalera dentro del mismo cuadro del tablero, entonces es obvio que eso va a causar problemas.

El segundo método de probar la componente aleatoria de un juego es simplemente el de examinar una gran cantidad de ejemplos. Ese es uno de los métodos usados para comprobar el programa de culebras y escaleras durante su desarrollo. Para hacer posible ver una gran cantidad de ejemplos en un intervalo razonable de tiempo, se cambió el programa principal para que fuera un bucle que repetidamente recurría a los procedimientos que exponen el tablero una y otra vez. Una vez que se ha localizado el problema de esta manera es importante asegurarse que puede hacerse que suceda **a voluntad** antes de intentar corregir la pifia. En el caso de Culebras y Escaleras, una vez que se hubo generado con una cabeza de culebra y un pie de escalera dentro del mismo cuadro, las funciones RND se sustituyeron por constantes que producían esa condición. Es decir, se modificó el programa de manera que cada vez que se ponía en marcha producía un tablero problemático. Esa es la única manera en que puede comprobarse la eficacia de una corrección de una pifia en un programa que usa aleatoriedad. Por ejemplo, imagínate que después de unas cuantas horas de prueba, brota un problema que hace que el programa se la pegue; si la pifia se corrige inmediatamente, entonces tendrás que esperar otras cuantas horas para decir si esa corrección funciona o no. Por otro lado, si encuentras primero una manera de modificar el programa de manera que el problema surja cada vez que ejecutas el programa, puedes asegurarte que tu corrección del percance funciona sin tener que comprobarlo durante horas.

La **prueba** y **depuración** de programas son áreas que ciertamente requieren una cantidad de conocimientos de programación. Sin embargo, producir programas **amistosos al usuario** es igual de importante y requiere también habilidad completa. En algunos aspectos es la parte de la programación que demanda más habilidad porque es casi imposible decir lo que hace que un programa sea cómodo para el usuario. La única manera de descubrir lo que es bueno en este aspecto es observar la forma en que los usuarios reaccionan ante tu programa.

Cuando estás diseñando tu programa, debes intentar pensar en la forma en que será usado y así encontrar el **orden natural** para el curso del programa. Sin embargo, incluso si produces un programa que **tú mismo** encuentras fácil de usar, tienes que recordar permanentemente que tú no eres un usuario típico. Los programadores tendemos a olvidar lo difícil y confuso que son el teclado y la pantalla del ordenador para los no programadores y que ellos ciertamente conocen su propio programa mejor que cualquier otro. Como resultado de eso, algo que parece fácil u obvio a ti como programador, puede parecer incómodo y oscuro a un usuario que no programa. Todo programador ha tenido la experiencia de observar a alguien más usar su programa en una manera tan idiota que le causa desesperación pensar en la estupidez aparente del usuario. Esa es, desde luego, la actitud errónea -los usuarios siempre tienen una preferencia razonable y si encuentran que tu programa es difícil de usar, debes tratar eso como un problema de tu programa, no un problema del usuario.

La variedad de juegos

Si estás inspirado para poner tu mano en la invención y confección de tus propios programas para juegos, puede que estés interesado en saber algo sobre las clases de juego que actualmente son populares.

Los juegos de animación con motivos gráficos forman el más amplio y más popular grupo de juegos. Con la posible excepción de Escaleras y Culebras, todos los juegos de este libro se basan en la animación. La faceta primordial de todos esos juegos es que el jugador tiene control sobre el movimiento de algún objeto en la pantalla. Aunque tú puedas crear nuevos juegos de animación pensando sobre temas y objetivos nuevos, la única diferencia real entre los juegos radica en la manera en que el jugador puede controlar el objeto movible. Por ejemplo, puedes construir un nuevo juego a partir del hormiguero si cambias todas las hormigas por fantasmas y el nidal de huevos por un cofre del tesoro, pero el juego todavía continuará siendo similar al de hormiguero. Sin embargo, la culebra vista en el Capítulo Cinco es muy diferente del Hormiguero en la habilidad que el jugador necesita tener para controlar el motivo gráfico. Si quieres crear un nuevo juego de animación, necesitas considerar no sólo el tema del juego, sino también cómo el jugador controlará los motivos gráficos y qué nuevo elemento de habilidad se requerirá.

