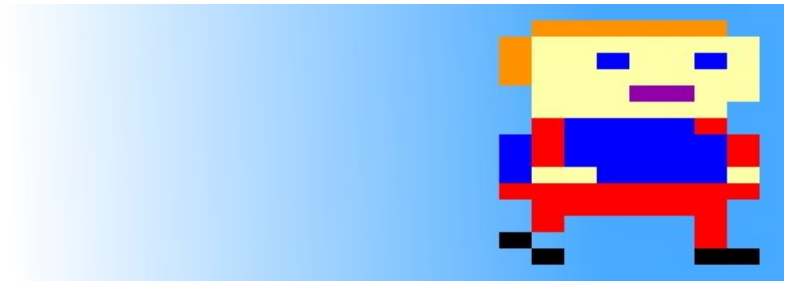


Introducción a la programación de juegos

(versión 2012 – 0.47)

Contenido

0. ¿Por qué este curso?

Crear un juego es uno de los ejercicios de programación más completos. En concreto, este curso está pensado como apoyo para alumnos que estén aprendiendo programación, y que quieran aplicar sus conocimientos a crear juegos sencillos, primero en modo texto y más adelante en modo gráfico.

Avanzaré casi desde cero, de modo que la primera entrega use poco más que una orden de escribir en pantalla (Write), después se vea cómo comprobar condiciones básicas (if), más adelante cómo repetir una zona del programa (while), y así sucesivamente.

El curso usará lenguaje C#, que permite simplificar muchas tareas "rutinarias" si se compara con lenguajes más antiguos como C, pero a la vez tiene una curva de aprendizaje menos pronunciada que la de otros lenguajes modernos como Java. Además, se trata de un lenguaje multi-plataforma, gracias al proyecto Mono, de modo que los ejemplos se podrán probar en Windows, Linux y MacOS X, entre otros sistemas, aunque quizá los ejemplos más avanzados en modo gráfico sólo funcionen bajo Windows.

Condiciones de uso.

¿Cómo puedo colaborar?

Si descubres algún error, házmelo saber.

¿Cómo puedo preguntar dudas?

Si me conoces "en persona", ya sabes dónde encontrarme. ;-)

Si la magia de Google te ha llevado hasta aquí, y tienes dudas, posiblemente tu mejor alternativa será acudir al foro de C# que hay en AprendeAProgramar.com

¿Qué herramientas emplearemos en este curso?

Usaremos el lenguaje C#, apoyándonos en la propia plataforma .Net, o bien en el proyecto Mono. En el próximo apartado veremos cómo instalar estas herramientas.

¿Y si no sé programar en C#?

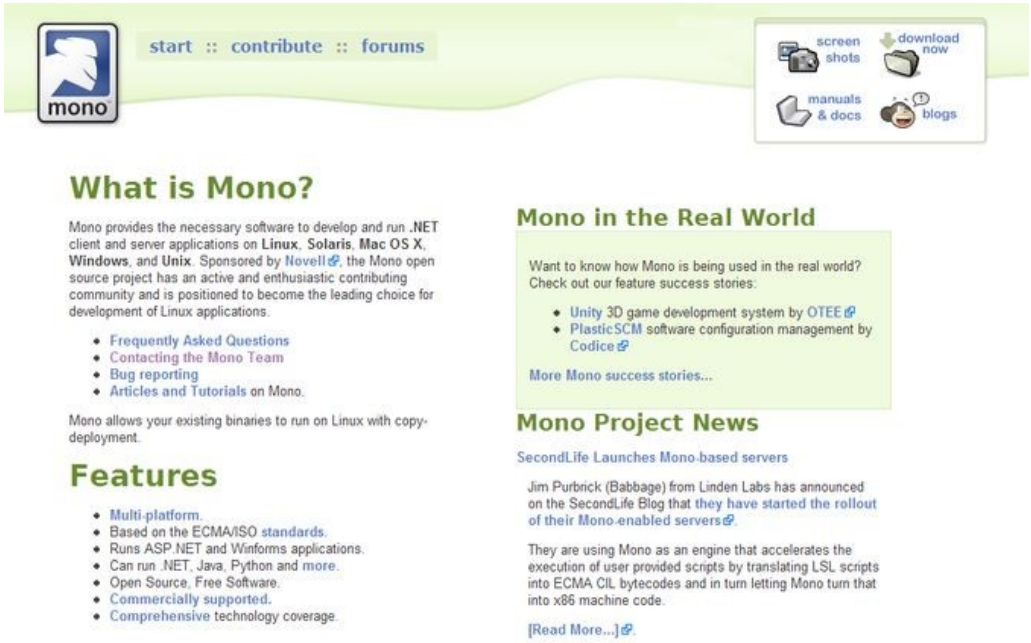
Es casi seguro que serás capaz de seguir el curso sin problemas si tienes conocimientos de c**ualquier otro lenguaje** de programación. Aun así, intentaré que cada apartado sea "casi totalmente autocontenido", porque partirán casi desde cero, el nivel de dificultad será creciente, e incluiré enlaces a los apartados correspondientes de mi curso de C#.

0b. Preparando las herramientas

Usaremos Mono como plataforma de desarrollo para nuestros primeros programas. Por eso, vamos a comenzar por ver dónde encontrar esta herramienta, cómo instalarla y cómo utilizarla.

Podemos descargar Mono desde su página oficial:

http://www.mono-project.com/



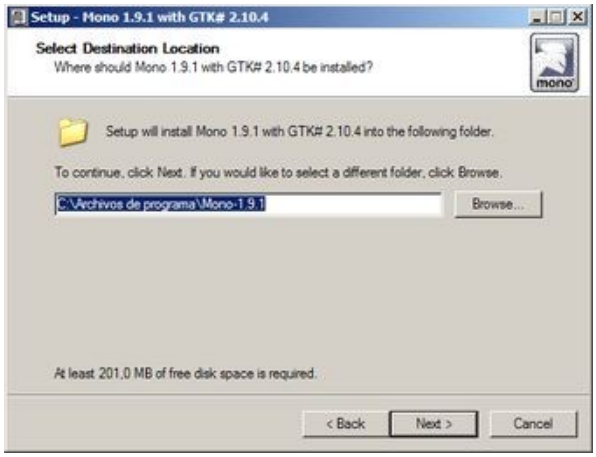
En la parte superior derecha aparece el enlace para descargar ("download now"), que nos lleva a una nueva página en la que debemos elegir la plataforma para la que queremos nuestro Mono. Nosotros descargaremos la versión más reciente para Windows (la 1.9.1 en el momento de escribir este texto).

Se trata de un fichero de algo más de 70 Mb. Cuando termina la descarga, hacemos doble clic en el fichero recibido y comienza la instalación, en la que primero se nos muestra el mensaje de bienvenida:

El siguiente paso será aceptar el acuerdo de licencia.

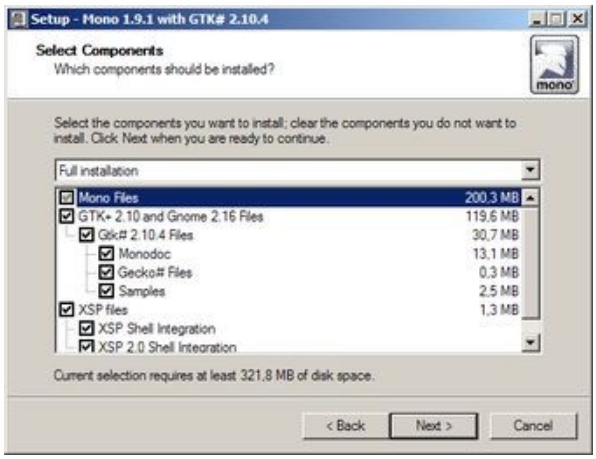
Después se nos muestra una ventana de información, en la que se nos avisa de que se va a instalar Mono 1.9.1, junto con las librerías Gtk# para creación de interfaces de usuario y XSP (eXtensible Server Pages, un servidor web).

A continuación se nos pregunta en qué carpeta queremos instalar. Como es habitual, se nos propone que sea dentro de "Archivos de programa":



Yo no soy partidario de instalar todo en "Archivos de Programa". Mis herramientas de programación suelen estar en otra unidad de disco (D:), así que prefiero cambiar esa opción por defecto:

El siguiente paso es elegir qué componentes queremos instalar (Mono, Gtk#, XSP):



Nuevamente, soy partidario de no instalar todo. Mono es imprescindible. La creación de

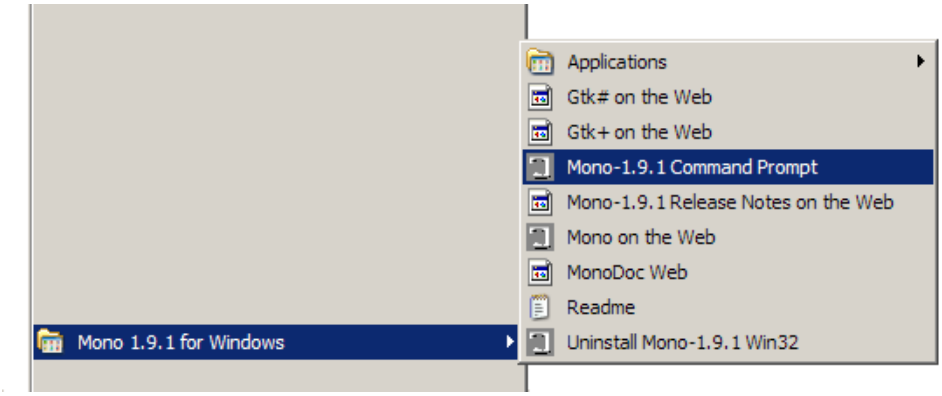
interfaces de usuario con Gtk# queda fuera del alcance que se pretende con este texto, pero aun así puede ser interesante para quien quiera profundizar. El servidor web XSP es algo claramente innecesario por ahora, y que además instalaría un "listener" que ralentizaría ligeramente el ordenador, así que puede ser razonable no instalarlo por ahora:

El siguiente paso es indicar en qué carpeta del menú de Inicio queremos que quede accesible:

A continuación se nos muestra el resumen de lo que se va a instalar. Si confirmamos que todo nos parece correcto, comienza la copia de ficheros:

Si todo es correcto, al cabo de un instante tendremos el mensaje de confirmación de que la instalación se ha completado:

Mono está listo para usar. En nuestro menú de Inicio deberíamos tener una nueva carpeta llamada "Mono 1.9.1 for Windows", y dentro de ella un acceso a "Mono-1.9.1 Command Prompt":

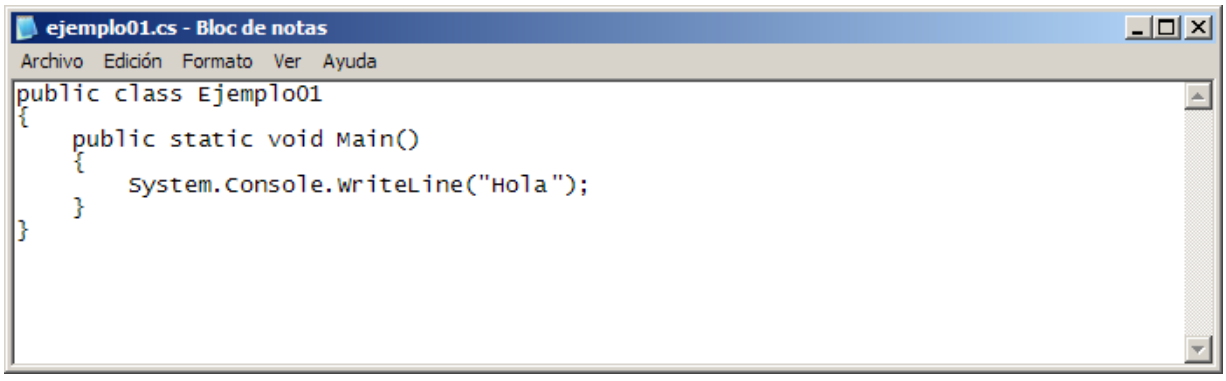


Si hacemos clic en esa opción, accedemos al símbolo de sistema ("command prompt"), la pantalla negra del sistema operativo, pero con el "path" (la ruta de búsqueda) preparada para que podamos acceder al compilador desde ella:

Primero debemos teclear nuestro fuente. Para ello podemos usar cualquier editor de texto. En este primer fuente, usaremos simplemente el "Bloc de notas" de Windows. Para ello tecleamos:

```
notepad ejemplo01.cs
```

Aparecerá la pantalla del "Bloc de notas", junto con un aviso que nos indica que no existe ese fichero, y que nos pregunta si deseamos crearlo. Respondemos que sí y podemos empezar a teclear el ejemplo que habíamos visto anteriormente:



Guardamos los cambios, salimos del "Bloc de notas" y nos volvemos a encontrar en la pantalla negra del símbolo del sistema. Nuestro fuente ya está escrito. El siguiente paso es compilarlo. Para eso, tecleamos

```
gmcs ejemplo01.cs
```

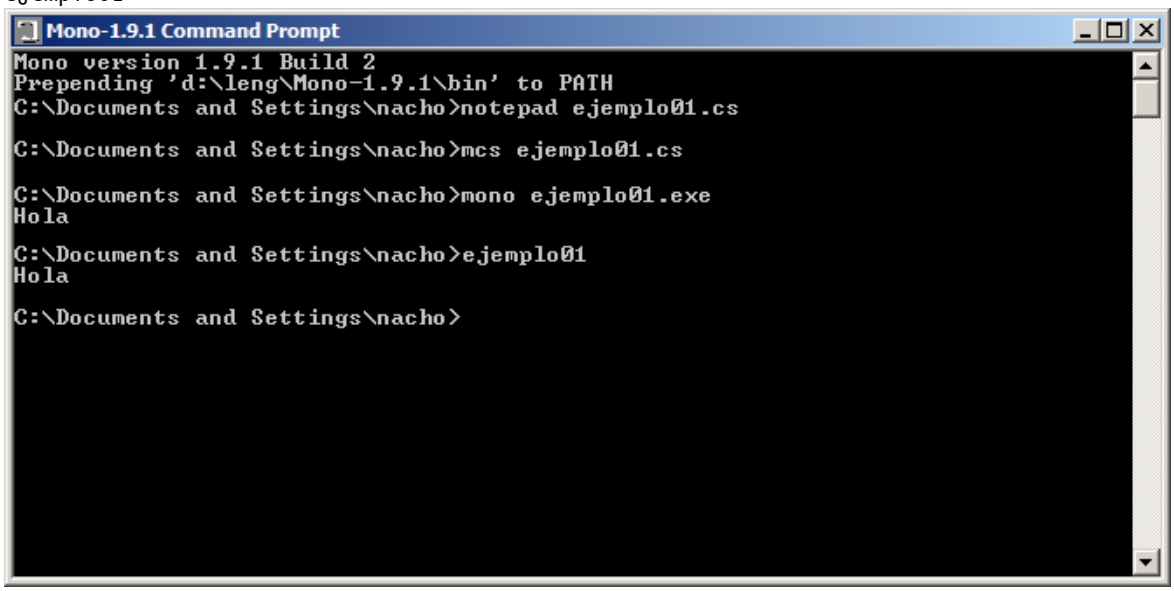
Si no se nos responde nada, quiere decir que no ha habido errores. Si todo va bien, se acaba de crear un fichero "ejemplo01.exe". En ese caso, podríamos lanzar el programa tecleando

```
mono ejemplo01
```

y el mensaje "Hola" debería aparecer en pantalla.

Si en nuestro ordenador está instalado el "Dot Net Framework" (algo que debería ser cierto en las últimas versiones de Windows), no debería hacer falta decir que queremos que sea Mono quien lance nuestro programa, y podremos ejecutarlo directamente con su nombre:

```
ejemplo01
```



Nota: Si quieres un editor más potente que el Bloc de notas de Windows, puedes probar Notepad++, que es gratuito (realmente más que eso: es de "código abierto") y podrás localizar fácilmente en Internet.

1. Escribir en pantalla en modo texto: lanzar un dado

Sencillo: se usa la orden Write si queremos escribir sin avanzar a la línea siguiente, o WriteLine cuando queramos avanzar.

```
// Escribir en pantalla

using System;
public class Juego01a
{
        public static void Main()
        {
                Console.WriteLine("¡Hola!");
        }
}
```

Las líneas que comienzan por doble barra son comentarios: aclaraciones para nosotros, pero no son "órdenes" como tal.

La única orden que nos interesa por ahora es la de "Console.WriteLine"; el resto, "deben estar", aunque no sepamos todavía por qué.

Números al azar

Como primera aproximación, podemos tomar los milisegundos de la hora actual, que es un valor entre 0 y 999 y es raro que se repita entre dos ejecuciones del "juego".

```
// Milisegundos

using System;
public class Juego01b
{
    public static void Main()
    {
        int miliseg = DateTime.Now.Millisecond;
        Console.WriteLine(miliseg);
    }
}
```

Si queremos que el número al azar esté entre 1 y 6, podemos usar la operación "módulo" (resto de la división): al hallar el x % 6 obtendremos un número entre 0 y 5, al que basta sumar uno para que quede entre 1 y 6.

```
// Dado del 1 al 6

using System;
public class Juego01c
{
    public static void Main()
    {
        int miliseg = DateTime.Now.Millisecond;
        int dado = miliseg % 6 + 1;

        Console.Write("El numero del dado es ");
        Console.WriteLine(dado);
    }
}
```

Para escribir dos (o más) "cosas" en la misma línea, podemos usar un "Write" y un "WriteLine", como en el ejemplo anterior, o bien usar un único WriteLine, que contenga {0} en la posición en la que queremos que se vea un dato, así:

```
// Dado del 1 al 6, con {0}

using System;
public class Juego01d
{
    public static void Main()
    {
        int miliseg = DateTime.Now.Millisecond;
        int dado = miliseg % 6 + 1;

        Console.WriteLine("El numero del dado es {0}",dado);
    }
}
```

Ejercicio propuesto: Mostrar un número al azar del 1 al 100.

2. Lanzar dos dados a la vez

Si queremos obtener dos números al azar, no sirve el método de tomar los milisegundos del reloj, porque con casi total seguridad ambos números tendrían el mismo valor.

La alternativa es usar Random: crear una variable de tipo Random, y usar Next para cada número que haya que obtener. A "Next" le indicaremos dos datos adicionales: el menor valor que queremos obtener y un valor justo por encima de los permitidos (para un número del 1 al 6, usaríamos 1,7) así:

```
// Lanzar dos dados seguidos

using System;
public class Juego01e
{
    public static void Main()
    {
        Random dado = new Random();

        int aleatorio1 = dado.Next(1,7);
        Console.WriteLine("El numero del 1º dado es {0}",aleatorio1);

        int aleatorio2 = dado.Next(1,7);
        Console.WriteLine("El numero del 2º dado es {0}",aleatorio2);
    }
}
```

Ejercicio propuesto: Mostrar un número al azar del 1 al 10 y otro entre 11 y 20.

3. Condiciones: El usuario intenta acertar en una vez

Podemos pedir un número al usuario, con "Console.ReadLine". Como es un número en vez de un texto, deberemos convertirlo con "Convert.ToInt32".

Para comprobar condiciones, y hacer algo en caso de que se cumplan, usaremos la orden "if". Entre paréntesis indicaremos la condición a comprobar, teniendo en cuenta que las comparaciones posibles son:

Operador	Operación
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual a
!=	No igual a (distinto de)

Con todo eso, tendríamos:

```
// Intentar acertar un dado a la primera

using System;
public class Juego01f
{
    public static void Main()
    {
        Random generador = new Random(DateTime.Now.Millisecond);

        int aleatorio = generador.Next(1, 7);
        int usuario;

        Console.WriteLine("Acierta el dado! Dime un número del 1 al 6");
        usuario = Convert.ToInt32(Console.ReadLine());

        if (usuario == aleatorio)
            Console.WriteLine("Acierto");

        if (usuario != aleatorio)
            Console.WriteLine("No has acertado el número era {0}",aleatorio);
    }
}
```

Como las dos condiciones son complementarias, podemos resumirlo en una única condición que termine con la orden "else" (en caso contrario), así:

```
// Intentar acertar un dado a la primera (2)

using System;
public class Juego01g
{
    public static void Main()
    {
        Random generador = new Random(DateTime.Now.Millisecond);

        int aleatorio = generador.Next(1, 7);
        int usuario;
```

```
        Console.WriteLine("Acierta el dado! Dime un número del 1 al 6");
        usuario = Convert.ToInt32(Console.ReadLine());

        if (usuario == aleatorio)
            Console.WriteLine("Acierto");
        else
            Console.WriteLine("No has acertado el número era {0}",aleatorio);
    }
}
```

Ejercicio propuesto: Preguntar al usuario si desea mostrar un número al azar del 1 al 10 (opción 1) o bien del 11 al 20 (opción 2).

4. Condiciones enlazadas: El usuario tiene dos intentos

Si queremos dar dos oportunidades, tendremos que pedir datos dos veces. Realmente, el segundo intento será sólo si el jugador falla el primero, de modo que tendremos un "if" dentro de otro.

Cuando queramos que se den varias pasos si se cumple o no se cumple una condición, deberemos encerrar todos los pasos entre llaves. Ese es el caso de nuestro "else", en esta variante del juego:

```
// Intentar acertar un dado en dos intentos

using System;
public class Juego01h
{
  public static void Main ()
  {

      int numero;
      Random r = new Random();
      int aleatorio = r.Next(1, 7);

      Console.WriteLine("Indica el numero que creas: ");
      numero = Convert.ToInt32(Console.ReadLine());

      if (numero == aleatorio)
          Console.WriteLine("Correcto, has acertado el numero");

      else
      {

        if (numero < aleatorio)
            Console.WriteLine("Prueba un numero mas alto");
        if (numero > aleatorio)
            Console.WriteLine("Prueba un numero mas bajo");

        Console.WriteLine("Indica el numero que creas: ");
        numero = Convert.ToInt32(Console.ReadLine());

        if (numero == aleatorio)
            Console.WriteLine("Correcto, has acertado el numero");
        else
            Console.WriteLine("No has acertado, era {0}",aleatorio);
        }
```

```
    }
}
```

Ejercicio propuesto: Ampliar el juego para que dé una tercera oportunidad al usuario si tampoco acierta en el segundo intento.

5. Repetir: adivinar un número del 1 al 1000

Hemos visto que dar varios intentos al usuario usando "if" va haciendo el programa cada vez menos legible y más complejo.

Hay una alternativa más razonable para estos casos: las estructuras repetitivas, como "while", que repiten un bloque de programa mientras se cumpla una condición. Así podemos dar al usuario 10 intentos sin que eso dispare la dificultad del programa.

Para probarlo, vamos a crear un programa que dé al usuario 10 oportunidades de adivinar un número del 1 al 1000.

Además de la orden "while", apenas hay una novedad: cómo comprobar dos condiciones a la vez, porque la partida podrá terminar cuando el usuario agote sus 10 oportunidades o quizá antes, si encuentra el número en menos de 10 intentos. Podemos enlazar varias condiciones con:

Operador	Significado
&&	Y
||	O
!	No

También hay que tener en cuenta que como se repita mientras se den dos condiciones (no haber acertado y no haber agotado intentos), cuando dejemos de repetir habrá que comprobar por cuál de esos dos motivos hemos salido. Es más razonable fijarnos en si ha acertado que en si ha usado los 10 intentos, porque puede ocurrir que el usuario acierte justo en su última oportunidad:

```
// Adivinar un numero del 1 al 1000 en 10 intentos

/* Pseudocodigo:
 *
- Generar aleatorio
- Pedir número
- restantes = 0
- Mientras no acertado y queden intentos restantes
  - Si número > aleatorio, Escribir "Pasado"
  - Si número < aleatorio, Escribir "Corto"
  - Disminuir restantes
  - Pedir número
- Fin mientras
- Si número = aleatorio, Escribir "Felicidades"
  Si no, Escribir aleatorio
*/


using System;

public class Juego02
{
    public static void Main()
    {
```

```
        //Declaración de variables
        int aleatorio, introducido;
        int restantes = 10;

        Random generador = new Random();
        aleatorio = generador.Next(1,1001);

        Console.WriteLine("Te quedan {0} intentos", restantes);
        Console.Write("Introduce un número: ");
        introducido = Convert.ToInt32(Console.ReadLine());

        //Bucle que se repite hasta que acierte o se quede sin intentos

        while (( introducido != aleatorio ) && ( restantes > 1 ))
        {
            restantes = restantes - 1;

            if (introducido < aleatorio)
                Console.WriteLine("Te has quedado corto");

            if (introducido > aleatorio)
                Console.WriteLine("Te has pasado");

            Console.WriteLine("Te quedan {0} intentos", restantes);
            Console.Write("Introduce un número: ");
            introducido = Convert.ToInt32(Console.ReadLine());
        }

        //Comprobar si gana o pierde

        if ( introducido == aleatorio )
            Console.WriteLine("Has ganado!");
        else
        {
            Console.WriteLine("Has perdido!");
            Console.WriteLine("Era el {0}",aleatorio);
        }
    }
}
```

Ejercicio propuesto: Modificar el juego para que el número esté entre 1 y 10.000, y el usuario tenga 13 intentos (intenta hacerlo desde cero, y comparar tu solución con la propuesta).

6. Un "bucle de juego". Consola avanzada 1 - Escribir en cualquier posición

Durante varias entregas, desarrollaremos el esqueleto de un juego de plataformas básico, pero "en modo texto" (usando letras en vez de imágenes), que nos permita practicar conceptos y estructuras un poco más avanzados, pero nos permita despreocuparnos todavía de detalles más difíciles como la carga de imágenes, la realización de animación o la reproducción de sonidos.

En el primer acercamiento, nos limitaremos a mover una letra por pantalla. Primero lo haremos usando "teclas normales", y después lo mejoraremos para que se puedan usar las flechas del teclado.

Vamos con las novedades:

- Usaremos "Console.Clear();" para borrar la pantalla en cada "fotograma", y que nuestro "personaje" no deje rastro al moverse.
- Con "Console.SetCursorPosition(x, y);" podremos colocar un texto en cierta posición de pantalla. La coordenada "x" se refiere a la posición en horizontal, empezando a contar desde el extremo izquierdo de la pantalla, y típicamente valdrá de 0 a 79. La coordenada "y" indica la posición vertical, contando desde el extremo superior de la pantalla, y típicamente valdrá de 0 a 24.
- Estas dos órdenes necesitan que nuestra versión de Mono (o de la plataforma .Net) sea la 2 o superior, y deberemos compilar con "gmcs", no con "mcs". No debería suponer un problema.
- Podemos mover el personaje con teclas como QAOP (Q=arriba, A=abajo, O=izquierda, P=derecha). Las leeremos como texto (string), por lo que no hará falta convertir. Eso sí, en este primer acercamiento habrá que pulsar Intro después de cada una de esas teclas.
- Usaremos expresiones abreviadas, como "x++;" para aumentar el valor de la coordenada x (horizontal), o "y--;" para disminuir la y (vertical).

Además, vamos a hacer que nuestro juego se repita indefinidamente, con una condición como "while (1==1)", y a hacer que la parte repetitiva siga la estructura de un "bucle de juego" clásico, en el que se repiten continuamente una serie de tareas como:

- Dibujar los elementos en pantalla
- Comprobar si el usuario ha pulsado alguna tecla (o ha movido el joystick, o el ratón, o algún otro dispositivo de entrada)
- Mover los enemigos, el entorno y cualquier otro componente del juego que sea capaz de moverse por sí solo.
- Comprobar colisiones, para ver si hemos recogido algún objeto, tocado algún enemigo, si algún disparo ha impactado con algo, etc., y así poder actualizar los puntos, las vidas y el resto del estado actual del juego.
- Si queremos que la velocidad del juego sea "estable" (que no se mueva mucho más rápido en ordenadores más recientes, lo que lo haría injugable), quizá necesitemos hacer una pausa al final de cada "fotograma".