El segundo grupo más popular de juegos son los **juegos de aventuras.** Estos algunas veces contienen juegos de animación sencillos dentro de ellos pero sólo como parte del objetivo global del juego. Un juego de aventuras no radica en la animación de los motivos para comprobar la habilidad del jugador; en lugar de eso crea un mundo simulado habitado por una diversidad de criaturas, algunas amigas y otras hostiles. El mundo y sus criaturas en la mayoría de los casos se crean usando descripciones y gráficos limitados, y el jugador generalmente interactúa en ese mundo mediante comandos introducidos por el teclado. Los juegos de aventura generalmente presentan menos problemas de programación que los juegos de animación, pero lo que sí necesitan es una gran cantidad de imaginación.

La tercera categoría de juegos, **juegos de estrategia,** incluyen el ajedrez por ordenador, las damas, tres en raya, Otelo, etc. Estos juegos de estrategia pueden algunas veces involucrar gráficos complicados (considera por ejemplo los problemas de exponer un tablero de ajedrez completo con sus piezas), pero el principal reto en programación que presenta es hacer que el ordenador juegue de una manera convincente. Intentar desarrollar cómo ha de programarse un ordenador para jugar un juego como el ajedrez, te llevará a las fronteras de la ciencia informática y de la **inteligencia artificial.**

Cuando estás trabajando sobre nuevos juegos siempre hay el elemento del descubrimiento por sorpresa. Es admirable cuán a menudo al implementar un juego, inmediatamente surge la idea para otro que usa los mismos ingredientes pero de diversas maneras. Esa es también otra razón para escribir siempre los programas usando **procedimientos,** o secciones separadas y concretas para realizar una tarea. Si se ha escrito un juego usando procedimientos, entonces esos procedimientos pueden a menudo volver a ser utilizados en la construcción de otros juegos, y en este sentido, cuantos más juegos escribas, más fácil te irán resultando.

Uso del QL

Puede que estés sorprendido que las excelentes posibilidades para gráficos de alta resolución disponibles en el QL fueran largamente ignoradas en los juegos presentados en este libro. Eso es sin embargo, fácil de explicar ya que el uso primordial de los gráficos de alta resolución en los juegos es para crear escenas de fondo complicadas. La mayoría de los juegos de animación son mucho más fáciles de implementar usando **símbolos definibles por el usuario,** cosa que no está inicialmente provista en el SuperBASIC del QL.

De manera que que, mientras que la gama standard de facilidades para gráficos del QL está excelentemente adecuada para su uso en aplicaciones comerciales y técnicas, no es capaz de producir inmediatamente los efectos requeridos para los juegos animados. Sin embargo, y esperamos que lo hayamos demostrado a lo largo del libro, es muy fácil de remediar esta deficiencia y agregar los procedimientos necesarios que pueden luego ser usados de la misma manera que las propias funciones incorporadas en el QL. Esta habilidad para aceptar ampliaciones al SuperBASIC es una de las facetas que hace del QL una máquina particularmente flexible y versátil. Al escribir tu propia colección de programas, encontrarás extremadamente útil que puedas ir confeccionando una **programoteca** con tus propios procedimientos que luego pueden ser agregados en programas subsiguientes.

El camino que queda por delante

Si has disfrutado con los juegos de este libro en el sentido de cambiarlos y en general practicar con ellos, la siguiente cosa que has de hacer es escribir un programa de juegos por ti mismo. Tu meta debiera ser la producción de un programa que otra gente pueda usar y con el que pueda disfrutar en lugar de algo **medio acabado** que sólo tú puedes apreciar. Si haces eso, encontrarás que te beneficias doblemente del programa -una vez obtienes beneficio con la diversión del juego, y otra vez de la satisfacción de haber creado y finalizado algo que otra gente puede disfrutar.

GENIO DE LOS JUEGOS CON EL QL

indescomp publicaciones
Avda. del Mediterráneo, 9 — 28007 MADRID