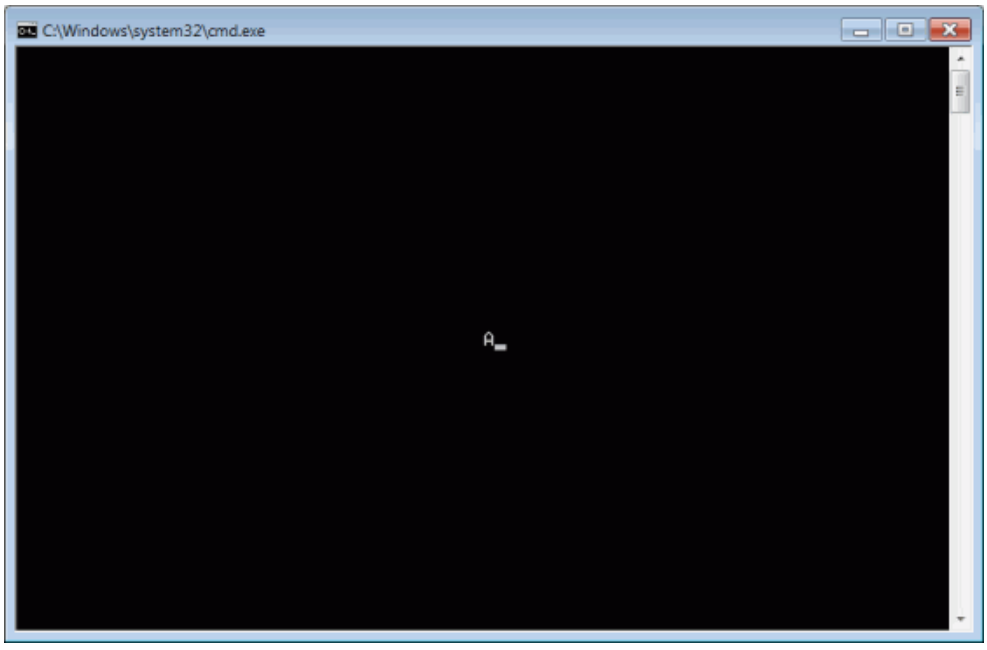
La mayoría de las tareas de este "bucle de juego" las iremos completando más adelante, y de momento quedarán como comentarios, para que el fuente "recuerde" a lo que acabará siendo.

El resultado podría ser algo como:

```
// Primer mini-esqueleto de juego en modo texto
// Versión "a"

using System;

public class Juego03a
{
     public static void Main()
     {
          string tecla;
          int x = 40, y=12;

          // Bucle de juego
          while( 1 == 1 )
          {
```

```
            // Dibujar
            Console.Clear();
            Console.SetCursorPosition(x, y);
            Console.Write("A");

            // Leer teclas y calcular nueva posición
            tecla = Console.ReadLine();

            if (tecla == "p") x++;
            if (tecla == "o") x--;
            if (tecla == "a") y++;
            if (tecla == "q") y--;

            // Mover enemigos, entorno
            // NADA

            // Colisiones, perder vidas, etc
            // NADA

            // Pausa hasta el siguiente "fotograma" del juego
            // NO TODAVIA
        }
    }
}
```

Que en pantalla se vería:



Eso de tener que pulsar Intro después de cada tecla para moverse... no queda muy amigable. Lo podemos mejorar si accedemos directamente al teclado. Lo veremos en el siguiente apartado...

Ejercicio propuesto: Mejora el juego para que no falle cuando el usuario se intenta mover más allá del extremo de la pantalla (por la derecha, izquierda o por arriba).

7. Consola avanzada 2 - Comprobar teclas sin esperar Intro

Hemos conseguido mover algo por la pantalla, pero eso de tener que pulsar Intro tras escoger cada dirección lo hace "poco jugable".

Ahora las novedades son:

- Para acceder directamente al teclado, necesitaremos usar un nuevo tipo de datos, llamado "ConsoleKeyInfo".
- Utilizaremos "Console.ReadKey();" para esperar a que se pulse una tecla (pero no Intro), y "Console.ReadKey(false);" si además queremos que esa tecla pulsada NO se muestre en pantalla.
- Para comprobar el código de la tecla pulsada usaremos ".Key", y compararemos con ciertos valores predefinidos, como "ConsoleKey.RightArrow" para la flecha derecha.

Suena más difícil de lo que realmente es:

```
// Primer mini-esqueleto de juego en modo texto
// Versión "b"

using System;

public class Juego03b
{
    public static void Main()
    {
        ConsoleKeyInfo tecla;
        int x = 40, y=12;

        // Bucle de juego
        while( 1 == 1 )
        {
            // Dibujar
            Console.Clear();
            Console.SetCursorPosition(x, y);
            Console.Write("A");

            // Leer teclas y calcular nueva posición
            tecla = Console.ReadKey(false);

            if(tecla.Key == ConsoleKey.RightArrow) x++;
            if(tecla.Key == ConsoleKey.LeftArrow) x--;
            if(tecla.Key == ConsoleKey.DownArrow) y++;
            if(tecla.Key == ConsoleKey.UpArrow) y--;

            // Mover enemigos, entorno
            // NADA

            // Colisiones, perder vidas, etc
            // NADA

            // Pausa hasta el siguiente "fotograma" del juego
            // NO TODAVIA
        }
    }
}
```

Ejercicio propuesto: Amplía el juego para que también se pueda mover con las teclas QAOP (sin pulsar Intro). Pista, como esas teclas, al contrario que las flechas, sí tienen un carácter asociado, puedes usar ".KeyChar" en vez de ".Key": if (tecla.KeyChar == 'a') ...

8. Dibujar tres obstáculos. Comprobar colisiones.

Vamos a hacer que aparezcan 3 obstáculos en pantalla, y que la partida acabe cuando choquemos con uno de ellos.

Esto supone algunos cambios, la mayoría sencillos:

- Si queremos tres obstáculos, necesitaremos 3 nuevas coordenadas X y 3 nuevas coordenadas Y (ya veremos formas menos repetitivas de trabajar), que llevaremos xo1, xo2 y así sucesivamente.
- A la hora de dibujar cosas en pantalla, ya no sólo dibujaremos nuestro personaje, sino también los obstáculos.
- El juego ya no se repetirá siempre (con ese "mientras 1=1"), sino que ahora usaremos una variable "fin", para comprobar si ha terminado la partida o no. A falta de conocer formas más elegantes, podemos hacer que la variable "fin" valga 0 cuando la partida no haya terminado, o bien 1 cuando sí termine (cuando choquemos con un obstáculo).
- Eso de comprobar colisiones también es fácil: por ahora bastará con ver si nuestro personaje está en la misma posición de pantalla que alguno de los obstáculos, comparando sus coordenadas X e Y: "if ((x==x2) && (y==y2)) fin = 1;"

El resultado sería:

```
// Primer mini-esqueleto de juego en modo texto
// Versión "c"

using System;

public class Juego03c
{
     public static void Main()
     {
          ConsoleKeyInfo tecla;
          int x = 40, y=12;
          int xo1 = 20, yo1 = 15;  // Obstáculo 1
          int xo2 = 25, yo2 = 5;   // Obstáculo 2
          int xo3 = 62, yo3 = 21;  // Obstáculo 3
          int fin = 0;   // 0 = no terminado, 1 = terminado

          // Bucle de juego
          while( fin == 0 )
          {
               // Dibujar
               Console.Clear();
               Console.SetCursorPosition(x, y);
               Console.Write("A");

               Console.SetCursorPosition(xo1, yo1); // Obstáculos
               Console.Write("o");

               Console.SetCursorPosition(xo2, yo2);
               Console.Write("o");

               Console.SetCursorPosition(xo3, yo3);
               Console.Write("o");

               // Leer teclas y calcular nueva posición
               tecla = Console.ReadKey(false);

               if(tecla.Key == ConsoleKey.RightArrow) x++;
```

```
            if(tecla.Key == ConsoleKey.LeftArrow) x--;
            if(tecla.Key == ConsoleKey.DownArrow) y++;
            if(tecla.Key == ConsoleKey.UpArrow) y--;

            // Mover enemigos, entorno
            // NADA

            // Colisiones, perder vidas, etc
            if ( ( (x==xo1) && (y==yo1) )
                 || ( (x==xo2) && (y==yo2) )
                 || ( (x==xo3) && (y==yo3) )
               )
               fin = 1;

            // Pausa hasta el siguiente "fotograma" del juego
            // NO TODAVIA
        }
    }
}
```

Ejercicio propuesto: Mejora esta versión del juego, para que las posiciones de los obstáculos no sean prefijadas, sino que se escojan al azar.

9. Un enemigo móvil

Hacer que un enemigo se mueva no es especialmente difícil: preparamos sus coordenadas x e y (que podríamos llamar xe1, ye1 por eso de que son del primer enemigo), lo dibujamos cuando sea el momento de dibujar cosas en pantalla y cambiamos su x y su y en la zona de "mover enemigos".

La forma más simple de cambiar su posición es modificando su x (por ejemplo). Podemos aumentar su x una unidad en cada fotograma haciendo "x = x + 1;", que se puede abreviar "x++;"

Pero eso de siempre aumentar el valor de X puede provocar que nos salgamos pronto de la pantalla, y esto puede provocar que el enemigo no se vea o incluso que el programa falle al intentar escribir en posiciones no permitidas de la pantalla. En la mayoría de bibliotecas de juegos (como la que usaremos más adelante), se puede escribir fuera de la pantalla sin problemas, pero no en la "consola" que estamos usando todavía, así que lo que haremos es comprobar si llegamos al extremo de la pantalla, para hacer que entonces el enemigo "rebote" y vuelva por donde venía.

Hay varias formas de conseguirlo. En la mayoría de casos, pasan por no sumar siempre 1, sino un "incremento", que empezará valiendo 1 (para que se mueva hacia la derecha) pero que en ciertos momentos nos puede interesar cambiar a -1 (para que vuelva hacia la izquierda).

Lo podríamos hacer con una construcción como

```
if (xe1 == 0) incr1 = 1;
if (xe1 == 79) incr1 = -1;
```

o bien, en una única condición:

```
if ((xe1 == 0) || (xe1 == 79))
     incr1 = - incr1;
```

Yo usaré la segunda construcción. El juego completo podría quedar así:

```
// Primer mini-esqueleto de juego en modo texto
// Versión "d"
```

```
using System;

public class Juego03d
{
     public static void Main()
     {
          ConsoleKeyInfo tecla;
          int x = 40, y=12;
          int xo1 = 20, yo1 = 15;  // Obstáculo 1
          int xo2 = 25, yo2 = 5;   // Obstáculo 2
          int xo3 = 62, yo3 = 21;  // Obstáculo 3
          int xe1 = 10, ye1 = 10, incr1 = 1;  // Enemigo 1
          int fin = 0;   // 0 = no terminado, 1 = terminado

          // Bucle de juego
          while( fin == 0 )
          {
               // Dibujar
               Console.Clear();
               Console.SetCursorPosition(x, y);
               Console.Write("A");

               Console.SetCursorPosition(xo1, yo1); // Obstáculos
               Console.Write("o");

               Console.SetCursorPosition(xo2, yo2);
               Console.Write("o");

               Console.SetCursorPosition(xo3, yo3);
               Console.Write("o");

               Console.SetCursorPosition(xe1, ye1); // Enemigo
               Console.Write("@");

               // Leer teclas y calcular nueva posición
               tecla = Console.ReadKey(false);

               if(tecla.Key == ConsoleKey.RightArrow) x++;
               if(tecla.Key == ConsoleKey.LeftArrow) x--;
               if(tecla.Key == ConsoleKey.DownArrow) y++;
               if(tecla.Key == ConsoleKey.UpArrow) y--;

               // Mover enemigos, entorno
               xe1 = xe1 + incr1;

               if ((xe1 == 0) || (xe1 == 79))
                    incr1 = - incr1;

               // Colisiones, perder vidas, etc
               if ( ( (x==xo1) && (y==yo1) )
                    || ( (x==xo2) && (y==yo2) )
                    || ( (x==xo3) && (y==yo3) )
                  )
                  fin = 1;

               // Pausa hasta el siguiente "fotograma" del juego
               // NO TODAVIA
          }
     }
```

```
}
```

Se puede ver que todavía "no es muy real": el enemigo se mueve, pero sólo cuando nosotros nos movemos. Lo solucionaremos en el siguiente apartado.

Ejercicio propuesto: Añade un segundo enemigo que también se mueva.

10. El enemigo se mueve "solo"

Hasta ahora, nuestro juego "se queda parado" hasta que pulsemos una tecla. Eso es aceptable cuando nuestro personaje es lo único que se mueve, pero en la mayoría de casos no es aceptable, y ya hemos visto que hace un efecto "desconcertante" ahora que tenemos un enemigo... que sólo se mueve exactamente en el mismo momento que nosotros.

La solución es sencilla: no dejar el juego parado, sino comprobar si hay una tecla disponible, y sólo en ese caso mirar de qué tecla se trata y mover nuestro personaje (si corresponde). Eso lo podemos hacer con la construcción "if (Console.KeyAvailable) ... "

Eso todavía no es perfecto, porque permitirá que nuestro enemigo se mueva aunque nosotros no lo hagamos, pero se moverá a la velocidad máxima que permita nuestro ordenador, lo que generalmente no es jugable.

Nos interesa que se mueva solo, pero no tan rápido, sino a una velocidad estable, que sea la misma en cualquier ordenador. Lo podemos conseguir añadiendo una pequeña pausa al final de cada fotograma. Por ejemplo, si queremos que nuestro juego se mueva a 25 fotogramas por segundo (25 fps), podríamos hacer una pausa de 1000 / 25 = 40 milisegundos. Para hacer esta pausa podemos usar "Thread.Sleep(40)", que nos obliga a incluir "using System.Threading;" al principio de nuestro programa.

En un juego real, si queremos que se tarde 40 milisegundos en cada fotograma (para que la velocidad realmente sea de 25 fps), no deberíamos esperar 40 milisegundos, sino contar cuánto hemos tardado en dibujar, calcular siguientes posiciones, comprobar colisiones, etc.; si estas tareas nos llevan 15 ms, deberíamos esperar otros 25 ms, para hacer un total de 40 milisegundos por fotograma. En nuestro juego, estas tareas son tan sencillas (por ahora) que tardan un tiempo despreciable, así que todavía no nos molestaremos en contar tiempos.

Con estas consideraciones, el juego podría quedar así:

```
// Primer mini-esqueleto de juego en modo texto
// Versión "e"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego03e
{
    public static void Main()
    {
        ConsoleKeyInfo tecla;
        int x = 40, y=12;
        int xo1 = 20, yo1 = 15;  // Obstáculo 1
        int xo2 = 25, yo2 = 5;   // Obstáculo 2
        int xo3 = 62, yo3 = 21;  // Obstáculo 3
        int xe1 = 10, ye1 = 10, incr1 = 1;  // Enemigo 1
        int fin = 0;   // 0 = no terminado, 1 = terminado

        // Bucle de juego
        while( fin == 0 )
        {
            // Dibujar
```

```
            Console.Clear();
            Console.SetCursorPosition(x, y);
            Console.Write("A");

            Console.SetCursorPosition(xo1, yo1); // Obstáculos
            Console.Write("o");

            Console.SetCursorPosition(xo2, yo2);
            Console.Write("o");

            Console.SetCursorPosition(xo3, yo3);
            Console.Write("o");

            Console.SetCursorPosition(xe1, ye1); // Enemigo
            Console.Write("@");

            // Leer teclas y calcular nueva posición
            if (Console.KeyAvailable)
            {
                 tecla = Console.ReadKey(false);

                 if(tecla.Key == ConsoleKey.RightArrow) x++;
                 if(tecla.Key == ConsoleKey.LeftArrow) x--;
                 if(tecla.Key == ConsoleKey.DownArrow) y++;
                 if(tecla.Key == ConsoleKey.UpArrow) y--;
            }

            // Mover enemigos, entorno
            xe1 = xe1 + incr1;

            if ((xe1 == 0) || (xe1 == 79))
                 incr1 = - incr1;

            // Colisiones, perder vidas, etc
            if ( ( (x==xo1) && (y==yo1) )
                 || ( (x==xo2) && (y==yo2) )
                 || ( (x==xo3) && (y==yo3) )
               )
               fin = 1;

            // Pausa hasta el siguiente "fotograma" del juego
            Thread.Sleep(40);
        }
    }
}
```

Ejercicio propuesto: Amplía el juego para que realmente calcule el tiempo empleado en cada fotograma (mirando los milisegundos de la hora actual), y espere menos de 40 ms si es necesario.

11. Ajustando la velocidad.

Ahora mismo, el personaje y el enemigo se mueven a la misma velocidad. Generalmente, eso no es lo más deseable para la jugabilidad. Suele ser preferible que se puedan mover a velocidad distinta, típicamente algo mayor para el personaje, de modo que "tengamos alguna posibilidad de escapar".

Por ejemplo, si queremos que nuestro personaje vaya al doble de velocidad que el enemigo, lo podemos hacer cambiando uno de los "incrementos" de su movimiento (que equivalen a la velocidad de cada uno de ellos).

Una posibilidad es que la velocidad del enemigo sea 1 ("incr1 = 1;") y que la velocidad de nuestro personaje sea 2 ("if(tecla.Key == ConsoleKey.RightArrow) x=x+2;"). La otra alternativa es que nuestro personaje se mueva de uno en uno, y que el enemigo lo haga "de media casilla en media casilla".

La primera alternativa (mover de 2 en 2) supone perder precisión de movimiento, y estamos trabajando en "modo texto", que ya de por sí supone poca precisión en pantalla. La segunda alternativa no supone perder precisión de movimiento (aunque cuando su x sea 2.5 no se verá ninguna diferencia con cuando x sea 2), y además permitirá practicar el uso de números con decimales, así que vamos a optar por esta segunda opción.

Ahora, la X, la Y y la velocidad del enemigo serán números reales:

```
float xe1 = 10, ye1 = 10, incr1 = 0.5f;
```

Pero al colocarnos en pantalla tendremos que "quitar los decimales", porque "SetCursorPosition" necesita trabajar con números enteros. La forma más simple es hacer una "conversión de tipos", poniendo "(int)" antes de las coordenadas:

```
Console.SetCursorPosition((int) xe1, (int) ye1);
```

El resultado sería:

```
// Primer mini-esqueleto de juego en modo texto
// Versión "f"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego03f
{
     public static void Main()
     {
          ConsoleKeyInfo tecla;
          int x = 40, y=12;
          int xo1 = 20, yo1 = 15;  // Obstáculo 1
          int xo2 = 25, yo2 = 5;   // Obstáculo 2
          int xo3 = 62, yo3 = 21;  // Obstáculo 3
          float xe1 = 10, ye1 = 10, incr1 = 0.5f;  // Enemigo 1
          int fin = 0;   // 0 = no terminado, 1 = terminado

          // Bucle de juego
          while( fin == 0 )
          {
               // Dibujar
               Console.Clear();
               Console.SetCursorPosition(x, y);
               Console.Write("A");

               Console.SetCursorPosition(xo1, yo1); // Obstáculos
               Console.Write("o");

               Console.SetCursorPosition(xo2, yo2);
               Console.Write("o");

               Console.SetCursorPosition(xo3, yo3);
               Console.Write("o");
```

```
            Console.SetCursorPosition((int) xe1, (int) ye1); // Enemigo
            Console.Write("@");

            // Leer teclas y calcular nueva posición
            if (Console.KeyAvailable)
            {
                 tecla = Console.ReadKey(false);

                 if(tecla.Key == ConsoleKey.RightArrow) x++;
                 if(tecla.Key == ConsoleKey.LeftArrow) x--;
                 if(tecla.Key == ConsoleKey.DownArrow) y++;
                 if(tecla.Key == ConsoleKey.UpArrow) y--;
            }

            // Mover enemigos, entorno
            xe1 = xe1 + incr1;

            if ((xe1 == 0) || (xe1 == 79))
                 incr1 = - incr1;

            // Colisiones, perder vidas, etc
            if ( ( (x==xo1) && (y==yo1) )
                 || ( (x==xo2) && (y==yo2) )
                 || ( (x==xo3) && (y==yo3) )
               )
               fin = 1;

            // Pausa hasta el siguiente "fotograma" del juego
            Thread.Sleep(40);
        }
    }
}
```

Ejercicio propuesto: Afina el juego, para que vaya a 50 fotogramas por segundo, y la velocidad del enemigo sea un 60% de la velocidad del personaje.

Nota: Hay ciertas cosas que no podremos evitar, porque no estamos usando una biblioteca de juegos real, sino la "consola de Windows". Por eso, si mantenemos una tecla pulsada, hay un retardo antes de que nuestro personaje se empiece a mover de forma repetitiva, igual que puede ocurrir que dejemos de pulsar la tecla y el personaje se siga moviendo un instante.

12. Arrays: simplificando la lógica

Ahora vamos a añadir 10 "premios" que (más adelante) podamos recoger para obtener puntos. Crear 10 variables X y 10 variables Y debería empezar a no parecer la solución más adecuada, y menos aún si hablamos de 100 premios en vez de 10.

Usaremos una única estructura de datos para guardar todas las X de esos 10 premios, como si se tratara de un cajón del que obtenemos una ficha concreta cuando nos hace falta. Esa estructura que contendrá esas 10 X juntas es lo que llamaremos una "tabla" o "array". Por ejemplo, podríamos prefijar 10 valores para las X de los premios haciendo:

```
int[] xPremio = {20,25,18,10,11,3,8,15,13,22};
```

De modo que xPremio[0] será la coordenada X del primer premio, y xPremio[9] será la del último.

En general, será más cómodo recorrer el array desde la posición 0 a la 9 de forma repetitiva, usando "for". Por ejemplo, podríamos mostrar los premios en pantalla haciendo:

```
for (int i=0; i<=9; i++)
{
          Console.SetCursorPosition(xPremio[i], yPremio[i]);
          Console.Write("/");
}
```

Hay programadores que prefieren que el contador del "for" llegue hasta 9, incluido (i<=9) y hay otros que prefieren expresarlo como que no llega hasta el 10 (i<10), así:

```
for (int i=0; i<10; i++)
```

En este ejemplo, usaremos esa segunda forma. De igual modo que dibujamos los premios de forma repetitiva, podemos darle valores al azar de forma repetitiva, en vez de dejar sus valores prefijados. Lo haremos en dos pasos: primero reservaremos espacio

```
int[] xPremio = new int[10];
```

y luego daremos valores:

```
Random generador = new Random();
for (int i=0; i<10; i++)
     xPremio[i] = generador.Next(0,80);
```

Y el fuente completo quedaría:

```
// Primer mini-esqueleto de juego en modo texto
// Versión "g"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego03g
{
     public static void Main()
     {
          ConsoleKeyInfo tecla;
          int x = 40, y=12;
          int xo1 = 20, yo1 = 15;  // Obstáculo 1
          int xo2 = 25, yo2 = 5;   // Obstáculo 2
          int xo3 = 62, yo3 = 21;  // Obstáculo 3
          float xe1 = 10, ye1 = 10, incr1 = 0.5f;  // Enemigo 1
          int fin = 0;   // 0 = no terminado, 1 = terminado
          int[] xPremio = new int[10]; // Premios
          int[] yPremio = new int[10];

          // Genero las posiciones de los premios al azar
          Random generador = new Random();
          for (int i=0; i<10; i++)
          {
               xPremio[i] = generador.Next(0,80);
               yPremio[i] = generador.Next(0,24);
          }

          // Bucle de juego
          while( fin == 0 )
          {
               // Dibujar
               Console.Clear();
               Console.SetCursorPosition(x, y);
```

```
            Console.Write("A");

            Console.SetCursorPosition(xo1, yo1); // Obstáculos
            Console.Write("o");

            Console.SetCursorPosition(xo2, yo2);
            Console.Write("o");

            Console.SetCursorPosition(xo3, yo3);
            Console.Write("o");

            Console.SetCursorPosition((int) xe1, (int) ye1); // Enemigo
            Console.Write("@");

            for (int i=0; i<10; i++)  // Premios
            {
                Console.SetCursorPosition(xPremio[i], yPremio[i]);
                Console.Write("/");
            }

            // Leer teclas y calcular nueva posición
            if (Console.KeyAvailable)
            {
                tecla = Console.ReadKey(false);

                if(tecla.Key == ConsoleKey.RightArrow) x++;
                if(tecla.Key == ConsoleKey.LeftArrow) x--;
                if(tecla.Key == ConsoleKey.DownArrow) y++;
                if(tecla.Key == ConsoleKey.UpArrow) y--;
            }

            // Mover enemigos, entorno
            xe1 = xe1 + incr1;

            if ((xe1 == 0) || (xe1 == 79))
                incr1 = - incr1;

            // Colisiones, perder vidas, etc
            if ( ( (x==xo1) && (y==yo1) )
                || ( (x==xo2) && (y==yo2) )
                || ( (x==xo3) && (y==yo3) )
               )
               fin = 1;

            // Pausa hasta el siguiente "fotograma" del juego
            Thread.Sleep(40);
        }
    }
}
```

Ejercicio propuesto: Mejora el juego para que los datos de los obstáculos también sean parte de un nuevo "array".

13. Pequeñas mejoras

Antes de seguir añadiendo nuevas funcionalidades, vamos a tomarnos un momento para mejorar un poco lo que tenemos hasta ahora:

- Tenemos 3 obstáculos. Vamos a incluirlos en un nuevo array, y de paso, podemos aumentar su cantidad a 20.

•Sólo hay un enemigo. Vamos a ampliarlo también, para que pasen a ser 10 (nuevamente, usando arrays)
•Vamos a hacer que la cantidad de obstáculos, premios y enemigos esté en variables, en vez de prefijado en "números mágicos", para que el programa sea más legible, más fácil de modificar y menos propenso a errores.
•Cuando nuestro personaje pasa por un sitio en el que hay un "premio", deja de verse en pantalla. Más adelante haremos que pueda "recoger" los premios, pero de momento haremos que al menos se vea al personaje, en vez de al premio.
•Vamos a hacer que la condición de fin de partida no sea "1 o 0" sino "verdadero o falso".

Vamos con ello...

Eso de que la condición de fin de partida sea "verdadero" o "falso" supone que la variable "fin" ya no será un número entero, sino un dato "booleano". Tendrá el valor inicial "false" (porque inicialmente no ha terminado la partida), pasará a ser "true" cuando choquemos con algo (porque entonces sí deberá finalizar la partida), y el juego se repetirá mientras no llegue el fin:

```
bool fin = false;
...
while ( ! fin )
{
   ...
```

Lo de añadir un array para los obstáculos no debería ser difícil. La única complicación que añade es la de que ahora la comprobación de colisiones habrá que hacerla de forma repetitiva, por ejemplo con un "for":

```
for (int i=0; i<20; i++)  // Obstáculos
{
     if ((xObstaculo[i] == x)  && (yObstaculo[i] == y))
          chocadoObstaculo = true;
}
```

Si queremos que la cantidad de obstáculos (y la de premios, y la de enemigos) estén en variables, los cambios no son grandes:

```
int numObstaculos = 20;
int[] xObstaculo = new int[numObstaculos];
...

for (int i=0; i<numObstaculos; i++)  // Obstáculos
{
     if ((xObstaculo[i] == x)  && (yObstaculo[i] == y))
          chocadoObstaculo = true;
}
```

Con relación a los enemigos, sólo hay un detalle adicional a tener en cuenta: no deberíamos comparar sus coordenadas directamente, porque son "float", de modo que puede ocurrir que nuestro personaje se encuentre en la posición 50 y un enemigo en la 50.2, que corresponden a la misma posición en pantalla, pero son distintas coordenadas, de modo que no se detectaría como una colisión. Por eso, deberemos convertir sus coordenadas a entero, antes de comparar con la posición del enemigo o con los márgenes de la pantalla:

```
if (( (int) xEnemigo[i] == x) && ( (int) yEnemigo[i] == y))
     fin = true;
```

El resultado, en pantalla, podría ser algo como:

Y todo el fuente junto se podría convertir en:

```
// Primer mini-esqueleto de juego en modo texto
// Versión "h"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego03h
{
     public static void Main()
     {
          int x = 40, y=12; // Posición del personaje
          int numPremios = 10, numEnemigos = 10, numObstaculos = 20;

          // Reservo espacio para los datos repetitivos
          int[] xObstaculo = new int[numObstaculos]; // Obstáculos
          int[] yObstaculo = new int[numObstaculos];

          float[] xEnemigo = new float[numEnemigos]; // Enemigos
          float[] yEnemigo = new float[numEnemigos];
          float[] incr  = new float[numEnemigos];

          int[] xPremio = new int[numPremios]; // Premios
          int[] yPremio = new int[numPremios];

          bool fin = false;
          ConsoleKeyInfo tecla;  // Tecla pulsada

          // Genero las posiciones de los elementos al azar
          Random generador = new Random();
          for (int i=0; i<numObstaculos; i++)  // Obstaculos
          {
               xObstaculo[i] = generador.Next(0,80);
               yObstaculo[i] = generador.Next(0,24);
          }
          for (int i=0; i<numEnemigos; i++)  // Enemigos
          {
               xEnemigo[i] = generador.Next(0,80);
               yEnemigo[i] = generador.Next(0,24);
```

```
        incr[i] = 0.5f;
    }
    for (int i=0; i<numPremios; i++)  // Premios
    {
        xPremio[i] = generador.Next(0,80);
        yPremio[i] = generador.Next(0,24);
    }



    // ------ Bucle de juego ------
    while( ! fin  )
    {
        // -- Dibujar --
        Console.Clear();

        for (int i=0; i<numObstaculos; i++)  // Obstáculos
        {
            Console.SetCursorPosition(
                xObstaculo[i], yObstaculo[i]);
            Console.Write("o");
        }

        for (int i=0; i<numEnemigos; i++)  // Enemigos
        {
            Console.SetCursorPosition(
                (int) xEnemigo[i], (int) yEnemigo[i]);
            Console.Write("@");
        }

        for (int i=0; i<numPremios; i++)  // Premios
        {
            Console.SetCursorPosition(xPremio[i], yPremio[i]);
            Console.Write("/");
        }

        Console.SetCursorPosition(x, y);
        Console.Write("A");


        // -- Leer teclas y calcular nueva posición --
        if (Console.KeyAvailable)
        {
            tecla = Console.ReadKey(false);

            if(tecla.Key == ConsoleKey.RightArrow) x++;
            if(tecla.Key == ConsoleKey.LeftArrow) x--;
            if(tecla.Key == ConsoleKey.DownArrow) y++;
            if(tecla.Key == ConsoleKey.UpArrow) y--;
        }

        // -- Mover enemigos, entorno --
        for (int i=0; i<numEnemigos; i++)  // Enemigos
        {
            xEnemigo[i] = xEnemigo[i] + incr[i];
            if (( (int) xEnemigo[i] == 0)
                    || ( (int) xEnemigo[i] == 79))
                incr[i] = - incr[i];
        }
```

```
            // -- Colisiones, perder vidas, etc --
            for (int i=0; i<numObstaculos; i++)  // Obstáculos
            {
                if ((xObstaculo[i] == x)  && (yObstaculo[i] == y))
                    fin = true;
            }

            for (int i=0; i<numEnemigos; i++)  // Enemigos
            {
                if (( (int) xEnemigo[i] == x)
                        && ( (int) yEnemigo[i] == y))
                    fin = true;
            }

            // -- Pausa hasta el siguiente "fotograma" del juego --
            Thread.Sleep(40);
        }
    }
}
```

Ejercicio propuesto: Haz que los enemigos no puedan aparecer en la misma fila que el personaje, para evitar que se pueda acabar la partida demasiado pronto.

14. Varias vidas

Ahora vamos a hacer que nuestro personaje comience la partida con 3 vidas, y que cada vez que toque un obstáculo o un enemigo pierda una de esas vidas. Cuando llegue a 0 vidas, terminará la partida.

Los cambios serán:

- Añadir un contador de vidas
- Mostrar un marcador, dejando hueco para él (no deberemos mostrar obstáculos, premios ni enemigos en la primera fila de pantalla)
- Cuando haya una colisión (con un obstáculo o un enemigo), todavía no debe acabar la partida. Deberemos restar una vida y comprobar si es la última (empezaremos con 3). Si es la última, terminará el juego, pero si no lo era, habrá que recolocar el personaje en su posición inicial, para que no pierda varias vidas seguidas (porque en el siguiente fotograma seguiría chocando con el mismo obstáculo o quizá incluso con el mismo personaje móvil que provocó que perdiéramos una vida).

Veamos cómo hacerlo...

El contador de vidas es trivial, una nueva variable entera:

```
int vidas = 3;
```

Escribir el marcador no es mucho más difícil: un Write. Si lo hacemos justo después de borrar la pantalla, se escribirá en la posición (0,0), sin necesidad de un SetCursorPosition:

```
Console.Clear();
Console.Write("Vidas: {0}", vidas);
```

Dejar hueco para el marcador apenas es un poco más difícil: cuando generamos coordenadas al azar, no permitiremos que Y valga 0, de modo que la primera fila quede libre:

```
xPremio[i] = generador.Next(0,80);
yPremio[i] = generador.Next(1,24);
```

Y lo de que las colisiones supongan perder sólo una vida (pero quizá no acabar la partida) y recolocar el personaje, también será sencillo:

```
if (( (int) xEnemigo[i] == x)  && ( (int) yEnemigo[i] == y))
{
    vidas --;
    if (vidas == 0)
        fin=true;
```

```
     x = xInicial;
     y = yInicial;
}
```

Lo que supone que será necesario declarar esas nuevas variables auxiliares "xInicial" e "yInicial" al principio el programa:

```
int xInicial = 40, yInicial = 12;
int x=xInicial, y=yInicial; // Posición del personaje
```

El fuente completo sería:

```
// Primer mini-esqueleto de juego en modo texto
// Versión "i"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego03i
{
     public static void Main()
     {
          int vidas = 3;
          int xInicial = 40, yInicial = 12;
          int x=xInicial, y=yInicial; // Posición del personaje

          int numPremios = 10, numEnemigos = 10, numObstaculos = 20;

          // Reservo espacio para los datos repetitivos
          int[] xObstaculo = new int[numObstaculos]; // Obstáculos
          int[] yObstaculo = new int[numObstaculos];

          float[] xEnemigo = new float[numEnemigos]; // Enemigos
          float[] yEnemigo = new float[numEnemigos];
          float[] incr  = new float[numEnemigos];

          int[] xPremio = new int[numPremios]; // Premios
          int[] yPremio = new int[numPremios];

          bool fin = false;
          ConsoleKeyInfo tecla;  // Tecla pulsada

          // Genero las posiciones de los elementos al azar
          Random generador = new Random();
          for (int i=0; i<numObstaculos; i++)  // Obstaculos
          {
               xObstaculo[i] = generador.Next(0,80);
               yObstaculo[i] = generador.Next(1,24);
          }
          for (int i=0; i<numEnemigos; i++)  // Enemigos
          {
               xEnemigo[i] = generador.Next(0,80);
               yEnemigo[i] = generador.Next(1,24);
               incr[i] = 0.5f;
          }
          for (int i=0; i<numPremios; i++)  // Premios
          {
               xPremio[i] = generador.Next(0,80);
               yPremio[i] = generador.Next(1,24);
          }
```

```
// ------ Bucle de juego ------
while( ! fin  )
{
     // -- Dibujar --
     Console.Clear();

     // Marcador
     Console.Write("Vidas: {0}", vidas);

     for (int i=0; i<numObstaculos; i++)  // Obstáculos
     {
          Console.SetCursorPosition(
               xObstaculo[i], yObstaculo[i]);
          Console.Write("o");
     }

     for (int i=0; i<numEnemigos; i++)  // Enemigos
     {
          Console.SetCursorPosition(
               (int) xEnemigo[i], (int) yEnemigo[i]);
          Console.Write("@");
     }

     for (int i=0; i<numPremios; i++)  // Premios
     {
          Console.SetCursorPosition(xPremio[i], yPremio[i]);
          Console.Write("/");
     }

     Console.SetCursorPosition(x, y);
     Console.Write("A");


     // -- Leer teclas y calcular nueva posición --
     if (Console.KeyAvailable)
     {
          tecla = Console.ReadKey(false);

          if(tecla.Key == ConsoleKey.RightArrow) x++;
          if(tecla.Key == ConsoleKey.LeftArrow) x--;
          if(tecla.Key == ConsoleKey.DownArrow) y++;
          if(tecla.Key == ConsoleKey.UpArrow) y--;
     }

     // -- Mover enemigos, entorno --
     for (int i=0; i<numEnemigos; i++)  // Enemigos
     {
          xEnemigo[i] = xEnemigo[i] + incr[i];
          if (( (int) xEnemigo[i] == 0)
                    || ( (int) xEnemigo[i] == 79))
               incr[i] = - incr[i];
     }

     // -- Colisiones, perder vidas, etc --
     for (int i=0; i<numObstaculos; i++)  // Obstáculos
     {
          if ((xObstaculo[i] == x)
```

```
                    && (yObstaculo[i] == y))
                {
                    vidas --;
                    if (vidas == 0)
                        fin=true;
                    x = xInicial;
                    y = yInicial;
                }
            }

            for (int i=0; i<numEnemigos; i++)  // Enemigos
            {
                if (( (int) xEnemigo[i] == x)
                        && ( (int) yEnemigo[i] == y))
                {
                    vidas --;
                    if (vidas == 0)
                        fin=true;
                    x = xInicial;
                    y = yInicial;
                }
            }

            // -- Pausa hasta el siguiente "fotograma" del juego --
            Thread.Sleep(40);
        }
    }
}
```

Ejercicio propuesto: Haz que los obstáculos no puedan aparecer en la misma posición que el personaje, para evitar que la partida acabe de forma prematura y sin motivo aparente.

15. Recoger premios y obtener puntos

Ahora vamos a hacer que nuestro personaje pueda obtener puntos cada vez que toque un "premio"

Supone los siguientes cambios:

- Comprobar colisiones con los premios
- Si se toca uno, aumentarán los puntos (de modo que necesitaremos un contador de puntos)
- El premio tocado deberá dejar de estar visible y no deberán comprobarse colisiones con él a partir de ese momento
- ¿Qué ocurre cuando ya no quedan premios? Podrían volver a aparecer todos, podríamos pasar a otro nivel... pero de momento, en nuestro juego no ocurrirá nada

No es difícil:

Sabemos cómo hacer un contador de puntos

```
int puntos = 0;
```

Sabemos cómo comprobar colisiones básicas y cómo incrementar ese contador:

```
for (int i=0; i<numPremios; i++)  // Premios
{
    if ((xPremio[i] == x)
        && (yPremio[i] == y))
    {
        puntos += 10;
    }
}
```

Para que los premios desaparezcan, usaremos un "array" de "booleanos", que nos diga si cada uno de ellos está visible (con valores iniciales "true"):

```
bool[] premioVisible = new bool[numPremios];
...
for (int i=0; i<numPremios; i++)  // Premios
{
     xPremio[i] = generador.Next(0,80);
     yPremio[i] = generador.Next(1,24);
     premioVisible[i] = true;
}
```

Y cada vez que toquemos uno, deberemos obtener puntos y ocultarlo. Pero hay que tener presente que si ya no está visible, no se deberá comprobar colisiones con él:

```
for (int i=0; i<numPremios; i++)  // Premios
{
     if ((xPremio[i] == x)
          && (yPremio[i] == y)
          && premioVisible[i])
     {
          puntos += 10;
          premioVisible[i] = false;
     }
}
```

Todo junto se convertiría en algo como:

```
// Primer mini-esqueleto de juego en modo texto
// Versión "j"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego03j
{
     public static void Main()
     {
          int vidas = 3;
          int puntos = 0;
          int xInicial = 40, yInicial = 12;
          int x=xInicial, y=yInicial; // Posición del personaje

          int numPremios = 10, numEnemigos = 10, numObstaculos = 20;

          // Reservo espacio para los datos repetitivos
          int[] xObstaculo = new int[numObstaculos]; // Obstáculos
          int[] yObstaculo = new int[numObstaculos];

          float[] xEnemigo = new float[numEnemigos]; // Enemigos
          float[] yEnemigo = new float[numEnemigos];
          float[] incr  = new float[numEnemigos];

          int[] xPremio = new int[numPremios]; // Premios
          int[] yPremio = new int[numPremios];
          bool[] premioVisible = new bool[numPremios];

          bool fin = false;
          ConsoleKeyInfo tecla;  // Tecla pulsada

          // Genero las posiciones de los elementos al azar
          Random generador = new Random();
          for (int i=0; i<numObstaculos; i++)  // Obstaculos
          {
               xObstaculo[i] = generador.Next(0,80);
```

```
        yObstaculo[i] = generador.Next(1,24);
    }
    for (int i=0; i<numEnemigos; i++)  // Enemigos
    {
        xEnemigo[i] = generador.Next(0,80);
        yEnemigo[i] = generador.Next(1,24);
        incr[i] = 0.5f;
    }
    for (int i=0; i<numPremios; i++)  // Premios
    {
        xPremio[i] = generador.Next(0,80);
        yPremio[i] = generador.Next(1,24);
        premioVisible[i] = true;
    }



    // ------ Bucle de juego ------
    while( ! fin  )
    {
        // -- Dibujar --
        Console.Clear();

        // Marcador
        Console.Write("Vidas: {0}  -  Puntos {1}",
            vidas, puntos);

        for (int i=0; i<numObstaculos; i++)  // Obstáculos
        {
            Console.SetCursorPosition(
                xObstaculo[i], yObstaculo[i]);
            Console.Write("o");
        }

        for (int i=0; i<numEnemigos; i++)  // Enemigos
        {
            Console.SetCursorPosition(
                (int) xEnemigo[i], (int) yEnemigo[i]);
            Console.Write("@");
        }

        for (int i=0; i<numPremios; i++)  // Premios
        {
            if (premioVisible[i])
            {
                Console.SetCursorPosition(xPremio[i], yPremio[i]);
                Console.Write("/");
            }
        }

        Console.SetCursorPosition(x, y);
        Console.Write("A");


        // -- Leer teclas y calcular nueva posición --
        if (Console.KeyAvailable)
        {
            tecla = Console.ReadKey(false);
```

```
                if(tecla.Key == ConsoleKey.RightArrow) x++;
                if(tecla.Key == ConsoleKey.LeftArrow) x--;
                if(tecla.Key == ConsoleKey.DownArrow) y++;
                if(tecla.Key == ConsoleKey.UpArrow) y--;
            }

            // -- Mover enemigos, entorno --
            for (int i=0; i<numEnemigos; i++)  // Enemigos
            {
                xEnemigo[i] = xEnemigo[i] + incr[i];
                if (( (int) xEnemigo[i] == 0)
                        || ( (int) xEnemigo[i] == 79))
                    incr[i] = - incr[i];
            }

            // -- Colisiones, perder vidas, etc --
            for (int i=0; i<numObstaculos; i++)  // Obstáculos
            {
                if ((xObstaculo[i] == x)
                    && (yObstaculo[i] == y))
                {
                    vidas --;
                    if (vidas == 0)
                        fin=true;
                    x = xInicial;
                    y = yInicial;
                }
            }

            for (int i=0; i<numEnemigos; i++)  // Enemigos
            {
                if (( (int) xEnemigo[i] == x)
                        && ( (int) yEnemigo[i] == y))
                {
                    vidas --;
                    if (vidas == 0)
                        fin=true;
                    x = xInicial;
                    y = yInicial;
                }
            }

            for (int i=0; i<numPremios; i++)  // Premios
            {
                if ((xPremio[i] == x)
                    && (yPremio[i] == y)
                    && premioVisible[i])
                {
                    puntos += 10;
                    premioVisible[i] = false;
                }
            }

            // -- Pausa hasta el siguiente "fotograma" del juego --
            Thread.Sleep(40);
        }
    }
}
```

Ejercicio propuesto: Haz que los premios que desaparecen vuelvan a aparecer inmediatamente en una nueva posición escogida al azar.

Ejercicio propuesto (2): Haz que los premios desaparezcan uno a uno, y que cuando ya ninguno esté visible, vuelvan a aparecer todos en posiciones escogidas al azar.

16. Agrupando datos: structs

Cada enemigo tiene su coordenada X, su coordenada Y, su velocidad (incremento de X)... pero estos datos son parte de 3 arrays distintos. Parece más natural pensar en el enemigo como un conjunto de esas 3 cosas (y, de paso, alguna más, como su carácter representativo en pantalla), que guardaremos en un "struct". Como los obstáculos, premios y enemigos comparten muchas cosas con un enemigo, podemos crear un tipo de datos "Elemento gráfico" que nos permita guardar las características que nos puedan llegar a interesar de cualquier de ellos:

```
struct ElemGrafico {
     public float x;
     public float y;
     public int xInicial;
     public int yInicial;
     public float incrX;
     public float incrY;
     public char simbolo;
     public bool visible;
}
```

Así, el "personaje" sería un "ElemGrafico":

```
ElemGrafico personaje;
```

Y para los enemigos tendríamos un "array de structs":

```
int numEnemigos = 10;
ElemGrafico[] enemigos = new ElemGrafico[numEnemigos];
```

De igual modo, tendríamos otro array de "elementos gráficos" para los obstáculos y otro para los premios. Cualquiera de ellos lo recorreríamos con un "for" para dar sus valores iniciales:

```
for (int i=0; i<numEnemigos; i++)
{
     enemigos[i].x = generador.Next(0,80);
     enemigos[i].y = generador.Next(1,24);
     enemigos[i].incrX = 0.5f;
     enemigos[i].simbolo = '@';
}
```

Y también para dibujar sus elementos:

```
for (int i=0; i<numEnemigos; i++)
{
     Console.SetCursorPosition(
          (int) enemigos[i].x, (int) enemigos[i].y);
     Console.Write( enemigos[i].simbolo );
}
```

O para comprobar colisiones:

```
for (int i=0; i<numEnemigos; i++)
{
     if (( (int) enemigos[i].x == personaje.x)
               && ( (int) enemigos[i].y == personaje.y))
     {
          vidas --;
          if (vidas == 0)
               fin=true;
          personaje.x = personaje.xInicial;
```

```
            personaje.y = personaje.yInicial;
        }
    }
```

El programa completo no tiene muchos más cambios que esos:

```
// Primer mini-esqueleto de juego en modo texto
// Versión "k"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego03k
{
    struct ElemGrafico {
        public float x;
        public float y;
        public int xInicial;
        public int yInicial;
        public float incrX;
        public float incrY;
        public char simbolo;
        public bool visible;
    }

    public static void Main()
    {
        ElemGrafico personaje;
        personaje.xInicial = 40;
        personaje.yInicial = 12;
        personaje.x = personaje.xInicial;
        personaje.y = personaje.yInicial;
        personaje.simbolo = 'A';

        int numPremios = 10, numEnemigos = 10, numObstaculos = 20;

        // Reservo espacio para los datos repetitivos
        ElemGrafico[] obstaculos = new ElemGrafico[numObstaculos];
        ElemGrafico[] enemigos = new ElemGrafico[numEnemigos];
        ElemGrafico[] premios = new ElemGrafico[numPremios];

        int vidas = 3;
        int puntos = 0;

        bool fin = false;
        ConsoleKeyInfo tecla;  // Tecla pulsada

        // Genero las posiciones de los elementos al azar
        Random generador = new Random();
        for (int i=0; i<numObstaculos; i++)  // Obstaculos
        {
            obstaculos[i].x = generador.Next(0,80);
            obstaculos[i].y = generador.Next(1,24);
            obstaculos[i].simbolo = 'o';
        }

        for (int i=0; i<numEnemigos; i++)  // Enemigos
        {
            enemigos[i].x = generador.Next(0,80);
            enemigos[i].y = generador.Next(1,24);
            enemigos[i].incrX = 0.5f;
```

```
        enemigos[i].simbolo = '@';
    }

    for (int i=0; i<numPremios; i++)  // Premios
    {
        premios[i].x = generador.Next(0,80);
        premios[i].y = generador.Next(1,24);
        premios[i].simbolo = '/';
        premios[i].visible = true;
    }



    // ------ Bucle de juego ------
    while( ! fin  )
    {
        // -- Dibujar --
        Console.Clear();

        // Marcador
        Console.Write("Vidas: {0}  -  Puntos {1}",
            vidas, puntos);

        for (int i=0; i<numObstaculos; i++)  // Obstáculos
        {
            Console.SetCursorPosition(
                (int) obstaculos[i].x, (int) obstaculos[i].y);
            Console.Write( obstaculos[i].simbolo );
        }

        for (int i=0; i<numEnemigos; i++)  // Enemigos
        {
            Console.SetCursorPosition(
                (int) enemigos[i].x, (int) enemigos[i].y);
            Console.Write( enemigos[i].simbolo );
        }

        for (int i=0; i<numPremios; i++)  // Premios
        {
            if (premios[i].visible)
            {
                Console.SetCursorPosition(
                    (int) premios[i].x, (int) premios[i].y);
                Console.Write( premios[i].simbolo );
            }
        }

        Console.SetCursorPosition(
            (int) personaje.x, (int) personaje.y);
        Console.Write( personaje.simbolo );


        // -- Leer teclas y calcular nueva posición --
        if (Console.KeyAvailable)
        {
            tecla = Console.ReadKey(false);

            if(tecla.Key == ConsoleKey.RightArrow) personaje.x++;
            if(tecla.Key == ConsoleKey.LeftArrow) personaje.x--;
```

```
                if(tecla.Key == ConsoleKey.DownArrow) personaje.y++;
                if(tecla.Key == ConsoleKey.UpArrow) personaje.y--;
            }

            // -- Mover enemigos, entorno --
            for (int i=0; i<numEnemigos; i++)  // Enemigos
            {
                enemigos[i].x = enemigos[i].x + enemigos[i].incrX;
                if (( (int) enemigos[i].x == 0)
                          || ( (int) enemigos[i].x == 79))
                    enemigos[i].incrX = - enemigos[i].incrX;
            }

            // -- Colisiones, perder vidas, etc --
            for (int i=0; i<numObstaculos; i++)  // Obstáculos
            {
                if ((obstaculos[i].x == personaje.x)
                    && (obstaculos[i].y == personaje.y))
                {
                    vidas --;
                    if (vidas == 0)
                        fin=true;
                    personaje.x = personaje.xInicial;
                    personaje.y = personaje.yInicial;
                }
            }

            for (int i=0; i<numPremios; i++)  // Obstáculos
            {
                if ((premios[i].x == personaje.x)
                    && (premios[i].y == personaje.y)
                    && premios[i].visible )
                {
                    puntos += 10;
                    premios[i].visible = false;
                }
            }

            for (int i=0; i<numEnemigos; i++)  // Enemigos
            {
                if (( (int) enemigos[i].x == personaje.x)
                          && ( (int) enemigos[i].y == personaje.y))
                {
                    vidas --;
                    if (vidas == 0)
                        fin=true;
                    personaje.x = personaje.xInicial;
                    personaje.y = personaje.yInicial;
                }
            }

            // -- Pausa hasta el siguiente "fotograma" del juego --
            Thread.Sleep(40);
        }
    }
}
```

Ejercicio propuesto: Haz que el personaje no se mueva de 1 en 1 haciendo "x++", sino usando el valor de su campo "incrX".

17. Colores en consola

Dentro de poco estaremos empezando a usar el modo gráfico, que dará una apariencia mucho más agradable a la mayoría de juegos. Aun así, para muchos tipos de juegos sencillos el modo texto puede ser suficiente, pero el poder usar colores le daría un poco más de "gracia". Por eso, vamos a ver cómo cambiar el color del texto y el del fondo.

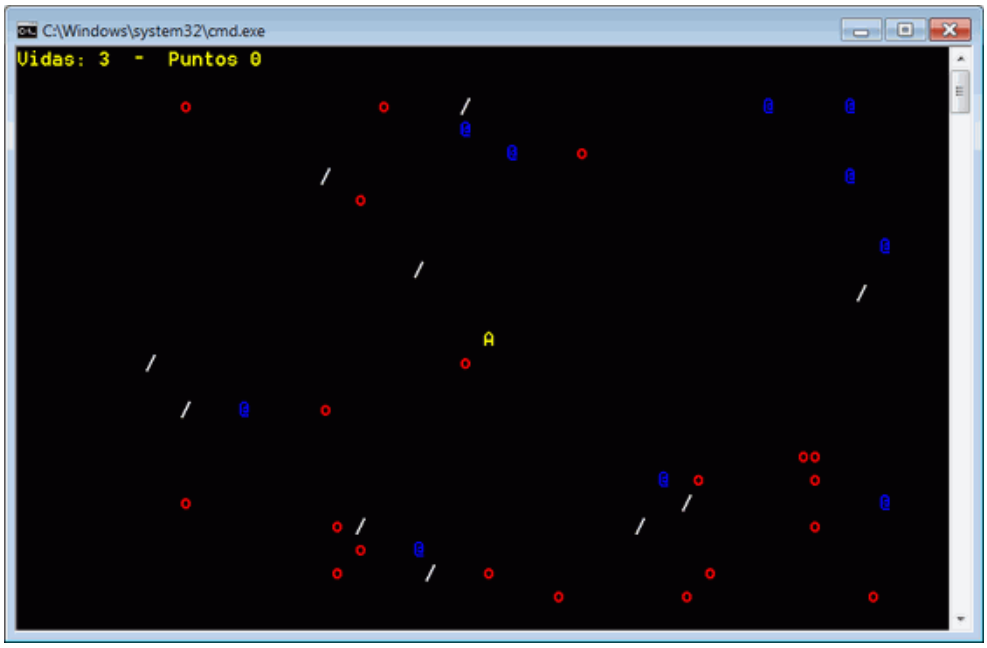
En el "modo texto" (la "consola") de C# tenemos varios colores disponibles. Para ello usaremos el tipo de datos "ConsoleColor", y cambiaremos el color de escritura con "Console.ForegroundColor = ...". Tenemos varios colores predefinidos a partir de su nombre en inglés, como Black, White, Yellow, Red, Blue, DarkRed, Magenta... Así, un ejemplo básico de cómo escribir en colores podría ser:

```
ConsoleColor miColor;
miColor = ConsoleColor.Red;

Console.ForegroundColor = miColor;
Console.Write("Hola");
```

En nuestro fuente, esto supone 3 cambios:

- Tendremos un nuevo campo en nuestro "struct", que será el color de cada elemento gráfico.
- Cuando damos los valores iniciales a nuestros "structs", tendremos que escoger cuál sera el color para ese elemento.
- Justo antes de "escribir" cada elemento gráfico en pantalla, deberemos cambiar el color de escritura, para que pase a ser al color que tiene ese elemento.

No son grandes cambios. La apariencia del juego podría ser algo como:



Y el fuente podría quedar así:

```
// Primer mini-esqueleto de juego en modo texto
// Versión "1"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego031
{
    struct ElemGrafico {
        public float x;
        public float y;
        public int xInicial;
        public int yInicial;
        public float incrX;
        public float incrY;
```

```
    public char simbolo;
    public ConsoleColor color;
    public bool visible;
}

public static void Main()
{
    ElemGrafico personaje;
    personaje.xInicial = 40;
    personaje.yInicial = 12;
    personaje.x = personaje.xInicial;
    personaje.y = personaje.yInicial;
    personaje.simbolo = 'A';
    personaje.color = ConsoleColor.Yellow;

    int numPremios = 10, numEnemigos = 10, numObstaculos = 20;

    // Reservo espacio para los datos repetitivos
    ElemGrafico[] obstaculos = new ElemGrafico[numObstaculos];
    ElemGrafico[] enemigos = new ElemGrafico[numEnemigos];
    ElemGrafico[] premios = new ElemGrafico[numPremios];

    int vidas = 3;
    int puntos = 0;

    bool fin = false;
    ConsoleKeyInfo tecla;  // Tecla pulsada

    // Genero las posiciones de los elementos al azar
    Random generador = new Random();
    for (int i=0; i<numObstaculos; i++)  // Obstaculos
    {
        obstaculos[i].x = generador.Next(0,80);
        obstaculos[i].y = generador.Next(1,24);
        obstaculos[i].simbolo = 'o';
        obstaculos[i].color = ConsoleColor.Red;
    }

    for (int i=0; i<numEnemigos; i++)  // Enemigos
    {
        enemigos[i].x = generador.Next(0,80);
        enemigos[i].y = generador.Next(1,24);
        enemigos[i].incrX = 0.5f;
        enemigos[i].simbolo = '@';
        enemigos[i].color = ConsoleColor.Blue;
    }

    for (int i=0; i<numPremios; i++)  // Premios
    {
        premios[i].x = generador.Next(0,80);
        premios[i].y = generador.Next(1,24);
        premios[i].simbolo = '/';
        premios[i].color = ConsoleColor.White;
        premios[i].visible = true;
    }



    // ------ Bucle de juego ------
```

```
while( ! fin  )
{
     // -- Dibujar --
     Console.Clear();

     // Marcador
     Console.Write("Vidas: {0}  -  Puntos {1}",
          vidas, puntos);

     for (int i=0; i<numObstaculos; i++)  // Obstáculos
     {
          Console.SetCursorPosition(
               (int) obstaculos[i].x, (int) obstaculos[i].y);
          Console.ForegroundColor =  obstaculos[i].color;
          Console.Write( obstaculos[i].simbolo );
     }

     for (int i=0; i<numEnemigos; i++)  // Enemigos
     {
          Console.SetCursorPosition(
               (int) enemigos[i].x, (int) enemigos[i].y);
          Console.ForegroundColor =  enemigos[i].color;
          Console.Write( enemigos[i].simbolo );
     }

     for (int i=0; i<numPremios; i++)  // Premios
     {
          if (premios[i].visible)
          {
               Console.SetCursorPosition(
                    (int) premios[i].x, (int) premios[i].y);
               Console.ForegroundColor =  premios[i].color;
               Console.Write( premios[i].simbolo );
          }
     }

     Console.SetCursorPosition(
          (int) personaje.x, (int) personaje.y);
     Console.ForegroundColor =  personaje.color;
     Console.Write( personaje.simbolo );


     // -- Leer teclas y calcular nueva posición --
     if (Console.KeyAvailable)
     {
          tecla = Console.ReadKey(false);

          if(tecla.Key == ConsoleKey.RightArrow) personaje.x++;
          if(tecla.Key == ConsoleKey.LeftArrow) personaje.x--;
          if(tecla.Key == ConsoleKey.DownArrow) personaje.y++;
          if(tecla.Key == ConsoleKey.UpArrow) personaje.y--;
     }

     // -- Mover enemigos, entorno --
     for (int i=0; i<numEnemigos; i++)  // Enemigos
     {
          enemigos[i].x = enemigos[i].x + enemigos[i].incrX;
          if (( (int) enemigos[i].x == 0)
                    || ( (int) enemigos[i].x == 79))
```

```
                    enemigos[i].incrX = - enemigos[i].incrX;
               }

               // -- Colisiones, perder vidas, etc --
               for (int i=0; i<numObstaculos; i++)  // Obstáculos
               {
                    if ((obstaculos[i].x == personaje.x)
                         && (obstaculos[i].y == personaje.y))
                    {
                         vidas --;
                         if (vidas == 0)
                              fin=true;
                         personaje.x = personaje.xInicial;
                         personaje.y = personaje.yInicial;
                    }
               }

               for (int i=0; i<numPremios; i++)  // Obstáculos
               {
                    if ((premios[i].x == personaje.x)
                         && (premios[i].y == personaje.y)
                         && premios[i].visible )
                    {
                         puntos += 10;
                         premios[i].visible = false;
                    }
               }

               for (int i=0; i<numEnemigos; i++)  // Enemigos
               {
                    if (( (int) enemigos[i].x == personaje.x)
                              && ( (int) enemigos[i].y == personaje.y))
                    {
                         vidas --;
                         if (vidas == 0)
                              fin=true;
                         personaje.x = personaje.xInicial;
                         personaje.y = personaje.yInicial;
                    }
               }

               // -- Pausa hasta el siguiente "fotograma" del juego --
               Thread.Sleep(40);
          }
     }
}
```

Ejercicio propuesto: Haz que haya enemigos de dos colores distintos y con dos velocidades ligeramente distintas.

18. Presentación. Poder volver a jugar

Ahora mismo, cuando termina la partida, también termina todo el juego. Pero esto no es lo habitual: en un "juego real suele haber una pantalla de presentación previa, y tras terminar una partida se vuelve a esa pantalla de presentación y se puede comenzar una nueva partida.

Vamos a añadir esa posibilidad a nuestro juego...

No debería ser especialmente complicado:

•Ahora el "bucle de juego" (que representa el transcurso de una partida) deberá estar rodeado por un nuevo "do...while", que represente toda una sesión de juego.
•Antes de que comience el bucle de juego, se mostrará una pantalla que dé al usuario una serie de opciones (como mínimo, la de comenzar una nueva partida y la de abandonar el programa).
•Esa parte repetitiva debe incluir también las tareas que se deben volver a realizar cada vez antes de cada nueva "partida", como dejar las vidas en su valor inicial (3), los puntos también con su valor inicial (0), todos los premios visibles, volver a generar posiciones al azar para todos los elementos, etc.

```
// Primer mini-esqueleto de juego en modo texto
// Versión "m"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego03m
{
    struct ElemGrafico {
        public float x;
        public float y;
        public int xInicial;
        public int yInicial;
        public float incrX;
        public float incrY;
        public char simbolo;
        public ConsoleColor color;
        public bool visible;
    }

    public static void Main()
    {
        ElemGrafico personaje;
        personaje.xInicial = 40;
        personaje.yInicial = 12;
        personaje.x = personaje.xInicial;
        personaje.y = personaje.yInicial;
        personaje.simbolo = 'A';
        personaje.color = ConsoleColor.Yellow;

        int numPremios = 10, numEnemigos = 10, numObstaculos = 20;

        // Reservo espacio para los datos repetitivos
        ElemGrafico[] obstaculos = new ElemGrafico[numObstaculos];
        ElemGrafico[] enemigos = new ElemGrafico[numEnemigos];
        ElemGrafico[] premios = new ElemGrafico[numPremios];

        bool juegoTerminado = false;
        int vidas;
        int puntos;
        bool partidaTerminada;
        ConsoleKeyInfo tecla;  // Tecla pulsada
        Random generador = new Random();

        while (! juegoTerminado )
        {

            // En cada partida, hay que reiniciar ciertas variables
            vidas = 3;
            puntos = 0;
            partidaTerminada = false;

            // Genero las posiciones de los elementos al azar
            for (int i=0; i<numObstaculos; i++)  // Obstaculos
```

```
{
     obstaculos[i].x = generador.Next(0,80);
     obstaculos[i].y = generador.Next(1,24);
     obstaculos[i].simbolo = 'o';
     obstaculos[i].color = ConsoleColor.Red;
}

for (int i=0; i<numEnemigos; i++)  // Enemigos
{
     enemigos[i].x = generador.Next(0,80);
     enemigos[i].y = generador.Next(1,24);
     enemigos[i].incrX = 0.5f;
     enemigos[i].simbolo = '@';
     enemigos[i].color = ConsoleColor.Blue;
}

for (int i=0; i<numPremios; i++)  // Premios
{
     premios[i].x = generador.Next(0,80);
     premios[i].y = generador.Next(1,24);
     premios[i].simbolo = '/';
     premios[i].color = ConsoleColor.White;
     premios[i].visible = true;
}

// ---- Pantalla de presentación ----
Console.Clear();
Console.ForegroundColor = ConsoleColor.White;
Console.SetCursorPosition(34,10);
Console.Write("Jueguecillo");
Console.SetCursorPosition(31,12);
Console.Write("Escoja una opción:");
Console.SetCursorPosition(15,17);
Console.Write("1.- Jugar una partida");
Console.SetCursorPosition(15,19);
Console.Write("0.- Salir ");
tecla = Console.ReadKey(false);
if (tecla.KeyChar == '0')
{
     partidaTerminada = true;
     juegoTerminado = true;
}



// ------ Bucle de juego ------
while( ! partidaTerminada  )
{
     // -- Dibujar --
     Console.Clear();

     // Marcador
     Console.Write("Vidas: {0}  -  Puntos {1}",
          vidas, puntos);

     for (int i=0; i<numObstaculos; i++)  // Obstáculos
     {
          Console.SetCursorPosition(
               (int) obstaculos[i].x, (int) obstaculos[i].y);
```

```
     Console.ForegroundColor =  obstaculos[i].color;
     Console.Write( obstaculos[i].simbolo );
}

for (int i=0; i<numEnemigos; i++)  // Enemigos
{
     Console.SetCursorPosition(
          (int) enemigos[i].x, (int) enemigos[i].y);
     Console.ForegroundColor =  enemigos[i].color;
     Console.Write( enemigos[i].simbolo );
}

for (int i=0; i<numPremios; i++)  // Premios
{
     if (premios[i].visible)
     {
          Console.SetCursorPosition(
               (int) premios[i].x, (int) premios[i].y);
          Console.ForegroundColor =  premios[i].color;
          Console.Write( premios[i].simbolo );
     }
}

Console.SetCursorPosition(
     (int) personaje.x, (int) personaje.y);
Console.ForegroundColor =  personaje.color;
Console.Write( personaje.simbolo );


// -- Leer teclas y calcular nueva posición --
if (Console.KeyAvailable)
{
     tecla = Console.ReadKey(false);

     if(tecla.Key == ConsoleKey.RightArrow) personaje.x++;
     if(tecla.Key == ConsoleKey.LeftArrow) personaje.x--;
     if(tecla.Key == ConsoleKey.DownArrow) personaje.y++;
     if(tecla.Key == ConsoleKey.UpArrow) personaje.y--;
}

// -- Mover enemigos, entorno --
for (int i=0; i<numEnemigos; i++)  // Enemigos
{
     enemigos[i].x = enemigos[i].x + enemigos[i].incrX;
     if (( (int) enemigos[i].x == 0)
               || ( (int) enemigos[i].x == 79))
          enemigos[i].incrX = - enemigos[i].incrX;
}

// -- Colisiones, perder vidas, etc --
for (int i=0; i<numObstaculos; i++)  // Obstáculos
{
     if ((obstaculos[i].x == personaje.x)
          && (obstaculos[i].y == personaje.y))
     {
          vidas --;
          if (vidas == 0)
               partidaTerminada=true;
          personaje.x = personaje.xInicial;
```

```
                    personaje.y = personaje.yInicial;
                }
            }

            for (int i=0; i<numPremios; i++)  // Obstáculos
            {
                if ((premios[i].x == personaje.x)
                    && (premios[i].y == personaje.y)
                    && premios[i].visible )
                {
                    puntos += 10;
                    premios[i].visible = false;
                }
            }

            for (int i=0; i<numEnemigos; i++)  // Enemigos
            {
                if (( (int) enemigos[i].x == personaje.x)
                        && ( (int) enemigos[i].y == personaje.y))
                {
                    vidas --;
                    if (vidas == 0)
                        partidaTerminada=true;
                    personaje.x = personaje.xInicial;
                    personaje.y = personaje.yInicial;
                }
            }

            // -- Pausa hasta el siguiente "fotograma" del juego --
            Thread.Sleep(40);

        } // Fin del bucle de juego

    }     // Fin de partida

}         // Fin de Main
}
```

Ejercicio propuesto: Añade al menú una nueva opción que permita mostrar una ayuda al usuario, que le diga qué debe hacer en el juego (recoger premios, esquivar obstáculos y enemigos, etc).

Ejercicio propuesto (2): Añade una opción que "memorice" la puntuación más alta conseguida durante una sesión de juego (varias partidas consecutivas).

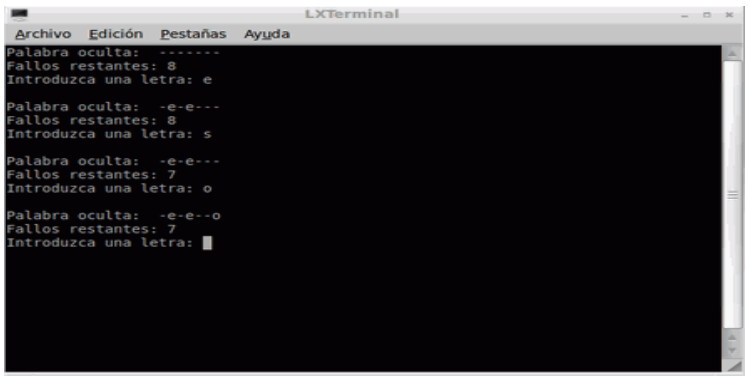
19. Ahorcado en modo texto

Antes de hacer que nuestro "juego de plataformas en modo texto" sea un poco más modular, descomponiéndolo en funciones, y ya que hemos empezado a manejar "arrays" y "structs", vamos a cambiar momentáneamente de juego para practicar el uso de "arrays" y de "cadenas de texto", creando un "juego del ahorcado".

La lógica básica del juego será la siguiente:

1. El ordenador escoge una palabra al azar (de entre un conjunto prefijado)
2. La palabra se muestra como una serie de guiones: ---- ----
3. El usuario elige una letra
4. Si la letra no es parte de la palabra, el jugador pierde un intento (de los ocho iniciales)
5. Si la letra es parte de la palabra, el jugador no pierde ningún intento, y la letra se muestra como parte de la palabra oculta: --a- -a--
6. Esta estructura se repite desde el punto 2, hasta que el usuario se queda sin intentos o adivina toda la palabra.

La apariencia debería ser algo como:



¿Cómo se puede conseguir? Algunas de esas tareas son simples, pero también hay alguna un poco más enrevesada:

1.Para escoger una palabra al azar de entre varias, basta con guardarlas en un "array de strings" y escoger una de ellas con la ayuda de un "Random".
2.Para mostrar la palabra como una serie de guiones (---- ----) podemos volcar de forma repetitiva en una nueva cadena de texto, recorriendo letra por letra, y añadiendo un espacio si la frase a adivinar contiene algún espacio, o añadiendo un punto en caso de que sea cualquier otra letra.
3.Lo de que el usuario elija una letra debería ser trivial a estas alturas...
4.Para ver si la letra no es parte de la palabra, podemos usar "IndexOf"
5.Para ir añadiendo letras a la palabra oculta (--a- -a--), como en C# no se pueden modificar directamente letras de una cadena, podemos ir volcando a una nueva cadena, de forma parecida a lo que hicimos en el punto 2: en una cadena auxiliar vamos guardando las letras de la frase que estábamos mostrando en pantalla, o la letra indicada por el usuario en caso de que coincida.
6.Finalmente, para saber el final de la partida: lo de que el usuario se quede sin intentos es fácil si estamos llevando un contador de errores; lo de si adivina toda la palabra tampoco es difícil: podemos comprobar si ya no quedan guiones en el texto que se muestra en pantalla, o si la palabra que había que adivinar coincide con ese texto que se muestra en pantalla.

En forma de programa quedaría:

```
// Juego del ahorcado en modo texto

using System;

public class Ahorcado
{

  public static void Main()
  {

      // 1: El ordenador escoge una palabra al azar (de entre un conjunto prefijado)

      string[] palabras = { "memento", "krull", "spiderman",
        "star wars", "hulk", "batman" };
      Random generadorAleatorio = new Random();
      int numeroAzar = generadorAleatorio.Next(0,palabras.Length);
      string palabraAdivinar = palabras[ numeroAzar ];

      // 1b: La palabra se muestra como serie de guiones: ---- ----
      string palabraMostrar = "";
      for (int i=0; i< palabraAdivinar.Length; i++)
          if (palabraAdivinar[i] == ' ')
              palabraMostrar += " ";
          else
              palabraMostrar += "-";


      // Otras variables
      int fallosRestantes = 8;
      char letraActual;
      bool terminado = false;
```

```
        // Parte repetitiva
        do
        {

            // 2: Se muestra la palabra oculta y el usuario elige una letra
            Console.WriteLine( "Palabra oculta:  {0}", palabraMostrar);
            Console.WriteLine( "Fallos restantes: {0}", fallosRestantes);

            // 3: El usuario elige una letra
            Console.Write( "Introduzca una letra: ");
            letraActual = Convert.ToChar( Console.ReadLine() );

            // 4: Si la letra no es parte de la palabra, el jugador
            // pierde un intento (de los ocho iniciales)
            if( palabraAdivinar.IndexOf( letraActual ) == -1 )
                fallosRestantes--;

            // 5: Si la letra es parte de la palabra, el jugador no
            // pierde ningún intento, y la letra se muestra como
            // parte de la palabra oculta: --a- -a--
            string siguienteMostrar = "";

            for( int i = 0; i < palabraAdivinar.Length; i++)
            {
                if( letraActual == palabraAdivinar[i] )
                    siguienteMostrar += letraActual;
                else
                    siguienteMostrar += palabraMostrar[i];
            }
            palabraMostrar = siguienteMostrar;

            // 6: Comprobar si ha terminado: si el usuario se queda sin intentos
            // o adivina toda la palabra.
            if( palabraMostrar.IndexOf("-") < 0 )
            {
                Console.WriteLine("Felicidades, acertaste!  ({0})",
                   palabraAdivinar);
                terminado = true;
            }

            if( fallosRestantes == 0 )
            {
                Console.WriteLine("Lo siento. Era {0}", palabraAdivinar);
                terminado = true;
            }

            Console.WriteLine();
        }
        while ( ! terminado );

    }
}
```

Ejercicio propuesto: Haz que este juego se comporte correctamente aunque la palabra tenga letras en mayúsculas o minúsculas, y aunque el usuario introduzca las letras en mayúsculas o en minúsculas.

Nota: también se puede modificar un poco, para practicar el uso de cadenas modificables (StringBuilder) y de "foreach", de forma que el paso 5 queda simplificado:

```
// Juego del ahorcado en modo texto

using System;
using System.Text; // Para "StringBuilder"

public class Ahorcado
{

   public static void Main()
   {

        // 1: El ordenador escoge una palabra al azar (de entre un conjunto prefijado)

        string[] palabras = { "memento", "krull", "spiderman",
          "star wars", "hulk", "batman" };
        Random generadorAleatorio = new Random();
        int numeroAzar = generadorAleatorio.Next(0,palabras.Length);
        string palabraAdivinar = palabras[ numeroAzar ];

        // 1b: La palabra se muestra como serie de guiones: ---- ----
        StringBuilder palabraMostrar =  new StringBuilder();
        foreach (char letra in palabraAdivinar)
          if (letra == ' ')
               palabraMostrar.Append(" ");
          else
               palabraMostrar.Append("-");

        // Otras variables
        int fallosRestantes = 8;
        char letraActual;
        bool terminado = false;

        // Parte repetitiva
        do
        {

             // 2: Se muestra la palabra oculta y el usuario elige una letra
             Console.WriteLine( "Palabra oculta:  {0}", palabraMostrar);
             Console.WriteLine( "Fallos restantes: {0}", fallosRestantes);

             // 3: El usuario elige una letra
             Console.Write( "Introduzca una letra: ");
             letraActual = Convert.ToChar( Console.ReadLine() );

             // 4: Si la letra no es parte de la palabra, el jugador
             // pierde un intento (de los ocho iniciales)
             if( palabraAdivinar.IndexOf( letraActual ) == -1 )
                  fallosRestantes--;

             // 5: Si la letra es parte de la palabra, el jugador no
             // pierde ningún intento, y la letra se muestra como
             // parte de la palabra oculta: --a- -a--
             for( int i = 0; i < palabraAdivinar.Length; i++)
                  if( letraActual == palabraAdivinar[i] )
                       palabraMostrar[i]= letraActual;

             // 6: Comprobar si ha terminado: si el usuario se queda sin intentos
             // o adivina toda la palabra.
             if( palabraMostrar.ToString().IndexOf("-") < 0 )
             {
```

```
                Console.WriteLine("Felicidades, acertaste!  ({0})",
                    palabraAdivinar);
                terminado = true;
            }

            if( fallosRestantes == 0 )
            {
                Console.WriteLine("Lo siento. Era {0}", palabraAdivinar);
                terminado = true;
            }

            Console.WriteLine();
        }
        while ( ! terminado );

    }
}
```

20. Descomponiendo en funciones

Nuestro "Main" ya ocupa más de 200 líneas (más de 3 páginas de papel), a pesar de tratarse de un juego muy simple. Si seguimos ampliando de esta manera, pronto llegará un momento en que sea difícil encontrar dónde se encuentra cada cosa, lo que dificultará ampliar y hacer correcciones.

Por eso, va siendo el momento de descomponer el programa en bloques más pequeños, cada uno de los cuales tenga una misión más concreta.

El primer objetivo es llegar a un Main mucho más compacto y más facil de leer, algo como:

```
public static void Main()
{

    InicializarJuego();

    while (! juegoTerminado )
    {
        InicializarPartida();
        MostrarPresentacion();

        // ------ Bucle de juego ------
        while( ! partidaTerminada  )
        {
            Dibujar();
            ComprobarTeclas();
            MoverElementos();
            ComprobarColisiones();
            PausaFotograma();
        } // Fin del bucle de juego
    }     // Fin de partida
}
```

En una segunda etapa, nos permitirá simplificar tareas repetitivas, pero eso llegará un poco más adelante.

Los pasos que seguiremos para hacer el programa más modular serán los siguientes

- Pensar en las "tareas" que realmente componen Main
- Crear un "bloque" (una función) para cada una de esas tareas. Por ahora, para nosotros, el nombre cada una de esas tareas será típicamente una acción (como "ComprobarTeclas"), y cuando creemos el bloque, éste tendrá una apariencia muy similar a la de Main: comenzará con "public static void" y terminará con paréntesis vacíos.
- Finalmente, las variables que se vayan a compartir entre varias funciones, deberán estar fuera de Main y antes de todas esas funciones, y deberán estar precedidas (por ahora) de la palabra "static".

Debería ser un poco trabajoso, pero no especialmente difícil. El resultado sería algo como:

```
// Primer mini-esqueleto de juego en modo texto
// Versión "n"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego03n
{
     struct ElemGrafico {
          public float x;
          public float y;
          public int xInicial;
          public int yInicial;
          public float incrX;
          public float incrY;
          public char simbolo;
          public ConsoleColor color;
          public bool visible;
     }

     static ElemGrafico personaje;


     static int numPremios, numEnemigos, numObstaculos;

     static ElemGrafico[] obstaculos;
     static ElemGrafico[] enemigos;
     static ElemGrafico[] premios;

     static bool juegoTerminado;
     static int vidas;
     static int puntos;
     static bool partidaTerminada;
     static ConsoleKeyInfo tecla;  // Tecla pulsada
     static Random generador;


     public static void InicializarJuego()
     {
          juegoTerminado = false;
          numPremios = 10;
          numEnemigos = 10;
          numObstaculos = 20;
          obstaculos = new ElemGrafico[numObstaculos];
          enemigos = new ElemGrafico[numEnemigos];
          premios = new ElemGrafico[numPremios];
          generador = new Random();
     }


     public static void InicializarPartida()
     {
          // En cada partida, hay que reiniciar ciertas variables
          vidas = 3;
          puntos = 0;
          partidaTerminada = false;

          personaje.xInicial = 40;
          personaje.yInicial = 12;
```

```
        personaje.x = personaje.xInicial;
        personaje.y = personaje.yInicial;
        personaje.simbolo = 'A';
        personaje.color = ConsoleColor.Yellow;

        // Genero las posiciones de los elementos al azar
        for (int i=0; i<numObstaculos; i++)  // Obstaculos
        {
            obstaculos[i].x = generador.Next(0,80);
            obstaculos[i].y = generador.Next(1,24);
            obstaculos[i].simbolo = 'o';
            obstaculos[i].color = ConsoleColor.Red;
        }

        for (int i=0; i<numEnemigos; i++)  // Enemigos
        {
            enemigos[i].x = generador.Next(0,80);
            enemigos[i].y = generador.Next(1,24);
            enemigos[i].incrX = 0.5f;
            enemigos[i].simbolo = '@';
            enemigos[i].color = ConsoleColor.Blue;
        }

        for (int i=0; i<numPremios; i++)  // Premios
        {
            premios[i].x = generador.Next(0,80);
            premios[i].y = generador.Next(1,24);
            premios[i].simbolo = '/';
            premios[i].color = ConsoleColor.White;
            premios[i].visible = true;
        }
    }


    public static void MostrarPresentacion()
    {
        // ---- Pantalla de presentación ----
        Console.Clear();
        Console.ForegroundColor = ConsoleColor.White;
        Console.SetCursorPosition(34,10);
        Console.Write("Jueguecillo");
        Console.SetCursorPosition(31,12);
        Console.Write("Escoja una opción:");
        Console.SetCursorPosition(15,17);
        Console.Write("1.- Jugar una partida");
        Console.SetCursorPosition(15,19);
        Console.Write("0.- Salir ");

        tecla = Console.ReadKey(false);
        if (tecla.KeyChar == '0')
        {
            partidaTerminada = true;
            juegoTerminado = true;
        }
    }

    public static void Dibujar()
    {
        // -- Dibujar --
```

```
        Console.Clear();

        // Marcador
        Console.Write("Vidas: {0}  -  Puntos {1}",
            vidas, puntos);

        for (int i=0; i<numObstaculos; i++)  // Obstáculos
        {
            Console.SetCursorPosition(
                (int) obstaculos[i].x, (int) obstaculos[i].y);
            Console.ForegroundColor =  obstaculos[i].color;
            Console.Write( obstaculos[i].simbolo );
        }

        for (int i=0; i<numEnemigos; i++)  // Enemigos
        {
            Console.SetCursorPosition(
                (int) enemigos[i].x, (int) enemigos[i].y);
            Console.ForegroundColor =  enemigos[i].color;
            Console.Write( enemigos[i].simbolo );
        }

        for (int i=0; i<numPremios; i++)  // Premios
        {
            if (premios[i].visible)
            {
                Console.SetCursorPosition(
                    (int) premios[i].x, (int) premios[i].y);
                Console.ForegroundColor =  premios[i].color;
                Console.Write( premios[i].simbolo );
            }
        }

        Console.SetCursorPosition(
            (int) personaje.x, (int) personaje.y);
        Console.ForegroundColor =  personaje.color;
        Console.Write( personaje.simbolo );
    }


    public static void ComprobarTeclas()
    {
        // -- Leer teclas y calcular nueva posición --
        if (Console.KeyAvailable)
        {
            tecla = Console.ReadKey(false);

            if(tecla.Key == ConsoleKey.RightArrow) personaje.x++;
            if(tecla.Key == ConsoleKey.LeftArrow) personaje.x--;
            if(tecla.Key == ConsoleKey.DownArrow) personaje.y++;
            if(tecla.Key == ConsoleKey.UpArrow) personaje.y--;
        }

    }


    public static void MoverElementos()
    {
        // -- Mover enemigos, entorno --
```

```
     for (int i=0; i<numEnemigos; i++)  // Enemigos
     {
          enemigos[i].x = enemigos[i].x + enemigos[i].incrX;
          if (( (int) enemigos[i].x == 0)
                    || ( (int) enemigos[i].x == 79))
               enemigos[i].incrX = - enemigos[i].incrX;
     }
}


public static void ComprobarColisiones()
{
     // -- Colisiones, perder vidas, etc --
     for (int i=0; i<numObstaculos; i++)  // Obstáculos
     {
          if ((obstaculos[i].x == personaje.x)
               && (obstaculos[i].y == personaje.y))
          {
               vidas --;
               if (vidas == 0)
                    partidaTerminada=true;
               personaje.x = personaje.xInicial;
               personaje.y = personaje.yInicial;
          }
     }

     for (int i=0; i<numPremios; i++)  // Obstáculos
     {
          if ((premios[i].x == personaje.x)
               && (premios[i].y == personaje.y)
               && premios[i].visible )
          {
               puntos += 10;
               premios[i].visible = false;
          }
     }

     for (int i=0; i<numEnemigos; i++)  // Enemigos
     {
          if (( (int) enemigos[i].x == personaje.x)
                    && ( (int) enemigos[i].y == personaje.y))
          {
               vidas --;
               if (vidas == 0)
                    partidaTerminada=true;
               personaje.x = personaje.xInicial;
               personaje.y = personaje.yInicial;
          }
     }

}


public static void PausaFotograma()
{
     // -- Pausa hasta el siguiente "fotograma" del juego --
     Thread.Sleep(40);
}
```

```
    public static void Main()
    {

        InicializarJuego();

        while (! juegoTerminado )
        {
            InicializarPartida();
            MostrarPresentacion();

            // ------ Bucle de juego ------
            while( ! partidaTerminada  )
            {
                Dibujar();
                ComprobarTeclas();
                MoverElementos();
                ComprobarColisiones();
                PausaFotograma();
            } // Fin del bucle de juego
        }     // Fin de partida
    }         // Fin de Main
}
```

Ejercicio propuesto: Como ésta es la última versión que haremos en modo texto, complétala para que sea realmente jugable (la mayoría de estas mejoras ya se habían propuesto en apartados anteriores):

- Que el personaje no se pueda salir de la pantalla (porque además eso provoca que la ejecución del juego se detenga).
- Que cuando se recojan todos los premios, éstos vuelvan a aparecer y aumente la dificultad (por ejemplo, se incremente la velocidad de los enemigos o aparezca un nuevo enemigo).
- Que los obstáculos y los premios no puedan aparecer en la misma posición inicial que el personaje.
- Que los enemigos no puedan aparecer en la misma coordenada Y que el personaje, para evitar que nos puedan "cazar" nada más comenzar una partida.
- Que se memorice la puntuación más alta obtenida en una sesión de juego.

21. Contacto con el modo gráfico

El "modo texto" nos ha permitido practicar ciertos conceptos básicos de un modo sencillo. Pero es "demasiado poco vistoso" para la mayoría de juegos reales. Por eso, vamos a ver cómo mostrar imágenes en pantalla.

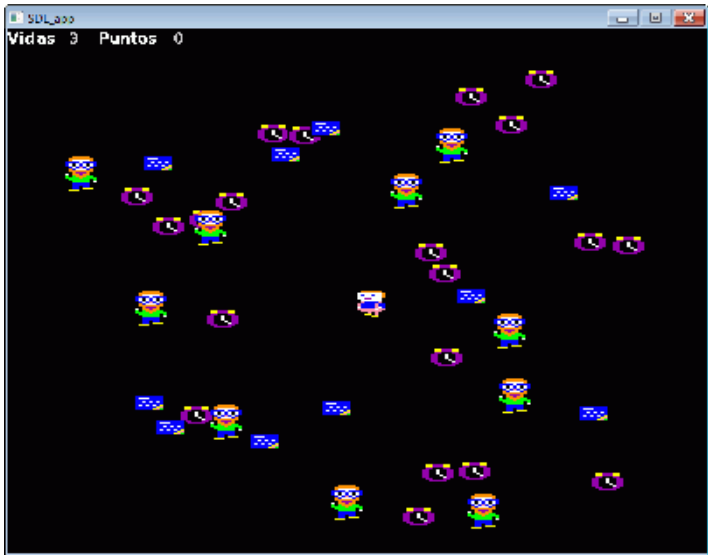
Usaremos "de fondo" la biblioteca gráfica SDL, pero la "ocultaremos" para poder esquivar la mayoría de sus detalles internos, y centrarnos en la forma de realizar las tareas habituales, como dibujar imágenes, comprobar colisiones entre ellas, conseguir movimientos animados, evitar parpadeos al dibujar en pantalla, etc.

Así, la apariencia de "Main" será idéntica a la de la versión en modo texto:

```
public static void Main()
{

    InicializarJuego();

    while (! juegoTerminado )
    {
        InicializarPartida();
        MostrarPresentacion();

        // ------ Bucle de juego ------
        while( ! partidaTerminada  )
        {
            Dibujar();
```

```
                ComprobarTeclas();
                MoverElementos();
                ComprobarColisiones();
                PausaFotograma();
            } // Fin del bucle de juego
        }     // Fin de partida
}
```

Pero el resultado en pantalla será mucho más vistoso que para la versión en modo texto , incluso si queremos conservar una cierta "estética retro":



En el fuente, las funciones que tienen que acceder a la consola sí cambiarán, porque ya no usaremos letras sino imágenes, y el manejo de teclado también será ligeramente distinto.

Por ejemplo, para dibujar un enemigo en pantalla, ya no usaremos un "SetCursorPosition" seguido de un "Write", sino una única orden que se encargará de mostrar una imagen en ciertas coordenadas de pantalla:

```
personaje.imagen.DibujarOculta(personaje.x, personaje.y);
```

Para escribir textos, usaremos una única función "EscribirTexto" que recibirá como detalles el texto a escribir (sólo un "string", no permitirá el uso de "{0}"), las coordenadas, el color (como 3 cifras R, G, B para detallar la cantidad de rojo, verde y azul) y el tipo de letra (porque podremos usar distintos tipos de letra en un mismo juego):

```
// Marcador
Hardware.EscribirTextoOculta("Vidas",
  0,0, // Coordenadas
  255, 255, 255, // Colores
  tipoDeLetra);

Hardware.EscribirTextoOculta( Convert.ToString(vidas),
  70,0, // Coordenadas
  200, 200, 200, // Colores
  tipoDeLetra);
```

Estamos hablando de "DibujarOculta" y de "EscribirTextoOculta", porque, para evitar parpadeos, no dibujaremos directamente en la pantalla visible, sino en otra zona de memoria (lo que se conoce como un "doble buffer"), y cuando hayamos preparado todo lo que debe verse en pantalla, entonces volcaremos esa "pantalla oculta" sobre la "pantalla visible". Así, la apariencia de la función "Dibujar" será:

```
public static void Dibujar()
{
```

```
    // -- Dibujar --
    Hardware.BorrarPantallaOculta(0,0,0);

    // Marcador
    Hardware.EscribirTextoOculta("Vidas",
              0,0, // Coordenadas
              255, 255, 255, // Colores
             tipoDeLetra);

    ...

    personaje.imagen.DibujarOculta(
         personaje.x, personaje.y);

    // Finalmente, muestro en pantalla
    Hardware.VisualizarOculta();
}
```

(Empezamos borrando la pantalla oculta en color negro, escribimos y dibujamos en ella todo lo que nos interese y finalmente visualizamos esa pantalla oculta)

La forma de comprobar teclas es parecida a la de antes, salvo por que la sintaxis es distinta, y que ahora el personaje no se moverá de uno en uno (que sería de píxel en píxel, demasiado lento), sino con un cierto incremento:

```
public static void ComprobarTeclas()
{
    // -- Leer teclas y calcular nueva posición --
    if (Hardware.TeclaPulsada(Hardware.TECLA_ESC) )
         partidaTerminada = true;

    if (Hardware.TeclaPulsada(Hardware.TECLA_DER) )
         personaje.x += personaje.incrX;
    if (Hardware.TeclaPulsada(Hardware.TECLA_IZQ) )
         personaje.x -= personaje.incrX;
    if (Hardware.TeclaPulsada(Hardware.TECLA_ARR) )
         personaje.y -= personaje.incrY;
    if (Hardware.TeclaPulsada(Hardware.TECLA_ABA) )
         personaje.y += personaje.incrY;
}
```

Las funciones de MoverElementos y ComprobarColisiones no cambian, aunque la segunda sí debería hacerlo, porque es difícil que nuestro personaje y un enemigo coincidan exactamente en la misma coordenada X de pantalla y en la misma Y, debería bastar con que se solaparan, pero eso lo mejoraremos en la siguiente entrega.

En "InicializarPartida" no habrá grandes diferencias: apenas que las coordenadas ahora de van de 0 a 79 en horizontal y de 0 a 24 en vertical, sino de 0 a 799 y de 0 a 599 (si usamos el modo de pantalla de 800x600 puntos), y que deberemos cargar las imágenes en vez de asignar letras (realmente no haría falta cargar las imágenes para cada nueva partida; lo mejoraremos más adelante):

```
public static void InicializarPartida()
{
    // En cada partida, hay que reiniciar ciertas variables
    vidas = 3;
    puntos = 0;
    partidaTerminada = false;

    personaje.xInicial = 400;
    personaje.yInicial = 300;
    personaje.incrX = 10;
    personaje.incrY = 10;
```

```
personaje.imagen = new Imagen("personaje.png");
...
```

Y en "InicializarJuego" el único cambio es que deberemos "entrar a modo gráfico", por ejemplo a 800x600, con 24 bits de color (16 millones de colores), y detallar también si queremos que el juego esté en ventana o en pantalla completa:

```
public static void InicializarJuego()
{
    // Entrar a modo grafico 800x600
    bool pantallaCompleta = false;
    Hardware.Inicializar(800, 600, 24, pantallaCompleta);

    // Resto de inicializacion
    tipoDeLetra = new Fuente("FreeSansBold.ttf", 18);
    juegoTerminado = false;
    numPremios = 10;
    ...
```

El fuente que une todas estas ideas podría ser así:

```
// Primer mini-esqueleto de juego en modo gráfico
// Versión "a"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego05a
{
    struct ElemGrafico {
        public int x;
        public int y;
        public int xInicial;
        public int yInicial;
        public int incrX;
        public int incrY;
        public Imagen imagen;
        public bool visible;
    }

    static ElemGrafico personaje;
    static Fuente tipoDeLetra;

    static int numPremios, numEnemigos, numObstaculos;

    static ElemGrafico[] obstaculos;
    static ElemGrafico[] enemigos;
    static ElemGrafico[] premios;

    static bool juegoTerminado;
    static int vidas;
    static int puntos;
    static bool partidaTerminada;
    static Random generador;


    public static void InicializarJuego()
    {
        // Entrar a modo grafico 800x600
        bool pantallaCompleta = false;
```

```
        Hardware.Inicializar(800, 600, 24, pantallaCompleta);

        // Resto de inicializacion
        tipoDeLetra = new Fuente("FreeSansBold.ttf", 18);
        juegoTerminado = false;
        numPremios = 10;
        numEnemigos = 10;
        numObstaculos = 20;
        obstaculos = new ElemGrafico[numObstaculos];
        enemigos = new ElemGrafico[numEnemigos];
        premios = new ElemGrafico[numPremios];
        generador = new Random();
    }


    public static void InicializarPartida()
    {
        // En cada partida, hay que reiniciar ciertas variables
        vidas = 3;
        puntos = 0;
        partidaTerminada = false;

        personaje.xInicial = 400;
        personaje.yInicial = 300;
        personaje.x = personaje.xInicial;
        personaje.y = personaje.yInicial;
        personaje.incrX = 10;
        personaje.incrY = 10;
        personaje.imagen = new Imagen("personaje.png");

        // Genero las posiciones de los elementos al azar
        for (int i=0; i<numObstaculos; i++)  // Obstaculos
        {
            obstaculos[i].x = generador.Next(50,700);
            obstaculos[i].y = generador.Next(30,550);
            obstaculos[i].imagen = new Imagen("obstaculo.png");
        }

        for (int i=0; i<numEnemigos; i++)  // Enemigos
        {
            enemigos[i].x = generador.Next(50,700);
            enemigos[i].y = generador.Next(30,550);
            enemigos[i].incrX = 5;
            enemigos[i].imagen = new Imagen("enemigo.png");
        }

        for (int i=0; i<numPremios; i++)  // Premios
        {
            premios[i].x = generador.Next(50,700);
            premios[i].y = generador.Next(30,550);
            premios[i].imagen = new Imagen("premio.png");
            premios[i].visible = true;
        }
    }


    public static void MostrarPresentacion()
    {
        // ---- Pantalla de presentación --
```

```
     Hardware.BorrarPantallaOculta(0,0,0);

     // Marcador
     Hardware.EscribirTextoOculta("Jueguecillo",
               340,200, // Coordenadas
               255, 255, 255, // Colores
              tipoDeLetra);

     Hardware.EscribirTextoOculta("Escoja una opción:",
               310,300, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);

     Hardware.EscribirTextoOculta("1.- Jugar una partida",
               150,390, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);

     Hardware.EscribirTextoOculta("0.- Salir",
               150,430, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);
     Hardware.VisualizarOculta();

     bool finPresentacion = false;
     do
     {
          Hardware.Pausa( 20 );
          if (Hardware.TeclaPulsada(Hardware.TECLA_1) )
               finPresentacion = true;

          if (Hardware.TeclaPulsada(Hardware.TECLA_0) )
          {
               finPresentacion = true;
               partidaTerminada = true;
               juegoTerminado = true;
          }
     } while (! finPresentacion );
}

public static void Dibujar()
{
     // -- Dibujar --
     Hardware.BorrarPantallaOculta(0,0,0);

     // Marcador
     Hardware.EscribirTextoOculta("Vidas        Puntos",
               0,0, // Coordenadas
               255, 255, 255, // Colores
              tipoDeLetra);

     Hardware.EscribirTextoOculta( Convert.ToString(vidas),
               70,0, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);

     Hardware.EscribirTextoOculta( Convert.ToString(puntos),
               190,0, // Coordenadas
               200, 200, 200, // Colores
```

```
                tipoDeLetra);

        for (int i=0; i<numObstaculos; i++)  // Obstáculos
        {
            obstaculos[i].imagen.DibujarOculta(
                (int) obstaculos[i].x, (int) obstaculos[i].y);
        }

        for (int i=0; i<numEnemigos; i++)  // Enemigos
        {
            enemigos[i].imagen.DibujarOculta(
                (int) enemigos[i].x, (int) enemigos[i].y);
        }

        for (int i=0; i<numPremios; i++)  // Premios
        {
            if (premios[i].visible)
            {
                premios[i].imagen.DibujarOculta(
                    premios[i].x, premios[i].y);
            }
        }

        personaje.imagen.DibujarOculta(
            personaje.x, personaje.y);

        // Finalmente, muestro en pantalla
        Hardware.VisualizarOculta();
    }


    public static void ComprobarTeclas()
    {
        // -- Leer teclas y calcular nueva posición --
        if (Hardware.TeclaPulsada(Hardware.TECLA_ESC) )
            partidaTerminada = true;

        if (Hardware.TeclaPulsada(Hardware.TECLA_DER) )
            personaje.x += personaje.incrX;
        if (Hardware.TeclaPulsada(Hardware.TECLA_IZQ) )
            personaje.x -= personaje.incrX;
        if (Hardware.TeclaPulsada(Hardware.TECLA_ARR) )
            personaje.y -= personaje.incrY;
        if (Hardware.TeclaPulsada(Hardware.TECLA_ABA) )
            personaje.y += personaje.incrY;
    }


    public static void MoverElementos()
    {
        // -- Mover enemigos, entorno --
        for (int i=0; i<numEnemigos; i++)  // Enemigos
        {
            enemigos[i].x = enemigos[i].x + enemigos[i].incrX;
            if (( (int) enemigos[i].x <= 50)
                    || ( (int) enemigos[i].x >= 700))
                enemigos[i].incrX = - enemigos[i].incrX;
        }
    }
```

```
public static void ComprobarColisiones()
{
     // -- Colisiones, perder vidas, etc --
     for (int i=0; i<numObstaculos; i++)  // Obstáculos
     {
          if ((obstaculos[i].x == personaje.x)
               && (obstaculos[i].y == personaje.y))
          {
               vidas --;
               if (vidas == 0)
                    partidaTerminada=true;
               personaje.x = personaje.xInicial;
               personaje.y = personaje.yInicial;
          }
     }

     for (int i=0; i<numPremios; i++)  // Obstáculos
     {
          if ((premios[i].x == personaje.x)
               && (premios[i].y == personaje.y)
               && premios[i].visible )
          {
               puntos += 10;
               premios[i].visible = false;
          }
     }

     for (int i=0; i<numEnemigos; i++)  // Enemigos
     {
          if (( (int) enemigos[i].x == personaje.x)
                    && ( (int) enemigos[i].y == personaje.y))
          {
               vidas --;
               if (vidas == 0)
                    partidaTerminada=true;
               personaje.x = personaje.xInicial;
               personaje.y = personaje.yInicial;
          }
     }

}


public static void PausaFotograma()
{
     // -- Pausa hasta el siguiente "fotograma" del juego --
     Hardware.Pausa( 20 );
}


public static void Main()
{

     InicializarJuego();

     while (! juegoTerminado )
     {
```

```
                InicializarPartida();
                MostrarPresentacion();

                // ------ Bucle de juego ------
                while( ! partidaTerminada  )
                {
                     Dibujar();
                     ComprobarTeclas();
                     MoverElementos();
                     ComprobarColisiones();
                     PausaFotograma();
                } // Fin del bucle de juego
           }     // Fin de partida
      }          // Fin de Main
}
```

Este programa no está formado sólo por ese fichero, sino que también existen otros ficheros que detallan como cargar una Imagen, o un tipo de letra (Fuente), o cómo acceder al Hardware (a la pantalla gráfica, el teclado, un joystick o gamepad). Además, son necesarios varios ficheros DLL. Por eso, si quieres probar a modificar este juego para tu propia versión, tendrás que descargar uno de los siguientes ficheros:

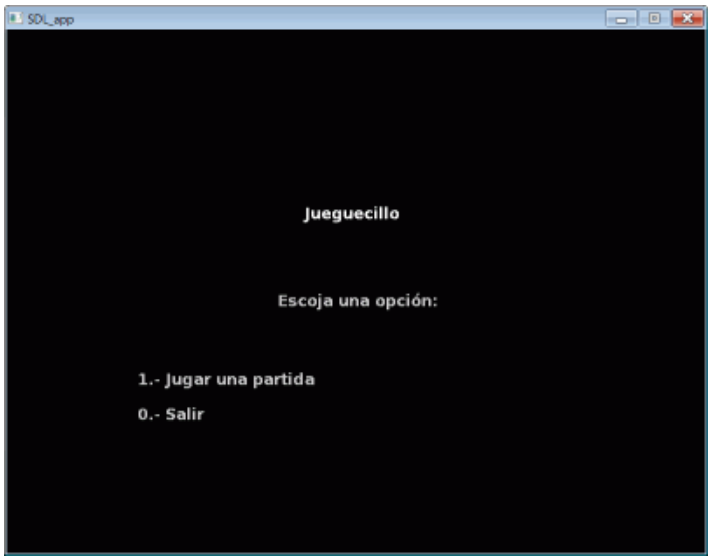
- Un proyecto para Visual Studio 2010, que incluye todo lo necesario para compilar y probar el juego. Basta con la versión Express, que se puede descargar de forma gratuita desde la página de Microsoft.
- Una versión para compilar desde línea de comandos, por si no tienes Visual Studio 2010 (pero necesitas tener al menos la plataforma "punto net" en su versión 2, y hacer doble clic en "CompilaDotNet" para crear el ejecutable.
- Una versión alternativa para MonoDevelop 2.4, que quizá te sirva si usas Linux o MacOS X (sí funciona en un Linux Mint 11 32 bits con SDL 1.2.x, quizá no funcione con versiones más modernas de SDL o con sistemas de 64 bits). Es posible que necesites instalar la versión "de desarrollo" de SDL: libsdl1.2-dev (o similar) y sus auxiliares correspondientes: libsdl-image1.2-dev para visualizar imágenes, libsdl-ttf2.0-dev para tipos de letra TrueType y (dentro de no mucho tiempo) libsdl-mixer1.2-dev para reproducir sonidos MP3 y MID.

Ejercicio propuesto: Incluye una imagen de fondo en la pantalla de presentación. Piensa cómo hacer que las colisiones sean correctas (si no lo descubres, no te preocupes: lo veremos en un apartado cercano).

22. Imagen de fondo para la presentación

Antes de hacer cosas más avanzadas, vamos a incluir una imagen como fondo de la presentación, que nos permita repasar la forma de cargar imágenes y de mostrarlas en pantalla.

Hasta ahora nuestra presentación contenía apenas 3 textos:



Ahora vamos a hacer que aparezca una imagen de fondo, para que sea un poco menos "sosa".

Para cargar la imagen podemos crear un nuevo elemento de tipo "Imagen", indicando el nombre de fichero que contiene esa imagen:

```
Imagen fondoPresentacion = new Imagen("present.jpg");
```

Y si la imagen es de tamaño 800x600, que es el tamaño de nuestra pantalla, la dibujaríamos en las coordenadas (0,0), antes de escribir los textos (para que no "los tape"):

```
fondoPresentacion.DibujarOculta(0,0);
```

Esto se haría antes de la parte repetitiva que comprueba pulsaciones de teclas, porque no tiene sentido cargar la imagen múltiples veces:

```
public static void MostrarPresentacion()
{
     Imagen fondoPresentacion = new Imagen("present.jpg");
     // ---- Pantalla de presentación --
     Hardware.BorrarPantallaOculta(0,0,0);

     // Pantalla de fondo
     fondoPresentacion.DibujarOculta(0,0);

     // Marcador
     Hardware.EscribirTextoOculta("Jueguecillo",
               340,200, // Coordenadas
               255, 255, 255, // Colores
              tipoDeLetra);

     ...
```

El programa completo, que no tiene más cambios que esos, quedaría así:

```
// Primer mini-esqueleto de juego en modo gráfico
// Versión "b"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego05b
{
     struct ElemGrafico {
          public int x;
          public int y;
          public int xInicial;
          public int yInicial;
          public int incrX;
          public int incrY;
          public Imagen imagen;
          public bool visible;
     }

     static ElemGrafico personaje;
     static Fuente tipoDeLetra;

     static int numPremios, numEnemigos, numObstaculos;

     static ElemGrafico[] obstaculos;
     static ElemGrafico[] enemigos;
     static ElemGrafico[] premios;

     static bool juegoTerminado;
     static int vidas;
```

```
static int puntos;
static bool partidaTerminada;
static Random generador;


public static void InicializarJuego()
{
     // Entrar a modo grafico 800x600
     bool pantallaCompleta = false;
     Hardware.Inicializar(800, 600, 24, pantallaCompleta);

     // Resto de inicializacion
     tipoDeLetra = new Fuente("FreeSansBold.ttf", 18);
     juegoTerminado = false;
     numPremios = 10;
     numEnemigos = 10;
     numObstaculos = 20;
     obstaculos = new ElemGrafico[numObstaculos];
     enemigos = new ElemGrafico[numEnemigos];
     premios = new ElemGrafico[numPremios];
     generador = new Random();
}


public static void InicializarPartida()
{
     // En cada partida, hay que reiniciar ciertas variables
     vidas = 3;
     puntos = 0;
     partidaTerminada = false;

     personaje.xInicial = 400;
     personaje.yInicial = 300;
     personaje.x = personaje.xInicial;
     personaje.y = personaje.yInicial;
     personaje.incrX = 10;
     personaje.incrY = 10;
     personaje.imagen = new Imagen("personaje.png");

     // Genero las posiciones de los elementos al azar
     for (int i=0; i<numObstaculos; i++)  // Obstaculos
     {
          obstaculos[i].x = generador.Next(50,700);
          obstaculos[i].y = generador.Next(30,550);
          obstaculos[i].imagen = new Imagen("obstaculo.png");
     }

     for (int i=0; i<numEnemigos; i++)  // Enemigos
     {
          enemigos[i].x = generador.Next(50,700);
          enemigos[i].y = generador.Next(30,550);
          enemigos[i].incrX = 5;
          enemigos[i].imagen = new Imagen("enemigo.png");
     }

     for (int i=0; i<numPremios; i++)  // Premios
     {
          premios[i].x = generador.Next(50,700);
          premios[i].y = generador.Next(30,550);
```

```
            premios[i].imagen = new Imagen("premio.png");
            premios[i].visible = true;
        }
    }


    public static void MostrarPresentacion()
    {
        // Cargo la imagen de la presentación
        Imagen fondo = new Imagen("present.jpg");

        // ---- Pantalla de presentación --
        Hardware.BorrarPantallaOculta(0,0,0);

        // Fondo de la presentación
        fondo.DibujarOculta(0,0);

        // Marcador
        Hardware.EscribirTextoOculta("Jueguecillo",
                  340,200, // Coordenadas
                  255, 255, 255, // Colores
                 tipoDeLetra);

        Hardware.EscribirTextoOculta("Escoja una opción:",
                  310,300, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);

        Hardware.EscribirTextoOculta("1.- Jugar una partida",
                  150,390, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);

        Hardware.EscribirTextoOculta("0.- Salir",
                  150,430, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);
        Hardware.VisualizarOculta();

        bool finPresentacion = false;
        do
        {
            Hardware.Pausa( 20 );
            if (Hardware.TeclaPulsada(Hardware.TECLA_1) )
                finPresentacion = true;

            if (Hardware.TeclaPulsada(Hardware.TECLA_0) )
            {
                finPresentacion = true;
                partidaTerminada = true;
                juegoTerminado = true;
            }
        } while (! finPresentacion );
    }

    public static void Dibujar()
    {
        // -- Dibujar --
        Hardware.BorrarPantallaOculta(0,0,0);
```

```
    // Marcador
    Hardware.EscribirTextoOculta("Vidas        Puntos",
              0,0, // Coordenadas
              255, 255, 255, // Colores
             tipoDeLetra);

    Hardware.EscribirTextoOculta( Convert.ToString(vidas),
              70,0, // Coordenadas
              200, 200, 200, // Colores
             tipoDeLetra);

    Hardware.EscribirTextoOculta( Convert.ToString(puntos),
              190,0, // Coordenadas
              200, 200, 200, // Colores
             tipoDeLetra);

    for (int i=0; i<numObstaculos; i++)  // Obstáculos
    {
         obstaculos[i].imagen.DibujarOculta(
              (int) obstaculos[i].x, (int) obstaculos[i].y);
    }

    for (int i=0; i<numEnemigos; i++)  // Enemigos
    {
         enemigos[i].imagen.DibujarOculta(
              (int) enemigos[i].x, (int) enemigos[i].y);
    }

    for (int i=0; i<numPremios; i++)  // Premios
    {
         if (premios[i].visible)
         {
              premios[i].imagen.DibujarOculta(
                   premios[i].x, premios[i].y);
         }
    }

    personaje.imagen.DibujarOculta(
         personaje.x, personaje.y);

    // Finalmente, muestro en pantalla
    Hardware.VisualizarOculta();
}


public static void ComprobarTeclas()
{
    // -- Leer teclas y calcular nueva posición --
    if (Hardware.TeclaPulsada(Hardware.TECLA_ESC) )
         partidaTerminada = true;

    if (Hardware.TeclaPulsada(Hardware.TECLA_DER) )
         personaje.x += personaje.incrX;
    if (Hardware.TeclaPulsada(Hardware.TECLA_IZQ) )
         personaje.x -= personaje.incrX;
    if (Hardware.TeclaPulsada(Hardware.TECLA_ARR) )
         personaje.y -= personaje.incrY;
    if (Hardware.TeclaPulsada(Hardware.TECLA_ABA) )
```

```
          personaje.y += personaje.incrY;
}


public static void MoverElementos()
{
     // -- Mover enemigos, entorno --
     for (int i=0; i<numEnemigos; i++)  // Enemigos
     {
          enemigos[i].x = enemigos[i].x + enemigos[i].incrX;
          if (( (int) enemigos[i].x <= 50)
                    || ( (int) enemigos[i].x >= 700))
               enemigos[i].incrX = - enemigos[i].incrX;
     }
}


public static void ComprobarColisiones()
{
     // -- Colisiones, perder vidas, etc --
     for (int i=0; i<numObstaculos; i++)  // Obstáculos
     {
          if ((obstaculos[i].x == personaje.x)
               && (obstaculos[i].y == personaje.y))
          {
               vidas --;
               if (vidas == 0)
                    partidaTerminada=true;
               personaje.x = personaje.xInicial;
               personaje.y = personaje.yInicial;
          }
     }

     for (int i=0; i<numPremios; i++)  // Obstáculos
     {
          if ((premios[i].x == personaje.x)
               && (premios[i].y == personaje.y)
               && premios[i].visible )
          {
               puntos += 10;
               premios[i].visible = false;
          }
     }

     for (int i=0; i<numEnemigos; i++)  // Enemigos
     {
          if (( (int) enemigos[i].x == personaje.x)
                    && ( (int) enemigos[i].y == personaje.y))
          {
               vidas --;
               if (vidas == 0)
                    partidaTerminada=true;
               personaje.x = personaje.xInicial;
               personaje.y = personaje.yInicial;
          }
     }

}
```

```
    public static void PausaFotograma()
    {
        // -- Pausa hasta el siguiente "fotograma" del juego --
        Hardware.Pausa( 20 );
    }


    public static void Main()
    {

        InicializarJuego();

        while (! juegoTerminado )
        {
            InicializarPartida();
            MostrarPresentacion();

            // ------ Bucle de juego ------
            while( ! partidaTerminada  )
            {
                Dibujar();
                ComprobarTeclas();
                MoverElementos();
                ComprobarColisiones();
                PausaFotograma();
            } // Fin del bucle de juego
        }     // Fin de partida
    }         // Fin de Main
}
```

Ejercicio propuesto: Esto se puede afinar un poco más: no es necesario cargar la imagen cada vez que se entra a la presentación, basta con cargarla una única vez cuando se inicializa el juego. Haz una versión mejorada del programa que incluya este cambio.

Ejercicio propuesto: Haz que las imágenes de personaje, enemigos, premios, etc. se carguen también en el momento de inicializar el juego, no cada vez que va a comenzar una nueva partida.

(Nota: si quieres ver cómo quedaría el fuente con estos cambios propuestos... tendrás que mirar en el siguiente apartado, que partirá de ese punto).

23. Pantalla de ayuda y de créditos

Vamos a ver también cómo añadir más opciones al menú principal. Una de ellas mostrará una pequeña ayuda, que recuerde (por ejemplo) las teclas que se pueden usar durante el juego. Otra será una pantalla de "créditos", que muestre datos sobre el autor de nuestro juego.

Crearemos un par de funciones nuevas. Su estructura será muy similar a la que tenía la presentación, salvo que ahora sólo es necesario esperar hasta que se pulse ESC:

```
public static void MostrarCreditos()
{
    // ---- Pantalla de presentación --
    Hardware.BorrarPantallaOculta(0, 0, 0);

    // Fondo de la presentación
    fondoCreditos.DibujarOculta(0, 0);

    // Marcador
    Hardware.EscribirTextoOculta("Creditos",
              250, 200, // Coordenadas
              255, 255, 255, // Colores
```

```
            tipoDeLetra);

    Hardware.EscribirTextoOculta("Por Nacho Cabanes, 2011",
                250, 300, // Coordenadas
                200, 200, 200, // Colores
               tipoDeLetra);

    Hardware.EscribirTextoOculta("ESC- volver",
               650, 530, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);
    Hardware.VisualizarOculta();

    do
    {
        Hardware.Pausa(20);
    } while (!Hardware.TeclaPulsada(Hardware.TECLA_ESC));
}
```

Dejamos una "Pausa" dentro del "do..while" para que no use el 100% del procesador para algo tan sencillo como esperar una tecla. Con esa "pausa" ayudamos a mantener la multitarea del sistema operativo.

Ahora tendremos que ampliar la presentación, para que avise de las nuevas opciones, compruebe las teclas necesarias y llame a las nuevas funciones cuando sea el caso:

```
public static void MostrarPresentacion()
{
    bool finPresentacion = false;

    do
    {
        // ---- Pantalla de presentación --
        Hardware.BorrarPantallaOculta(0, 0, 0);

        // Fondo de la presentación
        fondoPresentacion.DibujarOculta(0, 0);

        // Marcador
        Hardware.EscribirTextoOculta("Jueguecillo",
                  340, 200, // Coordenadas
                  255, 255, 255, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("Escoja una opción:",
                  310, 300, // Coordenadas
                  200, 200, 200, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("J.- Jugar una partida",
                  150, 390, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);

        Hardware.EscribirTextoOculta("A.- Ayuda",
                  150, 430, // Coordenadas
                  200, 200, 200, // Colores
                  tipoDeLetra);
```

```
            Hardware.EscribirTextoOculta("C.- Créditos",
                      150, 470, // Coordenadas
                      200, 200, 200, // Colores
                      tipoDeLetra);

            Hardware.EscribirTextoOculta("S.- Salir",
                      150, 510, // Coordenadas
                      200, 200, 200, // Colores
                      tipoDeLetra);

            Hardware.VisualizarOculta();

            Hardware.Pausa(20);

            if (Hardware.TeclaPulsada(Hardware.TECLA_A))
                 MostrarAyuda();

            if (Hardware.TeclaPulsada(Hardware.TECLA_C))
                 MostrarCreditos();

            if (Hardware.TeclaPulsada(Hardware.TECLA_J))
                 finPresentacion = true;

            if (Hardware.TeclaPulsada(Hardware.TECLA_S))
            {
                 finPresentacion = true;
                 partidaTerminada = true;
                 juegoTerminado = true;
            }
       } while (!finPresentacion);
}
```

Esto se haría antes de la parte repetitiva que comprueba pulsaciones de teclas, porque no tiene sentido cargar la imagen múltiples veces:

```
public static void MostrarPresentacion()
{
     Imagen fondoPresentacion = new Imagen("present.jpg");
     // ---- Pantalla de presentación --
     Hardware.BorrarPantallaOculta(0,0,0);

     // Pantalla de fondo
     fondoPresentacion.DibujarOculta(0,0);

     // Marcador
     Hardware.EscribirTextoOculta("Jueguecillo",
                340,200, // Coordenadas
                255, 255, 255, // Colores
               tipoDeLetra);

     ...
```

El programa completo sería algo como:

```
// Primer mini-esqueleto de juego en modo gráfico
// Versión "d"

using System;
using System.Threading; // Para Thread.Sleep
```

```
public class Juego05d
{
    struct ElemGrafico
    {
        public int x;
        public int y;
        public int xInicial;
        public int yInicial;
        public int incrX;
        public int incrY;
        public Imagen imagen;
        public bool visible;
    }

    static ElemGrafico personaje;
    static Fuente tipoDeLetra;

    static int numPremios, numEnemigos, numObstaculos;

    static ElemGrafico[] obstaculos;
    static ElemGrafico[] enemigos;
    static ElemGrafico[] premios;

    static bool juegoTerminado;
    static int vidas;
    static int puntos;
    static bool partidaTerminada;
    static Random generador;

    static Imagen fondoPresentacion;
    static Imagen fondoAyuda;
    static Imagen fondoCreditos;


    public static void InicializarJuego()
    {
        // Entrar a modo grafico 800x600
        bool pantallaCompleta = false;
        Hardware.Inicializar(800, 600, 24, pantallaCompleta);

        // Resto de inicializacion
        tipoDeLetra = new Fuente("FreeSansBold.ttf", 18);
        juegoTerminado = false;
        numPremios = 10;
        numEnemigos = 10;
        numObstaculos = 20;
        obstaculos = new ElemGrafico[numObstaculos];
        enemigos = new ElemGrafico[numEnemigos];
        premios = new ElemGrafico[numPremios];
        generador = new Random();

        // Cargo imágenes de elementos
        personaje.imagen = new Imagen("personaje.png");

        for (int i = 0; i < numObstaculos; i++)  // Obstaculos
            obstaculos[i].imagen = new Imagen("obstaculo.png");

        for (int i = 0; i < numEnemigos; i++)  // Enemigos
            enemigos[i].imagen = new Imagen("enemigo.png");
```

```
    for (int i = 0; i < numPremios; i++)  // Premios
        premios[i].imagen = new Imagen("premio.png");

    // Y cargo las imagenes de la presentación, ayuda y créditos
    fondoPresentacion = new Imagen("present.jpg");
    fondoAyuda = new Imagen("present.jpg");
    fondoCreditos = new Imagen("present.jpg");
}


public static void InicializarPartida()
{
    // En cada partida, hay que reiniciar ciertas variables
    vidas = 3;
    puntos = 0;
    partidaTerminada = false;

    personaje.xInicial = 400;
    personaje.yInicial = 300;
    personaje.x = personaje.xInicial;
    personaje.y = personaje.yInicial;
    personaje.incrX = 10;
    personaje.incrY = 10;

    // Genero las posiciones de los elementos al azar
    for (int i = 0; i < numObstaculos; i++)  // Obstaculos
    {
        obstaculos[i].x = generador.Next(50, 700);
        obstaculos[i].y = generador.Next(30, 550);
    }

    for (int i = 0; i < numEnemigos; i++)  // Enemigos
    {
        enemigos[i].x = generador.Next(50, 700);
        enemigos[i].y = generador.Next(30, 550);
        enemigos[i].incrX = 5;
    }

    for (int i = 0; i < numPremios; i++)  // Premios
    {
        premios[i].x = generador.Next(50, 700);
        premios[i].y = generador.Next(30, 550);
        premios[i].visible = true;
    }
}


public static void MostrarPresentacion()
{
    bool finPresentacion = false;

    do
    {
        // ---- Pantalla de presentación --
        Hardware.BorrarPantallaOculta(0, 0, 0);

        // Fondo de la presentación
        fondoPresentacion.DibujarOculta(0, 0);
```

```
        // Marcador
        Hardware.EscribirTextoOculta("Jueguecillo",
                  340, 200, // Coordenadas
                  255, 255, 255, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("Escoja una opción:",
                  310, 300, // Coordenadas
                  200, 200, 200, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("J.- Jugar una partida",
                  150, 390, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);

        Hardware.EscribirTextoOculta("A.- Ayuda",
                  150, 430, // Coordenadas
                  200, 200, 200, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("C.- Créditos",
                  150, 470, // Coordenadas
                  200, 200, 200, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("S.- Salir",
                  150, 510, // Coordenadas
                  200, 200, 200, // Colores
                  tipoDeLetra);

        Hardware.VisualizarOculta();

        Hardware.Pausa(20);

        if (Hardware.TeclaPulsada(Hardware.TECLA_A))
             MostrarAyuda();

        if (Hardware.TeclaPulsada(Hardware.TECLA_C))
             MostrarCreditos();

        if (Hardware.TeclaPulsada(Hardware.TECLA_J))
             finPresentacion = true;

        if (Hardware.TeclaPulsada(Hardware.TECLA_S))
        {
             finPresentacion = true;
             partidaTerminada = true;
             juegoTerminado = true;
        }
    } while (!finPresentacion);
}


public static void MostrarAyuda()
{
    // ---- Pantalla de presentación --
    Hardware.BorrarPantallaOculta(0, 0, 0);
```

```
    // Fondo de la presentación
    fondoAyuda.DibujarOculta(0, 0);

    // Marcador
    Hardware.EscribirTextoOculta("Ayuda",
               340, 200, // Coordenadas
               255, 255, 255, // Colores
              tipoDeLetra);

    Hardware.EscribirTextoOculta("(Texto de ejemplo)",
               310, 300, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);

    Hardware.EscribirTextoOculta("ESC- volver",
               650, 530, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);
    Hardware.VisualizarOculta();

    do
    {
        Hardware.Pausa(20);
    } while (!Hardware.TeclaPulsada(Hardware.TECLA_ESC));
}


public static void MostrarCreditos()
{
    // ---- Pantalla de presentación --
    Hardware.BorrarPantallaOculta(0, 0, 0);

    // Fondo de la presentación
    fondoCreditos.DibujarOculta(0, 0);

    // Marcador
    Hardware.EscribirTextoOculta("Creditos",
               250, 200, // Coordenadas
               255, 255, 255, // Colores
              tipoDeLetra);

    Hardware.EscribirTextoOculta("Por Nacho Cabanes, 2011",
                250, 300, // Coordenadas
                200, 200, 200, // Colores
               tipoDeLetra);

    Hardware.EscribirTextoOculta("ESC- volver",
               650, 530, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);
    Hardware.VisualizarOculta();

    do
    {
        Hardware.Pausa(20);
    } while (!Hardware.TeclaPulsada(Hardware.TECLA_ESC));
}
```

```
public static void Dibujar()
{
     // -- Dibujar --
     Hardware.BorrarPantallaOculta(0, 0, 0);

     // Marcador
     Hardware.EscribirTextoOculta("Vidas        Puntos",
               0, 0, // Coordenadas
               255, 255, 255, // Colores
              tipoDeLetra);

     Hardware.EscribirTextoOculta(Convert.ToString(vidas),
               70, 0, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);

     Hardware.EscribirTextoOculta(Convert.ToString(puntos),
               190, 0, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);

     for (int i = 0; i < numObstaculos; i++)  // Obstáculos
     {
          obstaculos[i].imagen.DibujarOculta(
               (int)obstaculos[i].x, (int)obstaculos[i].y);
     }

     for (int i = 0; i < numEnemigos; i++)  // Enemigos
     {
          enemigos[i].imagen.DibujarOculta(
               (int)enemigos[i].x, (int)enemigos[i].y);
     }

     for (int i = 0; i < numPremios; i++)  // Premios
     {
          if (premios[i].visible)
          {
               premios[i].imagen.DibujarOculta(
                    premios[i].x, premios[i].y);
          }
     }

     personaje.imagen.DibujarOculta(
          personaje.x, personaje.y);

     // Finalmente, muestro en pantalla
     Hardware.VisualizarOculta();
}


public static void ComprobarTeclas()
{
     // -- Leer teclas y calcular nueva posición --
     if (Hardware.TeclaPulsada(Hardware.TECLA_ESC))
          partidaTerminada = true;

     if (Hardware.TeclaPulsada(Hardware.TECLA_DER))
          personaje.x += personaje.incrX;
```

```
        if (Hardware.TeclaPulsada(Hardware.TECLA_IZQ))
            personaje.x -= personaje.incrX;
        if (Hardware.TeclaPulsada(Hardware.TECLA_ARR))
            personaje.y -= personaje.incrY;
        if (Hardware.TeclaPulsada(Hardware.TECLA_ABA))
            personaje.y += personaje.incrY;
    }


    public static void MoverElementos()
    {
        // -- Mover enemigos, entorno --
        for (int i = 0; i < numEnemigos; i++)  // Enemigos
        {
            enemigos[i].x = enemigos[i].x + enemigos[i].incrX;
            if (((int)enemigos[i].x <= 50)
                    || ((int)enemigos[i].x >= 700))
                enemigos[i].incrX = -enemigos[i].incrX;
        }
    }


    public static void ComprobarColisiones()
    {
        // -- Colisiones, perder vidas, etc --
        for (int i = 0; i < numObstaculos; i++)  // Obstáculos
        {
            if ((obstaculos[i].x == personaje.x)
                && (obstaculos[i].y == personaje.y))
            {
                vidas--;
                if (vidas == 0)
                    partidaTerminada = true;
                personaje.x = personaje.xInicial;
                personaje.y = personaje.yInicial;
            }
        }

        for (int i = 0; i < numPremios; i++)  // Obstáculos
        {
            if ((premios[i].x == personaje.x)
                && (premios[i].y == personaje.y)
                && premios[i].visible)
            {
                puntos += 10;
                premios[i].visible = false;
            }
        }

        for (int i = 0; i < numEnemigos; i++)  // Enemigos
        {
            if (((int)enemigos[i].x == personaje.x)
                    && ((int)enemigos[i].y == personaje.y))
            {
                vidas--;
                if (vidas == 0)
                    partidaTerminada = true;
                personaje.x = personaje.xInicial;
                personaje.y = personaje.yInicial;
```

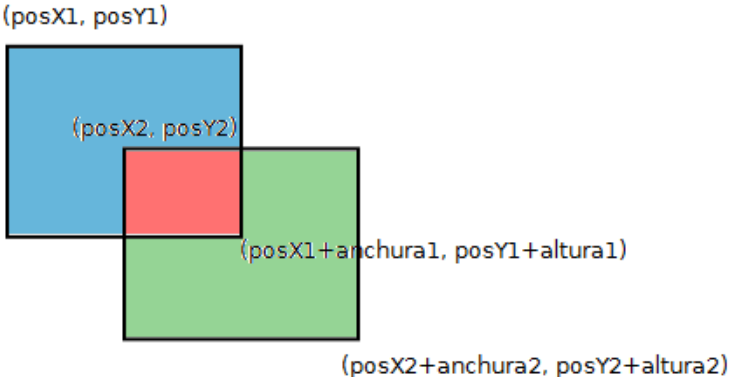
```
            }
        }

    }


    public static void PausaFotograma()
    {
        // -- Pausa hasta el siguiente "fotograma" del juego --
        Hardware.Pausa(20);
    }


    public static void Main()
    {

        InicializarJuego();

        while (!juegoTerminado)
        {
            InicializarPartida();
            MostrarPresentacion();

            // ------ Bucle de juego ------
            while (!partidaTerminada)
            {
                Dibujar();
                ComprobarTeclas();
                MoverElementos();
                ComprobarColisiones();
                PausaFotograma();
            } // Fin del bucle de juego
        }     // Fin de partida
    }         // Fin de Main
}
```

Ejercicio propuesto: Haz que la ayuda muestre textos más detallados: varias frases, que estarán almacenadas en un "array de strings". Deberás recorrer este array usando "for" o "foreach".

(Nota: nuevamente, si quieres ver cómo quedaría el fuente con estos cambios propuestos... tendrás que mirar en el siguiente apartado, que partirá de ese punto).

24. Colisiones simples en modo gráfico

Cuando estábamos en "modo texto", comprobábamos colisiones mirando si coincidían la X y la Y del personaje con las de algún enemigo (o premio, obstáculo...). Pero eso en modo gráfico no sirve, porque pueden "solaparse" aunque no lleguen a estar exactamente en la misma posición.



Si nos fijamos en la posición horizontal, para que se solapen, debe ocurrir que:

- El rectángulo azul no debe estar demasiado a la izquierda: su posición final (posX1+anchura1) debe ser mayor que la posición inicial del rectángulo verde (posX2)
- Además, el rectángulo azul no debe estar demasiado a la derecha: su posición inicial (posX1) debe ser menor que la posición final del rectángulo verde (posX2+anchura2)
- Las consideraciones en vertical son las mismas, con la única diferencia de que afectan a la posición Y y a la altura de cada rectángulo.

De esta forma, aproximamos los objetos por rectángulos, de modo que puede no ser útil si dejamos "hueco" alrededor de ellos cuando recortamos las imágenes, ni tampoco si los objetos son muy irregulares, pero es una forma sencilla y que funciona "razonablemente bien" en muchos casos.

Esto equivaldría a una función "Colision", que comprobara si dos elementos gráficos chocan entre sí, de la siguiente forma:

```
public static bool Colision(ElemGrafico e1, ElemGrafico e2)
{
     if ((e1.x + e1.ancho > e2.x)
            && (e1.x < e2.x + e2.ancho)
            && (e1.y + e1.alto > e2.y)
            && (e1.y < e2.y + e2.alto))
          return true;
     else
          return false;
}
```

Una primera mejora, ya que los premios "desaparecerán" (visible=false) cuando los recojamos, es que sólo se compruebe la colisión cuando los dos objetos sean visibles:

```
public static bool Colision(ElemGrafico e1, ElemGrafico e2)
{
     // No se debe chocar con un elemento oculto
     if ((e1.visible == false) || (e2.visible == false))
          return false;
     // Ahora ya compruebo coordenadas
     if ((e1.x + e1.ancho > e2.x)
            && (e1.x < e2.x + e2.ancho)
            && (e1.y + e1.alto > e2.y)
            && (e1.y < e2.y + e2.alto))
          return true;
     else
          return false;
}
```

Esto nos obliga a que, cada vez que definamos un elemento gráfico, indiquemos su ancho, su alto y si está visible o no (tanto para premios, como para obstáculos, enemigos o para nuestro propio personaje):

```
for (int i = 0; i < numPremios; i++)  // Premios
{
     premios[i].x = generador.Next(50, 700);
     premios[i].y = generador.Next(30, 550);
```

```
        premios[i].visible = true;
        premios[i].ancho = 34;
        premios[i].alto = 18;
}
```

Y ya que estamos, podemos hacer una pequeña mejora:

Con tantos obstáculos y enemigos en pantalla, es muy fácil que choquemos nada más comenzar la partida y que ésta termine exageradamente pronto. Podemos evitarlo en primer lugar haciendo que no pueda aparecer un obstáculo inicialmente en la misma posición que nuestro personaje:

```
for (int i = 0; i < numObstaculos; i++)  // Obstaculos
{
        obstaculos[i].visible = true;
        obstaculos[i].ancho = 38;
        obstaculos[i].alto = 22;
        // Al colocar un obstáculo, compruebo que no choque con el personaje, para que la partida
        // no acabe nada más empezar
        do
        {
                obstaculos[i].x = generador.Next(50, 700);
                obstaculos[i].y = generador.Next(30, 550);
        } while (Colision(obstaculos[i], personaje));
}
```

Y como los enemigos se mueven, no basta con que no se choquen inicialmente con nuestro personaje. Será más fiable si no permitimos que estén en la misma franja vertical que nosotros:

```
for (int i = 0; i < numEnemigos; i++)  // Enemigos
{
        enemigos[i].x = generador.Next(50, 700);
        // Para la Y, compruebo que no choque con el personaje, para que la partida
        // no acabe nada más empezar
        do
        {
                enemigos[i].y = generador.Next(30, 550);
        } while ((enemigos[i].y+enemigos[i].alto > personaje.y)
                && (enemigos[i].y < personaje.y+personaje.alto) );
        enemigos[i].incrX = 5;
        enemigos[i].visible = true;
        enemigos[i].ancho = 36;
        enemigos[i].alto = 42;
}
```

El programa completo quedaría así:

```
// Primer mini-esqueleto de juego en modo gráfico
// Versión "f"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego05f
{
        public struct ElemGrafico
        {
                public int x;
                public int y;
                public int xInicial;
                public int yInicial;
                public int ancho;
                public int alto;
                public int incrX;
                public int incrY;
```

```
    public Imagen imagen;
    public bool visible;
}

static ElemGrafico personaje;
static Fuente tipoDeLetra;

static int numPremios, numEnemigos, numObstaculos;

static ElemGrafico[] obstaculos;
static ElemGrafico[] enemigos;
static ElemGrafico[] premios;

static bool juegoTerminado;
static int vidas;
static int puntos;
static bool partidaTerminada;
static Random generador;

static Imagen fondoPresentacion;
static Imagen fondoAyuda;
static Imagen fondoCreditos;


public static void InicializarJuego()
{
    // Entrar a modo grafico 800x600
    bool pantallaCompleta = false;
    Hardware.Inicializar(800, 600, 24, pantallaCompleta);

    // Resto de inicializacion
    tipoDeLetra = new Fuente("FreeSansBold.ttf", 18);
    juegoTerminado = false;
    numPremios = 10;
    numEnemigos = 10;
    numObstaculos = 20;
    obstaculos = new ElemGrafico[numObstaculos];
    enemigos = new ElemGrafico[numEnemigos];
    premios = new ElemGrafico[numPremios];
    generador = new Random();

    // Cargo imágenes de elementos
    personaje.imagen = new Imagen("personaje.png");

    for (int i = 0; i < numObstaculos; i++)  // Obstaculos
        obstaculos[i].imagen = new Imagen("obstaculo.png");

    for (int i = 0; i < numEnemigos; i++)  // Enemigos
        enemigos[i].imagen = new Imagen("enemigo.png");

    for (int i = 0; i < numPremios; i++)  // Premios
        premios[i].imagen = new Imagen("premio.png");

    // Y cargo las imagenes de la presentación, ayuda y créditos
    fondoPresentacion = new Imagen("present.jpg");
    fondoAyuda = new Imagen("present.jpg");
    fondoCreditos = new Imagen("present.jpg");
}
```

```
public static void InicializarPartida()
{
     // En cada partida, hay que reiniciar ciertas variables
     vidas = 3;
     puntos = 0;
     partidaTerminada = false;

     personaje.xInicial = 400;
     personaje.yInicial = 300;
     personaje.x = personaje.xInicial;
     personaje.y = personaje.yInicial;
     personaje.visible = true;
     personaje.ancho = 32;
     personaje.alto = 30;
     personaje.incrX = 10;
     personaje.incrY = 10;

     // Genero las posiciones de los elementos al azar
     for (int i = 0; i < numObstaculos; i++)  // Obstaculos
     {
          obstaculos[i].visible = true;
          obstaculos[i].ancho = 38;
          obstaculos[i].alto = 22;
          // Al colocar un obstáculo, compruebo que no choque
          // con el personaje, para que la partida
          // no acabe nada más empezar
          do
          {
               obstaculos[i].x = generador.Next(50, 700);
               obstaculos[i].y = generador.Next(30, 550);
          } while (Colision(obstaculos[i], personaje));
     }

     for (int i = 0; i < numEnemigos; i++)  // Enemigos
     {
          enemigos[i].x = generador.Next(50, 700);
          // Para la Y, compruebo que no sea del rango de
          // la del personaje, para que la partida
          // no acabe nada más empezar
          do
          {
               enemigos[i].y = generador.Next(30, 550);
          } while ((enemigos[i].y+enemigos[i].alto > personaje.y)
               && (enemigos[i].y < personaje.y+personaje.alto) );
          enemigos[i].incrX = 5;
          enemigos[i].visible = true;
          enemigos[i].ancho = 36;
          enemigos[i].alto = 42;
     }

     for (int i = 0; i < numPremios; i++)  // Premios
     {
          premios[i].x = generador.Next(50, 700);
          premios[i].y = generador.Next(30, 550);
          premios[i].visible = true;
          premios[i].ancho = 34;
          premios[i].alto = 18;
     }
```

```
}


public static void MostrarPresentacion()
{
     bool finPresentacion = false;

     do
     {
          // ---- Pantalla de presentación --
          Hardware.BorrarPantallaOculta(0, 0, 0);

          // Fondo de la presentación
          fondoPresentacion.DibujarOculta(0, 0);

          // Marcador
          Hardware.EscribirTextoOculta("Jueguecillo",
                    340, 200, // Coordenadas
                    255, 255, 255, // Colores
                    tipoDeLetra);

          Hardware.EscribirTextoOculta("Escoja una opción:",
                    310, 300, // Coordenadas
                    200, 200, 200, // Colores
                    tipoDeLetra);

          Hardware.EscribirTextoOculta("J.- Jugar una partida",
                    150, 390, // Coordenadas
                    200, 200, 200, // Colores
                   tipoDeLetra);

          Hardware.EscribirTextoOculta("A.- Ayuda",
                    150, 430, // Coordenadas
                    200, 200, 200, // Colores
                    tipoDeLetra);

          Hardware.EscribirTextoOculta("C.- Créditos",
                    150, 470, // Coordenadas
                    200, 200, 200, // Colores
                    tipoDeLetra);

          Hardware.EscribirTextoOculta("S.- Salir",
                    150, 510, // Coordenadas
                    200, 200, 200, // Colores
                    tipoDeLetra);

          Hardware.VisualizarOculta();

          Hardware.Pausa(20);

          if (Hardware.TeclaPulsada(Hardware.TECLA_A))
               MostrarAyuda();

          if (Hardware.TeclaPulsada(Hardware.TECLA_C))
               MostrarCreditos();

          if (Hardware.TeclaPulsada(Hardware.TECLA_J))
               finPresentacion = true;
```

```
            if (Hardware.TeclaPulsada(Hardware.TECLA_S))
            {
                finPresentacion = true;
                partidaTerminada = true;
                juegoTerminado = true;
            }
        } while (!finPresentacion);
    }


    public static void MostrarAyuda()
    {
        string[] textosAyuda =
        {
          "Recoge los premios",
          "Evita los obstáculos y los enemigos",
          "Usa las flechas de cursor para mover"
        };

        // ---- Pantalla de presentación --
        Hardware.BorrarPantallaOculta(0, 0, 0);

        // Fondo de la presentación
        fondoAyuda.DibujarOculta(0, 0);

        // Marcador
        Hardware.EscribirTextoOculta("Ayuda",
                  340, 200, // Coordenadas
                  255, 255, 255, // Colores
                 tipoDeLetra);

        // Textos repetitivos
        short posicYtexto = 280;
        foreach (string texto in textosAyuda)
        {
            Hardware.EscribirTextoOculta(texto,
                       150, posicYtexto, // Coordenadas
                       200, 200, 200, // Colores
                      tipoDeLetra);
            posicYtexto += 30;
        }

        Hardware.EscribirTextoOculta("ESC- volver",
                  650, 530, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);
        Hardware.VisualizarOculta();

        do
        {
            Hardware.Pausa(20);
        } while (!Hardware.TeclaPulsada(Hardware.TECLA_ESC));
    }


    public static void MostrarCreditos()
    {
        // ---- Pantalla de presentación --
        Hardware.BorrarPantallaOculta(0, 0, 0);
```

```
    // Fondo de la presentación
    fondoCreditos.DibujarOculta(0, 0);

    // Marcador
    Hardware.EscribirTextoOculta("Creditos",
              250, 200, // Coordenadas
              255, 255, 255, // Colores
             tipoDeLetra);

    Hardware.EscribirTextoOculta("Por Nacho Cabanes, 2011",
               250, 300, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);

    Hardware.EscribirTextoOculta("ESC- volver",
              650, 530, // Coordenadas
              200, 200, 200, // Colores
             tipoDeLetra);
    Hardware.VisualizarOculta();

    do
    {
        Hardware.Pausa(20);
    } while (!Hardware.TeclaPulsada(Hardware.TECLA_ESC));
}


public static void Dibujar()
{
    // -- Dibujar --
    Hardware.BorrarPantallaOculta(0, 0, 0);

    // Marcador
    Hardware.EscribirTextoOculta("Vidas        Puntos",
              0, 0, // Coordenadas
              255, 255, 255, // Colores
             tipoDeLetra);

    Hardware.EscribirTextoOculta(Convert.ToString(vidas),
              70, 0, // Coordenadas
              200, 200, 200, // Colores
             tipoDeLetra);

    Hardware.EscribirTextoOculta(Convert.ToString(puntos),
              190, 0, // Coordenadas
              200, 200, 200, // Colores
             tipoDeLetra);

    for (int i = 0; i < numObstaculos; i++)  // Obstáculos
    {
        obstaculos[i].imagen.DibujarOculta(
            (int)obstaculos[i].x, (int)obstaculos[i].y);
    }

    for (int i = 0; i < numEnemigos; i++)  // Enemigos
    {
        enemigos[i].imagen.DibujarOculta(
            (int)enemigos[i].x, (int)enemigos[i].y);
```

```
        }

        for (int i = 0; i < numPremios; i++)  // Premios
        {
            if (premios[i].visible)
            {
                premios[i].imagen.DibujarOculta(
                    premios[i].x, premios[i].y);
            }
        }

        personaje.imagen.DibujarOculta(
            personaje.x, personaje.y);

        // Finalmente, muestro en pantalla
        Hardware.VisualizarOculta();
    }


    public static void ComprobarTeclas()
    {
        // -- Leer teclas y calcular nueva posición --
        if (Hardware.TeclaPulsada(Hardware.TECLA_ESC))
            partidaTerminada = true;

        if (Hardware.TeclaPulsada(Hardware.TECLA_DER))
            personaje.x += personaje.incrX;
        if (Hardware.TeclaPulsada(Hardware.TECLA_IZQ))
            personaje.x -= personaje.incrX;
        if (Hardware.TeclaPulsada(Hardware.TECLA_ARR))
            personaje.y -= personaje.incrY;
        if (Hardware.TeclaPulsada(Hardware.TECLA_ABA))
            personaje.y += personaje.incrY;
    }


    public static void MoverElementos()
    {
        // -- Mover enemigos, entorno --
        for (int i = 0; i < numEnemigos; i++)  // Enemigos
        {
            enemigos[i].x = enemigos[i].x + enemigos[i].incrX;
            if (((int)enemigos[i].x <= 50)
                    || ((int)enemigos[i].x >= 700))
                enemigos[i].incrX = -enemigos[i].incrX;
        }
    }


    public static void ComprobarColisiones()
    {
        // -- Colisiones, perder vidas, etc --
        for (int i = 0; i < numObstaculos; i++)  // Obstáculos
        {
            if (Colision(obstaculos[i], personaje))
            {
                vidas--;
                if (vidas == 0)
                    partidaTerminada = true;
```

```
                personaje.x = personaje.xInicial;
                personaje.y = personaje.yInicial;
            }
        }

        for (int i = 0; i < numPremios; i++)  // Premios
        {
            if (Colision(premios[i], personaje))
            {
                puntos += 10;
                premios[i].visible = false;
            }
        }

        for (int i = 0; i < numEnemigos; i++)  // Enemigos
        {
            if (Colision(enemigos[i], personaje))
            {
                vidas--;
                if (vidas == 0)
                    partidaTerminada = true;
                personaje.x = personaje.xInicial;
                personaje.y = personaje.yInicial;
            }
        }

}


public static void PausaFotograma()
{
    // -- Pausa hasta el siguiente "fotograma" del juego --
    Hardware.Pausa(20);
}


public static bool Colision(ElemGrafico e1, ElemGrafico e2)
{
    // No se debe chocar con un elemento oculto
    if ((e1.visible == false) || (e2.visible == false))
        return false;
    // Ahora ya compruebo coordenadas
    if ((e1.x + e1.ancho > e2.x)
          && (e1.x < e2.x + e2.ancho)
          && (e1.y + e1.alto > e2.y)
          && (e1.y < e2.y + e2.alto))
        return true;
    else
        return false;
}



public static void Main()
{

    InicializarJuego();

    while (!juegoTerminado)
```

```
        {
            InicializarPartida();
            MostrarPresentacion();

            // ------ Bucle de juego ------
            while (!partidaTerminada)
            {
                Dibujar();
                ComprobarTeclas();
                MoverElementos();
                ComprobarColisiones();
                PausaFotograma();
            } // Fin del bucle de juego
        }     // Fin de partida
    }         // Fin de Main
}
```

Ejercicio propuesto (1): Haz que cuando se recojan todos los premios, vuelvan a aparecer otros en nuevas posiciones al azar. Crea para ello una función llamada "RecolocarPremios".

Ejercicio propuesto (2): Añade un marcador de "mejor puntuación". Al terminar una partida, si se ha superado esta "mejor puntuación", deberá actualizarse su valor.

25. Cómo capturar imágenes para hacer un remake

El hecho de crear un "remake" de un juego ya existente, en vez de diseñar un nuevo juego desde cero es interesante cuando uno está aprendiendo a programar, por varios motivos:

- No hay que "diseñar una lógica de juego" desde cero, sino imitar algo ya existente.
- Precisamente, al tener algo que imitar, existe una serie de retos a superar: igualar cada una de las funcionalidades del juego original.
- Existe algo con lo que comparar: el juego estará más "terminado" cuanto más "se acerque" al juego original (lo que no impide que añadamos alguna posibilidad nueva).
- No hay que "diseñar pantallas" ni "diseñar personajes", sino que se pueden "capturar" las imágenes del juego original, y centrarse en las tareas que son estrictamente de programación.

Por eso, vamos a ver los pasos básicos para capturar imágenes de juegos clásicos, de forma que las podamos aplicar a crear un "remake" con un ordenador moderno, respetando la estética del juego original.

Los pasos serán:

- Lanzar el juego desde un emulador.
- Capturar las pantallas que nos interesen.
- Redimensionar hasta la resolución del equipo actual.
- Extraer los fragmentos que nos interesen.
- Hacer "transparentes" las imágenes, suavizar contornos, etc.

Como herramientas gráficas usaremos:

- XnView para casi todos los pasos, porque permite capturar múltiples imágenes con facilidad, redimensionar varias a la vez, recortar la zona que nos interese...
- GIMP para añadir transparencia (los otros pasos, que también se pueden dar con GIMP, son más sencillos con XnView, pero este último paso, por el contrario, es más "visual" y potente en GIMP que en XnView).

Lo aplicaremos a un remake de un juego de Amstrad CPC (con una resolución de 320x200 puntos), que convertiremos a modo gráfico 800x600 puntos, de modo que se aproveche más la resolución de un ordenador actual, pero que permita usarlo en un ordenador menos potente, como un "netbook", que suelen tener una resolución de 1024x600 puntos.

Los pasos que daremos serán:

1.- Lanzar el juego desde un emulador.

Para hacer capturas de un juego de Amstrad CPC, el emulador que más me gusta es CPCE, porque captura las imágenes tal como eran en el original, "pixeladas", mientras que otros emuladores como

WinApe, "suavizan" los contornos al exportar las imágenes, lo que supone que pierda algo de estética retro y que sea más difícil conseguir una transparencia perfecta en los contornos.

Los pasos concretos dependen de la distribución que se use. Por ejemplo, en el recopilatorio CpcGamesCd se muestra la lista de juegos, y al lanzar uno de ellos (con Intro o con doble clic) se abrirá el emulador y luego el juego; el riesgo puede ser que se lance un emulador que no es el que queremos, pero eso es fácil de solucionar: desde el menú Emulador, escogemos el que nos apetezca emplear.

Si se usa el emulador CPCE directamente, habría que pulsa F7 para abrir una imagen de disco (un fichero con extensión DSK, quizá comprimido en un fichero ZIP). En la mayoría de los casos, el juego se pondrá en marcha automáticamente.

2.- Capturar las pantallas.

Podemos usar herramientas de captura en Windows (XnView o cualquier otra), o bien la propia tecla ImprPant para luego pegar en algún programa de manipulación de imágenes, pero si queremos una imagen más fiable, lo ideal será pedir a nuestro emulador que sea él quien guarde la pantalla. Desde CPCE, esto se consigue pulsando la tecla F12, y se irán creando imágenes con nombres 0001.BMP, 0002.BMP y sucesivos.

Estas imágenes tendrán tamaño 768x576, que corresponden a 640x400 + borde de pantalla, si nuestro CPCE está configurado para trabajar a doble tamaño (si aparece IMAGE_DOUBLE=1 en el fichero CPCE.INI), o bien tamaño 384x288 (320x200 + borde) si está a tamaño normal (IMAGE_DOUBLE=0 en CPCE.INI)

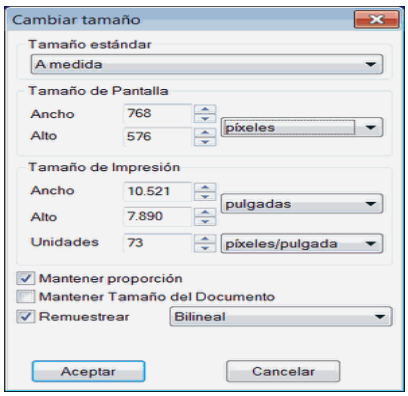
3.- Redimensionar hasta la resolución del equipo actual.

El tamaño obtenido (640x400 o, habitualmente, 320x200) es más pequeño que lo que queríamos para nuestro juego de destino, así que habrá que hacer la imagen más grande.

Se podría casi usar cualquier programa de manipulación de imágenes, desde alternativas gratuitas como XnView, Paint.net o GIMP hasta aplicaciones profesionales como Photoshop.

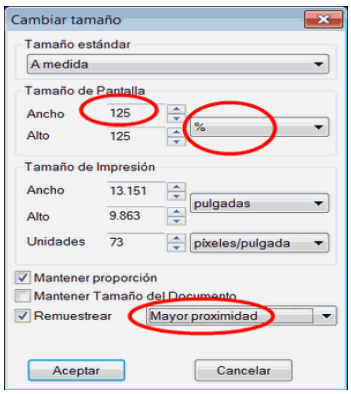
Vamos a ver cómo hacerlo con XnView, que es gratis, de pequeño tamaño, muy cómodo como visor y para manipulaciones simples, y además permite hacer dichas manipulaciones a muchas imágenes a la vez, "en lote", lo que puede ser una gran ganancia de tiempo si tenemos muchas imágenes que manipular.

Para redimensionar desde XnView la imagen que estamos viendo, podemos entrar al menú "Imagen" y escoger "Cambiar tamaño" (o usar la tecla rápida Mays+S). Veremos algo como esto:

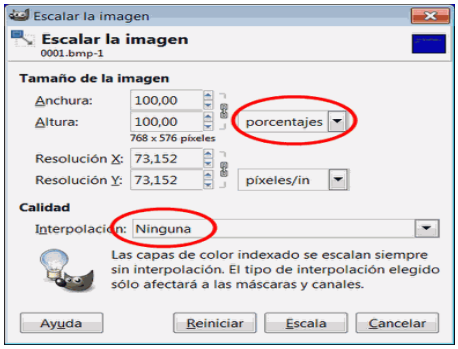


Podemos elegir el nuevo tamaño en píxeles o en porcentaje. Si partimos de una imagen de 320x200 (o 384x288 incluyendo borde), que queremos redimensionar para usarla en un juego de 800x600, deberíamos usar un 250% como tamaño final. Si el original era de 640x400 (o 768x576 con borde), deberíamos redimensionarlo en un 125%, para obtener una zona de juego de 800x500.

Y si queremos que no se suavicen los bordes, para conservar el estilo retro y para que nos sea más fácil conseguir el efecto de transparencia, deberíamos decirle que el método de redimensionar sea el de "Mayor proximidad":

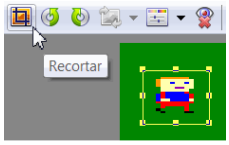


Si usamos GIMP, la opción equivalente se llama "Escalar imagen", dentro del menú "Imagen", y, al igual que XnView, nos permite indicar el tamaño final en píxeles o en porcentaje, y distintas formas de interpolar la imagen ("inventar" los puntos nuevos que aparezcan), de las cuales nos interesaría la llamada "ninguna":



4.- Extraer los fragmentos que nos interesen.

Nuevamente, bastaría cualquiera cualquiera los programas mencionados. En XnView bastaría con "pinchar y arrastrar" con el ratón sobre la imagen para seleccionar la zona que nos interesa, y luego hacer clic en el botón de Recortar (y guardar los cambios, claro):



Para conseguirlo en GIMP, comenzaríamos por escoger la herramienta de recorte (la que tiene una imagen que recuerda una cuchilla). Después pincharíamos y arrastraríamos con el ratón para seleccionar la zona que queremos conservar, y haríamos doble clic para completar la operación.

Si guardamos esta imagen (con otro nombre, para no perder la original), ya tendríamos una imagen de nuestro personaje (o un enemigo, o un elemento del fondo...), a un tamaño adecuado para nuestro juego, que podríamos cargar (y mostrar) desde nuestro programa.

Ejercicio propuesto: Captura y recorta imágenes de un juego (de plataformas o de complejidad similar) que quieras imitar, para incluirlas en tu juego.

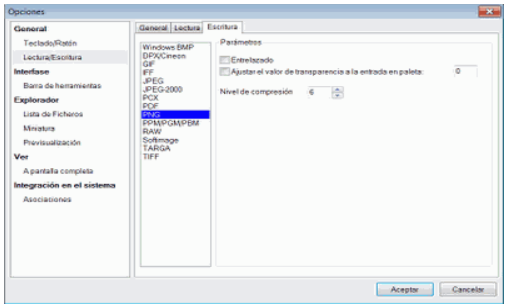
26. Imágenes transparentes

Es habitual que cuando "creamos" una imagen, ésta tenga fondo negro (o blanco, o de algún otro color). Pero cuando esa imagen con fondo negro pasa por encima de otra, "la tapa", no permitiendo ver qué imagen se encuentra debajo.

Vamos a ver de qué formas se puede hacer que la imagen superior tenga "huecos", de modo que se vea el fondo a través de ella...

Este paso depende de la biblioteca gráfica que usemos en nuestro juego: si empleamos Allegro, que en su versión más extendida utiliza imágenes en formato BMP, las zonas transparentes se indican rellenándolas con un "color clave", un color especial que luego no se dibujará en pantalla; por el contrario, si utilizamos SDL, que permite usar imágenes en formato PNG, será más razonable usar la transparencia propia de este formato.

Crear un PNG con transparencia es más incómodo y menos versátil en XnView que en GIMP: si usamos XnView tendremos que ir al menú "Archivo" y escoger la opción "Guardar como". Si escogemos el formato PNG, podremos indicar cual de los colores de la paleta queremos que se convierta en transparente:

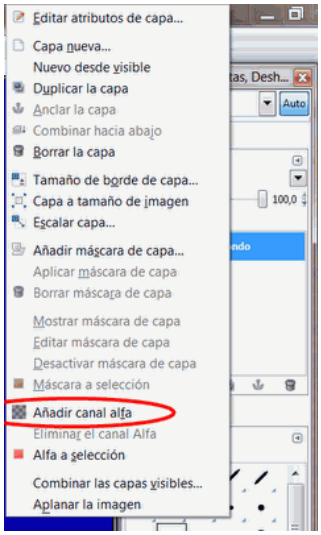


Esto tiene dos inconvenientes: por una parte, tendríamos que saber cual es el color que "sobra" (nos puede ayudar el menú "Imagen", en la opción "Modificar mapa de color", pero si hay varios colores parecidos, puede no ser de gran ayuda). Por otra parte, todo ese color se volvería transparente, y quizá eso no nos interese, sino que queramos que unas zonas sean transparentes y otras no.

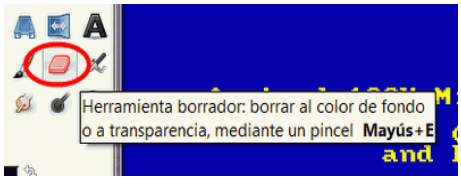
Es más cómodo y más potente hacerlo desde GIMP, porque podemos hacer que ciertas zonas sean transparentes y otras no lo sean. Los pasos a seguir serían:

0.- Abrir la imagen en The GIMP.

1.- En la ventana de capas hacer Clic con el botón derecho del ratón sobre la única capa que tiene nuestra imagen, y escoger la opción "Añadir canal alfa":



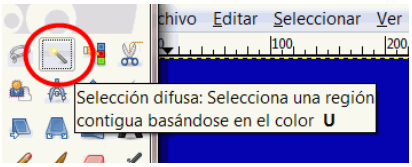
2.- En la barra de herramientas escoger la herramienta "Borrador":



Si comenzamos a borrar con ella, lo que ya no es parte de la imagen (sino del fondo transparente) debería verse como un trama de cuadraditos grises claros y grises oscuros:

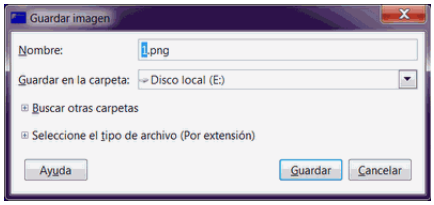
Generalmente será preferible no borrar "a pulso", así que podemos...

3.- Escoger la herramienta de "Selección difusa" (la "varita mágica")

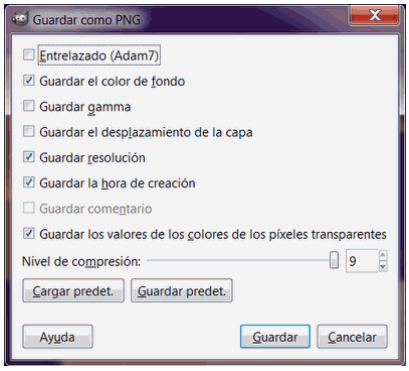


Hacer clic en una parte del fondo que queramos borrar. Todos los píxeles colindantes que sean del mismo color se quedarán seleccionados también, así que podemos usar un pincel más grande para borrar rápidamente:

4.- Cuando la imagen ya está a nuestro gusto, vamos a "Guardar como" (en el menú Archivo) y le damos un nombre que termine en ".png", para que GIMP sepa que queremos usar dicho formato gráfico, que sí permite transparencia, aunque nuestra imagen original fuera un BMP o cualquier otra que no la permitiera:



Y si todo ha ido bien, nos preguntará ciertos detalles adicionales (que normalmente no será necesario cambiar), entre los que está si queremos guardar los datos de transparencia:



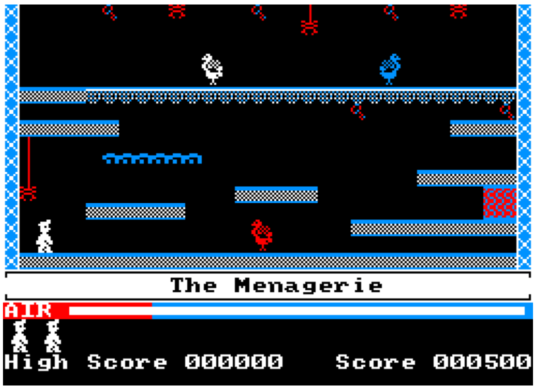
(Estos pasos serían muy parecidos si en vez de emplear GIMP, usamos Paint.Net, que es gratuito y aún más sencillo de manejar), o si utilizamos Photoshop (que es comercial y más potente) u otras muchas herramientas de retoque de imágenes.

Ejercicio propuesto: Haz transparente el borde o alguna zona interior de alguna de las imágenes que has capturado para tu juego en el apartado anterior.

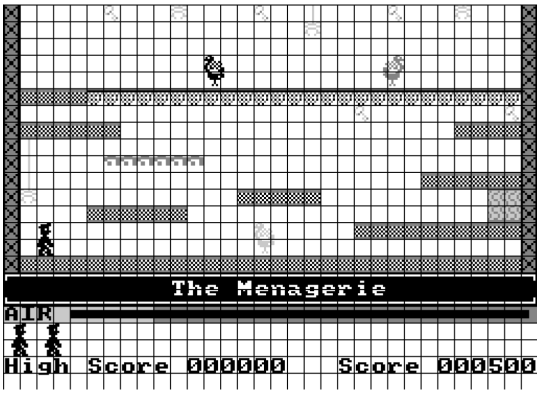
27. Un mapa para un nivel

En muchos juegos de plataformas "clásicos", la memoria era un bien tan escaso que había que optimizarla como fuera posible. Una de las formas más simples es hacer que el fondo sea una estructura repetitiva.

Vamos a ver un ejemplo real: en el juego Manic Miner, ésta era la apariencia de uno de los niveles:



Si superponemos una cuadrícula de 8x8 (en el juego original, o de 16x16 en esta imagen ampliada), vemos claramente que el fondo es una serie de casillas repetitivas (que llamaremos "tiles").



Vamos a usar esta técnica para incorporar un fondo a nuestro juego...

Ese fondo se podría plantear como un array de dos dimensiones, en el que un número 0 indicara que no hay que dibujar nada, y un 1 detallara dónde debe haber un fragmento de pared (obviamente, se podrían usar más números para indicar más tipos de casillas repetitivas en el fondo, pero nosotros no lo haremos por ahora):

```
static public byte[,] fondo =
{
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
};
```

Y serían necesarios ciertos datos adicionales, como el ancho y alto de estas casillas, el margen que debemos dejar por encima y a su izquierda cuando las dibujemos (en caso de que no vayan a ocupar toda la pantalla), y las nuevas imágenes que representen esos "tiles":

```
static byte anchoFondo = 20;
static byte altoFondo = 16;
static short margenXFondo = 80;
static byte margenYFondo = 30;
static byte anchoCasillaFondo = 32;
static byte altoCasillaFondo = 32;
static Imagen imgPared;
```

Así, a la hora de dibujar, bastaría recorrer el array bidimensional con dos "for" anidados, dibujando el "tile" correspondiente cuando en esa posición del array no haya un 0 sino otro número:

```
public static void Dibujar()
{
     // Pared de fondo
     for (int fila = 0; fila < altoFondo; fila++)  // Fondo
          for (int col = 0; col < anchoFondo; col++)
               if (fondo[fila, col] == 1)
                    imgPared.DibujarOculta(
                         margenXFondo + col * anchoCasillaFondo,
                         margenYFondo + fila * altoCasillaFondo);
     ...
```

Ahora la apariencia sería ésta:

Si tuviéramos unos cuantos tipos de casillas, usaríamos un "switch", en vez de varios "if" seguidos.

El fuente completo sería:

```
// Primer mini-esqueleto de juego en modo gráfico
// Versión "g"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego05g
{
     public struct ElemGrafico
     {
          public int x;
          public int y;
```

```
    public int xInicial;
    public int yInicial;
    public int ancho;
    public int alto;
    public int incrX;
    public int incrY;
    public Imagen imagen;
    public bool visible;
}

static byte anchoFondo = 20;
static byte altoFondo = 16;
static short margenXFondo = 80;
static byte margenYFondo = 30;
static byte anchoCasillaFondo = 32;
static byte altoCasillaFondo = 32;
static Imagen imgPared;
static public byte[,] fondo =
{
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
};


static ElemGrafico personaje;
static Fuente tipoDeLetra;

static int numPremios, numEnemigos, numObstaculos;

static ElemGrafico[] obstaculos;
static ElemGrafico[] enemigos;
static ElemGrafico[] premios;

static bool juegoTerminado;
static int vidas;
static int puntos;
static bool partidaTerminada;
static Random generador;

static Imagen fondoPresentacion;
static Imagen fondoAyuda;
static Imagen fondoCreditos;


public static void InicializarJuego()
```

```
{
    // Entrar a modo grafico 800x600
    bool pantallaCompleta = false;
    Hardware.Inicializar(800, 600, 24, pantallaCompleta);

    // Resto de inicializacion
    tipoDeLetra = new Fuente("FreeSansBold.ttf", 18);
    juegoTerminado = false;
    numPremios = 10;
    numEnemigos = 10;
    numObstaculos = 20;
    obstaculos = new ElemGrafico[numObstaculos];
    enemigos = new ElemGrafico[numEnemigos];
    premios = new ElemGrafico[numPremios];
    generador = new Random();

    // Cargo imágenes de elementos
    personaje.imagen = new Imagen("personaje.png");

    for (int i = 0; i < numObstaculos; i++)  // Obstaculos
        obstaculos[i].imagen = new Imagen("obstaculo.png");

    for (int i = 0; i < numEnemigos; i++)  // Enemigos
        enemigos[i].imagen = new Imagen("enemigo.png");

    for (int i = 0; i < numPremios; i++)  // Premios
        premios[i].imagen = new Imagen("premio.png");

    imgPared = new Imagen("pared.png");

    // Y cargo las imagenes de la presentación, ayuda y créditos
    fondoPresentacion = new Imagen("present.jpg");
    fondoAyuda = new Imagen("present.jpg");
    fondoCreditos = new Imagen("present.jpg");
}


public static void InicializarPartida()
{
    // En cada partida, hay que reiniciar ciertas variables
    vidas = 3;
    puntos = 0;
    partidaTerminada = false;

    personaje.xInicial = 400;
    personaje.yInicial = 300;
    personaje.x = personaje.xInicial;
    personaje.y = personaje.yInicial;
    personaje.visible = true;
    personaje.ancho = 32;
    personaje.alto = 30;
    personaje.incrX = 10;
    personaje.incrY = 10;

    // Genero las posiciones de los elementos al azar
    for (int i = 0; i < numObstaculos; i++)  // Obstaculos
    {
        obstaculos[i].visible = true;
        obstaculos[i].ancho = 38;
```

```
            obstaculos[i].alto = 22;
            // Al colocar un obstáculo, compruebo que no choque con el personaje, para que la partida
            // no acabe nada más empezar
            do
            {
                obstaculos[i].x = generador.Next(50, 700);
                obstaculos[i].y = generador.Next(30, 550);
            } while (Colision(obstaculos[i], personaje));
        }

        for (int i = 0; i < numEnemigos; i++)  // Enemigos
        {
            enemigos[i].x = generador.Next(50, 700);
            // Para la Y, compruebo que no choque con el personaje, para que la partida
            // no acabe nada más empezar
            do
            {
                enemigos[i].y = generador.Next(30, 550);
            } while ((enemigos[i].y+enemigos[i].alto > personaje.y) && (enemigos[i].y <
personaje.y+personaje.alto) );
            enemigos[i].incrX = 5;
            enemigos[i].visible = true;
            enemigos[i].ancho = 36;
            enemigos[i].alto = 42;
        }

        for (int i = 0; i < numPremios; i++)  // Premios
        {
            premios[i].x = generador.Next(50, 700);
            premios[i].y = generador.Next(30, 550);
            premios[i].visible = true;
            premios[i].ancho = 34;
            premios[i].alto = 18;
        }
    }


    public static void MostrarPresentacion()
    {
        bool finPresentacion = false;

        do
        {
            // ---- Pantalla de presentación --
            Hardware.BorrarPantallaOculta(0, 0, 0);

            // Fondo de la presentación
            fondoPresentacion.DibujarOculta(0, 0);

            // Marcador
            Hardware.EscribirTextoOculta("Jueguecillo",
                      340, 200, // Coordenadas
                      255, 255, 255, // Colores
                      tipoDeLetra);

            Hardware.EscribirTextoOculta("Escoja una opción:",
                      310, 300, // Coordenadas
                      200, 200, 200, // Colores
                      tipoDeLetra);
```

```
            Hardware.EscribirTextoOculta("J.- Jugar una partida",
                      150, 390, // Coordenadas
                      200, 200, 200, // Colores
                     tipoDeLetra);

            Hardware.EscribirTextoOculta("A.- Ayuda",
                      150, 430, // Coordenadas
                      200, 200, 200, // Colores
                      tipoDeLetra);

            Hardware.EscribirTextoOculta("C.- Créditos",
                      150, 470, // Coordenadas
                      200, 200, 200, // Colores
                      tipoDeLetra);

            Hardware.EscribirTextoOculta("S.- Salir",
                      150, 510, // Coordenadas
                      200, 200, 200, // Colores
                      tipoDeLetra);

            Hardware.VisualizarOculta();

            Hardware.Pausa(20);

            if (Hardware.TeclaPulsada(Hardware.TECLA_A))
                 MostrarAyuda();

            if (Hardware.TeclaPulsada(Hardware.TECLA_C))
                 MostrarCreditos();

            if (Hardware.TeclaPulsada(Hardware.TECLA_J))
                 finPresentacion = true;

            if (Hardware.TeclaPulsada(Hardware.TECLA_S))
            {
                 finPresentacion = true;
                 partidaTerminada = true;
                 juegoTerminado = true;
            }
       } while (!finPresentacion);
  }


  public static void MostrarAyuda()
  {
       string[] textosAyuda =
       {
         "Recoge los premios",
         "Evita los obstáculos y los enemigos",
         "Usa las flechas de cursor para mover"
       };

       // ---- Pantalla de presentación --
       Hardware.BorrarPantallaOculta(0, 0, 0);

       // Fondo de la presentación
       fondoAyuda.DibujarOculta(0, 0);
```

```
        // Marcador
        Hardware.EscribirTextoOculta("Ayuda",
                  340, 200, // Coordenadas
                  255, 255, 255, // Colores
                 tipoDeLetra);

        // Textos repetitivos
        short posicYtexto = 280;
        foreach (string texto in textosAyuda)
        {
            Hardware.EscribirTextoOculta(texto,
                       150, posicYtexto, // Coordenadas
                       200, 200, 200, // Colores
                      tipoDeLetra);
            posicYtexto += 30;
        }

        Hardware.EscribirTextoOculta("ESC- volver",
                  650, 530, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);
        Hardware.VisualizarOculta();

        do
        {
            Hardware.Pausa(20);
        } while (!Hardware.TeclaPulsada(Hardware.TECLA_ESC));
    }


    public static void MostrarCreditos()
    {
        // ---- Pantalla de presentación --
        Hardware.BorrarPantallaOculta(0, 0, 0);

        // Fondo de la presentación
        fondoCreditos.DibujarOculta(0, 0);

        // Marcador
        Hardware.EscribirTextoOculta("Creditos",
                  250, 200, // Coordenadas
                  255, 255, 255, // Colores
                 tipoDeLetra);

        Hardware.EscribirTextoOculta("Por Nacho Cabanes, 2011",
                   250, 300, // Coordenadas
                   200, 200, 200, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("ESC- volver",
                  650, 530, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);
        Hardware.VisualizarOculta();

        do
        {
            Hardware.Pausa(20);
        } while (!Hardware.TeclaPulsada(Hardware.TECLA_ESC));
```

```
}


public static void Dibujar()
{
     // -- Dibujar --
     Hardware.BorrarPantallaOculta(0, 0, 0);

     // Marcador
     Hardware.EscribirTextoOculta("Vidas        Puntos",
               0, 0, // Coordenadas
               255, 255, 255, // Colores
              tipoDeLetra);

     Hardware.EscribirTextoOculta(Convert.ToString(vidas),
               70, 0, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);

     Hardware.EscribirTextoOculta(Convert.ToString(puntos),
               190, 0, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);

     // Pared de fondo
     for (int fila = 0; fila < altoFondo; fila++)  // Fondo
          for (int col = 0; col < anchoFondo; col++)
               if (fondo[fila, col] == 1)
                    imgPared.DibujarOculta(
                         margenXFondo + col * anchoCasillaFondo,
                         margenYFondo + fila * altoCasillaFondo);

     for (int i = 0; i < numObstaculos; i++)  // Obstáculos
     {
          obstaculos[i].imagen.DibujarOculta(
               (int)obstaculos[i].x, (int)obstaculos[i].y);
     }

     for (int i = 0; i < numEnemigos; i++)  // Enemigos
     {
          enemigos[i].imagen.DibujarOculta(
               (int)enemigos[i].x, (int)enemigos[i].y);
     }

     for (int i = 0; i < numPremios; i++)  // Premios
     {
          if (premios[i].visible)
          {
               premios[i].imagen.DibujarOculta(
                    premios[i].x, premios[i].y);
          }
     }

     personaje.imagen.DibujarOculta(
          personaje.x, personaje.y);

     // Finalmente, muestro en pantalla
     Hardware.VisualizarOculta();
}
```

```
public static void ComprobarTeclas()
{
     // -- Leer teclas y calcular nueva posición --
     if (Hardware.TeclaPulsada(Hardware.TECLA_ESC))
          partidaTerminada = true;

     if (Hardware.TeclaPulsada(Hardware.TECLA_DER))
          personaje.x += personaje.incrX;
     if (Hardware.TeclaPulsada(Hardware.TECLA_IZQ))
          personaje.x -= personaje.incrX;
     if (Hardware.TeclaPulsada(Hardware.TECLA_ARR))
          personaje.y -= personaje.incrY;
     if (Hardware.TeclaPulsada(Hardware.TECLA_ABA))
          personaje.y += personaje.incrY;
}


public static void MoverElementos()
{
     // -- Mover enemigos, entorno --
     for (int i = 0; i < numEnemigos; i++)  // Enemigos
     {
          enemigos[i].x = enemigos[i].x + enemigos[i].incrX;
          if (((int)enemigos[i].x <= 50)
                    || ((int)enemigos[i].x >= 700))
               enemigos[i].incrX = -enemigos[i].incrX;
     }
}


public static void ComprobarColisiones()
{
     // -- Colisiones, perder vidas, etc --
     for (int i = 0; i < numObstaculos; i++)  // Obstáculos
     {
          if (Colision(obstaculos[i], personaje))
          {
               vidas--;
               if (vidas == 0)
                    partidaTerminada = true;
               personaje.x = personaje.xInicial;
               personaje.y = personaje.yInicial;
          }
     }

     for (int i = 0; i < numPremios; i++)  // Premios
     {
          if (Colision(premios[i], personaje))
          {
               puntos += 10;
               premios[i].visible = false;
          }
     }

     for (int i = 0; i < numEnemigos; i++)  // Enemigos
     {
          if (Colision(enemigos[i], personaje))
```

```
            {
                vidas--;
                if (vidas == 0)
                    partidaTerminada = true;
                personaje.x = personaje.xInicial;
                personaje.y = personaje.yInicial;
            }
        }

    }


    public static void PausaFotograma()
    {
        // -- Pausa hasta el siguiente "fotograma" del juego --
        Hardware.Pausa(20);
    }


    public static bool Colision(ElemGrafico e1, ElemGrafico e2)
    {
        // No se debe chocar con un elemento oculto
        if ((e1.visible == false) || (e2.visible == false))
            return false;
        // Ahora ya compruebo coordenadas
        if ((e1.x + e1.ancho > e2.x)
              && (e1.x < e2.x + e2.ancho)
              && (e1.y + e1.alto > e2.y)
              && (e1.y < e2.y + e2.alto))
            return true;
        else
            return false;
    }



    public static void Main()
    {

        InicializarJuego();

        while (!juegoTerminado)
        {
            InicializarPartida();
            MostrarPresentacion();

            // ------ Bucle de juego ------
            while (!partidaTerminada)
            {
                Dibujar();
                ComprobarTeclas();
                MoverElementos();
                ComprobarColisiones();
                PausaFotograma();
            } // Fin del bucle de juego
        }     // Fin de partida
    }         // Fin de Main
}
```

Esta forma de trabajar, en la que tenemos un array de "bytes" y sólo hemos cargado la imagen una vez, para irla dibujando en distintas posiciones, permite aprovechar mejor la memoria que si fuera un array de "elementos gráficos", cada uno con su imagen (repetitiva) y demás características. A cambio, complica un poco la detección de colisiones con el fondo. Por eso, como en los ordenadores actuales, la memoria no es tan escasa como en los años 80 (que es cuando se crearon la mayoría de los juegos que estamos tomando como ejemplo), veremos una versión alternativa, en la que a partir de ese array de dos dimensiones se cree una serie de elementos gráficos que representen los objetos del fondo.

Ejercicio propuesto: Amplía esta versión del juego, para que el fondo no esté formato por un único tipo de casilla, sino por dos o más.

28. Descubrir tiles en un juego

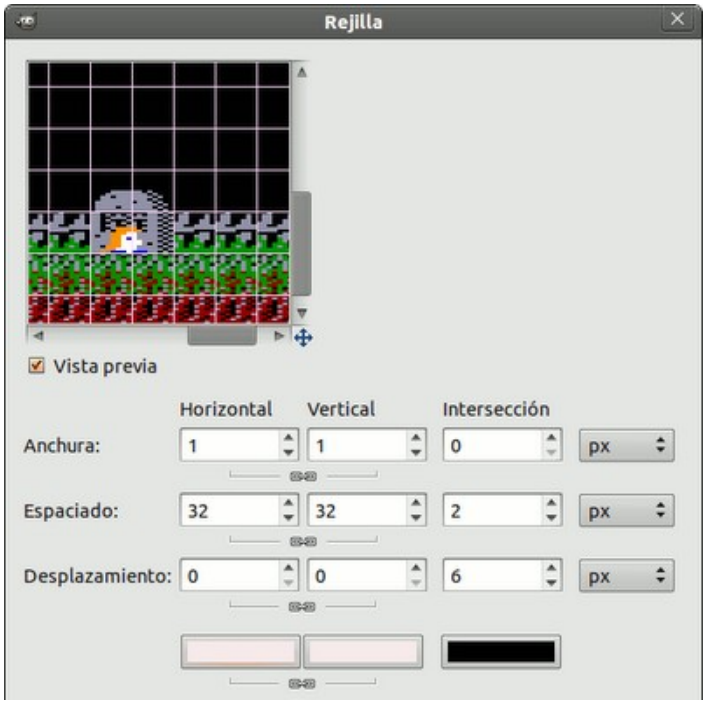
Como hemos visto en el apartado anterior, si estamos pensando hacer un remake, o al menos tomar ideas de un juego clásico, nos puede interesar tratar de descubrir qué casillas repetitivas (tiles) formaban la pantalla de fondo de ese juego.

GIMP tiene una opción que puede ser de gran ayuda para conseguirlo. Se trata del filtro de "rejilla", que superpone una cuadrícula a nuestra imagen.

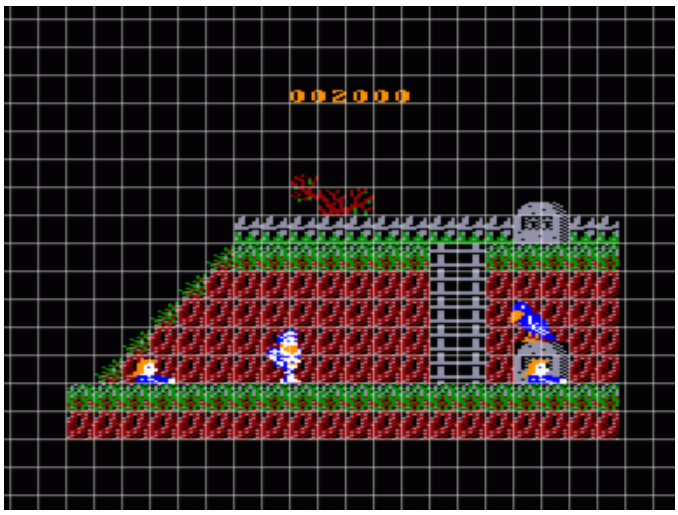
Lo podemos encontrar en el menú Filtros, dentro de la opción Filtros -> Renderizado -> Patrón -> Rejilla.

Nos preguntará:

- El espaciado horizontal y vertical, que normalmente será 8 o 16 píxeles en una imagen de un juego de 8 bits que no se haya redimensionado, o bien 16 o 32 píxeles si la imagen está a doble tamaño, como ocurre con algunas que exporta CPCE y algún otro emulador.
- El desplazamiento inicial, tanto en horizontal como en vertical, que suele ser 0.
- La anchura de las líneas, que debería bastar con 1 píxel, para que no nos tape mucho de la imagen original.
- El color de las líneas, que depende de cómo sea el juego: en los que tienen una pantalla predominantemente negra, podríamos dibujar líneas blancas, amarillas o incluso rojas.



El resultado debería ser algo parecido a:



En el que se ve claramente las casillas repetitivas que forman el fondo.

Si ahora queremos extraer esas casillas para usarlas nosotros en nuestro remake, tenemos dos opciones (ambas partiendo de la imagen original, no de la recuadrada):

- Extraer los fragmentos que nos interesan, con un poco de paciencia y la herramienta de recorte, como ya vimos.
- Crear un programita que extraiga todos los tiles por nosotros, y luego eliminamos los que estén repetidos.

Ese tipo de programas puede ser muy fácil de hacer, si nos apoyamos en herramientas como "nconvert" (parte de XnView: www.xnview.com), que sean capaces de convertir una imagen a PNG y de cambiarle el tamaño o recortarla, todo ello a partir de los parámetros que le indiquemos en línea de comandos. Así, nuestro programa simplemente tendría que llamar a "nconvert" (o la herramienta similar que escojamos) de forma repetitiva, indicando qué fragmento queremos extraer cada vez.

Un programa básico en C# que nos ayudara en esta tarea podría ser así (**nota**: este fuente tiene algunos detalles avanzados que aún no hemos visto, como puede ser la copia de ficheros, el llamar a otros programas o la lectura de los parámetros que se indican en línea de comandos; tómalo como una ayuda, sin tratar de entender todo lo que hace):

```
// Recortador de imágenes en tiles
// (se apoya en nconvert)

using System;              // WriteLine
using System.IO;           // Path
using System.Diagnostics;  // Process

public class RecortadorDeFicheros
{
  public static void Main( string[] args )
  {
    if (args.Length < 1)
    {
      Console.WriteLine("Indica el nombre de fichero a recortar");
      return;
    }

    int anchoImagen = 768;
    int altoImagen = 576;

    int anchoTile = 32;
    int altoTile = 32;

    int cantidadColumnas = anchoImagen / anchoTile;
    int cantidadFilas = altoImagen / altoTile;

    string nombre = args[0];
    string nombreSalida = Path.GetFileNameWithoutExtension(nombre);
    string extension = Path.GetExtension(nombre);
```

```
    for (int fila = 0; fila < cantidadFilas; fila++)
      for (int columna = 0; columna < cantidadColumnas; columna++)
      {
        Console.WriteLine("Fragmento: "+fila+","+columna);
        string filaTexto = fila.ToString("00");
        string columnaTexto = columna.ToString("00");
        string nombreFinal = nombreSalida + filaTexto +
          columnaTexto + "." + extension;
        File.Copy(nombre, nombreFinal);
        Process.Start("nconvert.exe",
          "-out png -D -crop "
          +columna*anchoTile + " "  // x
          +fila*altoTile + " "      // y
          +anchoTile + " "          // w
          +altoTile + " "           // h
          +nombreFinal
          );
      }
  }
}
```

29. Paredes que no se pueden atravesar

Hemos visto como dibujar un fondo formado a partir de "tiles" (casillas repetitivas). Ahora vamos a ver cómo comprobar colisiones con los elementos que forman ese fondo, de modo que no se puedan atravesar las paredes.

La primera idea es que tendremos que comprobar si es posible movernos a una cierta posición antes de desplazarnos a ella. Esa posición será la misma que ocupa el personaje, pero ligeramente desplazada. Por ejemplo, si nuestro personaje ocupa un rectángulo que empieza en las coordenadas (x,y) y termina en (x+ancho,y+alto), cuando se mueva hacia la derecha pasará a ocupar un rectángulo que empezará en (x+incrX,y) y termina en (x+ancho+incrX,y+alto). Del mismo modo, podemos saber qué rectángulo ocupará si se mueve a izquierda (-incrX), hacia arriba (-incrY) o hacia abajo (+incrY).

Así, podemos crear una función "EsPosibleMover", que reciba esas coordenadas mínimas y máximas del rectángulo, y compruebe si esa es una posición válida para nuestro personaje.

Esa función "EsPosibleMover" se limitaría a comprobar si chocamos con alguno de los elementos del fondo (y en ese caso no podremos mover, luego "EsPosibleMover" devolverá "false") o si no chocamos con nada (y entonces sí podremos mover, por lo que devolverá "true"):

```
public static bool EsPosibleMover(int x, int y, int xFin, int yFin)
{
  // Veo si choca con algún elemento del fondo
  for (int fila = 0; fila < altoFondo; fila++)  // Fondo
    for (int col = 0; col < anchoFondo; col++)
      if ((fondo[fila, col] == 1) &&  Colision(
          x,y,xFin,yFin,  // Posicion del personaje
          margenXFondo + col * anchoCasillaFondo,  // x inicial de casilla actual de fondo
          margenYFondo + fila * altoCasillaFondo,  // y inicial
          margenXFondo + (col+1) * anchoCasillaFondo,  // x final
          margenYFondo + (fila+1) * altoCasillaFondo))  // y final
        return false;

  // Si no ha chocado con ninguno, es posible moverse a esa posición
  return true;
}
```

Hasta ahora sólo teníamos una función "Colision" que comprobaba si chocaban dos "elementos gráficos"; como nuestro fondo no está formado por elementos gráficos, sino que nos limitamos a dibujar una imagen de forma repetitiva, necesitaríamos

una variante de la función "Colisión" que sea capaz de comprobar si chocan dos rectángulos, sean elementos gráficos o no. Podría ser así:

```
public static bool Colision(
     int x1, int y1, int xFin1, int yFin1,
     int x2, int y2, int xFin2, int yFin2)
{
     if ((xFin2 > x1)
            && (x2 < xFin1)
            && (yFin2 > y1)
            && (y2 < yFin1))
          return true;
     else
          return false;
}
```

Como se puede ver, en C# (y en otros muchos lenguajes modernos) podemos tener dos funciones que se llaman igual pero que reciben distintos parámetros. No es problema, el compilador será capaz de deducir a cual de ellas llamamos en cada caso.

Finalmente, como ya habíamos anticipado, a la hora de comprobar teclas, no cambiaremos el valor de X o Y de nuestro personaje hasta que no veamos si realmente podemos mover a una cierta posición:

```
if (Hardware.TeclaPulsada(Hardware.TECLA_DER)
     && EsPosibleMover(personaje.x + personaje.incrX, personaje.y,
               personaje.x + personaje.ancho + personaje.incrX,
               personaje.y + personaje.alto))
          personaje.x += personaje.incrX;

if (Hardware.TeclaPulsada(Hardware.TECLA_IZQ)
     && EsPosibleMover(personaje.x - personaje.incrX, personaje.y,
               personaje.x + personaje.ancho - personaje.incrX,
               personaje.y + personaje.alto))
          personaje.x -= personaje.incrX;

...
```

Con esos cambios, el fuente completo quedaría así:

```
// Primer mini-esqueleto de juego en modo gráfico
// Versión "h1"

using System;
using System.Threading; // Para Thread.Sleep

public class JuegoO5h1
{
     public struct ElemGrafico
     {
          public int x;
          public int y;
          public int xInicial;
          public int yInicial;
          public int ancho;
          public int alto;
          public int incrX;
          public int incrY;
          public Imagen imagen;
          public bool visible;
     }

     static byte anchoFondo = 20;
     static byte altoFondo = 16;
     static short margenXFondo = 80;
```

```
static byte margenYFondo = 30;
static byte anchoCasillaFondo = 32;
static byte altoCasillaFondo = 32;
static Imagen imgPared;
static public byte[,] fondo =
{
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
};

static ElemGrafico personaje;
static Fuente tipoDeLetra;

static int numPremios, numEnemigos, numObstaculos;

static ElemGrafico[] obstaculos;
static ElemGrafico[] enemigos;
static ElemGrafico[] premios;

static bool juegoTerminado;
static int vidas;
static int puntos;
static bool partidaTerminada;
static Random generador;

static Imagen fondoPresentacion;
static Imagen fondoAyuda;
static Imagen fondoCreditos;


public static void InicializarJuego()
{
    // Entrar a modo grafico 800x600
    bool pantallaCompleta = false;
    Hardware.Inicializar(800, 600, 24, pantallaCompleta);

    // Resto de inicializacion
    tipoDeLetra = new Fuente("FreeSansBold.ttf", 18);
    juegoTerminado = false;
    numPremios = 10;
    numEnemigos = 10;
    numObstaculos = 20;
    obstaculos = new ElemGrafico[numObstaculos];
    enemigos = new ElemGrafico[numEnemigos];
    premios = new ElemGrafico[numPremios];
```

```
        generador = new Random();

        // Cargo imágenes de elementos
        personaje.imagen = new Imagen("personaje.png");

        for (int i = 0; i < numObstaculos; i++)  // Obstaculos
            obstaculos[i].imagen = new Imagen("obstaculo.png");

        for (int i = 0; i < numEnemigos; i++)  // Enemigos
            enemigos[i].imagen = new Imagen("enemigo.png");

        for (int i = 0; i < numPremios; i++)  // Premios
            premios[i].imagen = new Imagen("premio.png");

        imgPared = new Imagen("pared.png");

        // Y cargo las imagenes de la presentación, ayuda y créditos
        fondoPresentacion = new Imagen("present.jpg");
        fondoAyuda = new Imagen("present.jpg");
        fondoCreditos = new Imagen("present.jpg");
    }


    public static void InicializarPartida()
    {
        // En cada partida, hay que reiniciar ciertas variables
        vidas = 3;
        puntos = 0;
        partidaTerminada = false;

        personaje.xInicial = 400;
        personaje.yInicial = 300;
        personaje.x = personaje.xInicial;
        personaje.y = personaje.yInicial;
        personaje.visible = true;
        personaje.ancho = 32;
        personaje.alto = 30;
        personaje.incrX = 10;
        personaje.incrY = 10;

        // Genero las posiciones de los elementos al azar
        for (int i = 0; i < numObstaculos; i++)  // Obstaculos
        {
            obstaculos[i].visible = true;
            obstaculos[i].ancho = 38;
            obstaculos[i].alto = 22;
            // Al colocar un obstáculo, compruebo que no choque
            // con el personaje, para que la partida
            // no acabe nada más empezar
            do
            {
                obstaculos[i].x = generador.Next(50, 700);
                obstaculos[i].y = generador.Next(30, 550);
            } while (Colision(obstaculos[i], personaje));
        }

        for (int i = 0; i < numEnemigos; i++)  // Enemigos
        {
            enemigos[i].incrX = 5;
```

```
            enemigos[i].visible = true;
            enemigos[i].ancho = 36;
            enemigos[i].alto = 42;
            enemigos[i].x = generador.Next(50, 700);
            // Para la Y, compruebo que no sea del rango de
            // la del personaje, para que la partida
            // no acabe nada más empezar
            do
            {
                enemigos[i].y = generador.Next(30, 550);
            } while ((enemigos[i].y + enemigos[i].alto > personaje.y)
                && (enemigos[i].y < personaje.y + personaje.alto));
        }

        for (int i = 0; i < numPremios; i++)  // Premios
        {
            premios[i].x = generador.Next(50, 700);
            premios[i].y = generador.Next(30, 550);
            premios[i].visible = true;
            premios[i].ancho = 34;
            premios[i].alto = 18;
        }
    }


    public static void MostrarPresentacion()
    {
        bool finPresentacion = false;

        do
        {
            // ---- Pantalla de presentación --
            Hardware.BorrarPantallaOculta(0, 0, 0);

            // Fondo de la presentación
            fondoPresentacion.DibujarOculta(0, 0);

            // Marcador
            Hardware.EscribirTextoOculta("Jueguecillo",
                       340, 200, // Coordenadas
                       255, 255, 255, // Colores
                       tipoDeLetra);

            Hardware.EscribirTextoOculta("Escoja una opción:",
                       310, 300, // Coordenadas
                       200, 200, 200, // Colores
                       tipoDeLetra);

            Hardware.EscribirTextoOculta("J.- Jugar una partida",
                       150, 390, // Coordenadas
                       200, 200, 200, // Colores
                      tipoDeLetra);

            Hardware.EscribirTextoOculta("A.- Ayuda",
                       150, 430, // Coordenadas
                       200, 200, 200, // Colores
                       tipoDeLetra);

            Hardware.EscribirTextoOculta("C.- Créditos",
```

```
                    150, 470, // Coordenadas
                    200, 200, 200, // Colores
                    tipoDeLetra);

          Hardware.EscribirTextoOculta("S.- Salir",
                    150, 510, // Coordenadas
                    200, 200, 200, // Colores
                    tipoDeLetra);

          Hardware.VisualizarOculta();

          Hardware.Pausa(20);

          if (Hardware.TeclaPulsada(Hardware.TECLA_A))
               MostrarAyuda();

          if (Hardware.TeclaPulsada(Hardware.TECLA_C))
               MostrarCreditos();

          if (Hardware.TeclaPulsada(Hardware.TECLA_J))
               finPresentacion = true;

          if (Hardware.TeclaPulsada(Hardware.TECLA_S))
          {
               finPresentacion = true;
               partidaTerminada = true;
               juegoTerminado = true;
          }
     } while (!finPresentacion);
}


public static void MostrarAyuda()
{
     string[] textosAyuda =
     {
       "Recoge los premios",
       "Evita los obstáculos y los enemigos",
       "Usa las flechas de cursor para mover"
     };

     // ---- Pantalla de presentación --
     Hardware.BorrarPantallaOculta(0, 0, 0);

     // Fondo de la presentación
     fondoAyuda.DibujarOculta(0, 0);

     // Marcador
     Hardware.EscribirTextoOculta("Ayuda",
               340, 200, // Coordenadas
               255, 255, 255, // Colores
               tipoDeLetra);

     // Textos repetitivos
     short posicYtexto = 280;
     foreach (string texto in textosAyuda)
     {
          Hardware.EscribirTextoOculta(texto,
                    150, posicYtexto, // Coordenadas
```

```
                    200, 200, 200, // Colores
                   tipoDeLetra);
          posicYtexto += 30;
     }

     Hardware.EscribirTextoOculta("ESC- volver",
               650, 530, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);
     Hardware.VisualizarOculta();

     do
     {
          Hardware.Pausa(20);
     } while (!Hardware.TeclaPulsada(Hardware.TECLA_ESC));
}


public static void MostrarCreditos()
{
     // ---- Pantalla de presentación --
     Hardware.BorrarPantallaOculta(0, 0, 0);

     // Fondo de la presentación
     fondoCreditos.DibujarOculta(0, 0);

     // Marcador
     Hardware.EscribirTextoOculta("Creditos",
               250, 200, // Coordenadas
               255, 255, 255, // Colores
              tipoDeLetra);

     Hardware.EscribirTextoOculta("Por Nacho Cabanes, 2011",
                250, 300, // Coordenadas
                200, 200, 200, // Colores
               tipoDeLetra);

     Hardware.EscribirTextoOculta("ESC- volver",
               650, 530, // Coordenadas
               200, 200, 200, // Colores
              tipoDeLetra);
     Hardware.VisualizarOculta();

     do
     {
          Hardware.Pausa(20);
     } while (!Hardware.TeclaPulsada(Hardware.TECLA_ESC));
}


public static void Dibujar()
{
     // -- Dibujar --
     Hardware.BorrarPantallaOculta(0, 0, 0);

     // Marcador
     Hardware.EscribirTextoOculta("Vidas        Puntos",
               0, 0, // Coordenadas
               255, 255, 255, // Colores
```

```
                tipoDeLetra);

        Hardware.EscribirTextoOculta(Convert.ToString(vidas),
                 70, 0, // Coordenadas
                 200, 200, 200, // Colores
                tipoDeLetra);

        Hardware.EscribirTextoOculta(Convert.ToString(puntos),
                 190, 0, // Coordenadas
                 200, 200, 200, // Colores
                tipoDeLetra);

        // Pared de fondo
        for (int fila = 0; fila < altoFondo; fila++)  // Fondo
            for (int col = 0; col < anchoFondo; col++)
                if (fondo[fila, col] == 1)
                    imgPared.DibujarOculta(
                        margenXFondo + col * anchoCasillaFondo,
                        margenYFondo + fila * altoCasillaFondo);

        for (int i = 0; i < numObstaculos; i++)  // Obstáculos
        {
            obstaculos[i].imagen.DibujarOculta(
                (int)obstaculos[i].x, (int)obstaculos[i].y);
        }

        for (int i = 0; i < numEnemigos; i++)  // Enemigos
        {
            enemigos[i].imagen.DibujarOculta(
                (int)enemigos[i].x, (int)enemigos[i].y);
        }

        for (int i = 0; i < numPremios; i++)  // Premios
        {
            if (premios[i].visible)
            {
                premios[i].imagen.DibujarOculta(
                    premios[i].x, premios[i].y);
            }
        }

        personaje.imagen.DibujarOculta(
            personaje.x, personaje.y);

        // Finalmente, muestro en pantalla
        Hardware.VisualizarOculta();
    }


    public static void ComprobarTeclas()
    {
        // -- Leer teclas y calcular nueva posición --
        if (Hardware.TeclaPulsada(Hardware.TECLA_ESC))
            partidaTerminada = true;

        if (Hardware.TeclaPulsada(Hardware.TECLA_DER)
            && EsPosibleMover(personaje.x + personaje.incrX, personaje.y,
                    personaje.x + personaje.ancho + personaje.incrX, personaje.y + personaje.alto))
                personaje.x += personaje.incrX;
```

```
    if (Hardware.TeclaPulsada(Hardware.TECLA_IZQ)
        && EsPosibleMover(personaje.x - personaje.incrX, personaje.y,
                personaje.x + personaje.ancho - personaje.incrX, personaje.y + personaje.alto))
            personaje.x -= personaje.incrX;
    if (Hardware.TeclaPulsada(Hardware.TECLA_ARR)
        && EsPosibleMover(personaje.x, personaje.y - personaje.incrY,
                personaje.x + personaje.ancho, personaje.y + personaje.alto - personaje.incrY))
            personaje.y -= personaje.incrY;
    if (Hardware.TeclaPulsada(Hardware.TECLA_ABA)
        && EsPosibleMover(personaje.x, personaje.y + personaje.incrY,
                personaje.x + personaje.ancho, personaje.y + personaje.alto + personaje.incrY))
            personaje.y += personaje.incrY;
}


public static void MoverElementos()
{
    // -- Mover enemigos, entorno --
    for (int i = 0; i < numEnemigos; i++)  // Enemigos
    {
        enemigos[i].x = enemigos[i].x + enemigos[i].incrX;
        if (((int)enemigos[i].x <= 50)
                || ((int)enemigos[i].x >= 700))
            enemigos[i].incrX = -enemigos[i].incrX;
    }
}


public static void ComprobarColisiones()
{
    // -- Colisiones, perder vidas, etc --
    for (int i = 0; i < numObstaculos; i++)  // Obstáculos
    {
        if (Colision(obstaculos[i], personaje))
        {
            vidas--;
            if (vidas == 0)
                partidaTerminada = true;
            personaje.x = personaje.xInicial;
            personaje.y = personaje.yInicial;
        }
    }

    for (int i = 0; i < numPremios; i++)  // Premios
    {
        if (Colision(premios[i], personaje))
        {
            puntos += 10;
            premios[i].visible = false;
        }
    }

    for (int i = 0; i < numEnemigos; i++)  // Enemigos
    {
        if (Colision(enemigos[i], personaje))
        {
            vidas--;
            if (vidas == 0)
                partidaTerminada = true;
```

```
                personaje.x = personaje.xInicial;
                personaje.y = personaje.yInicial;
            }
        }

}


public static void PausaFotograma()
{
    // -- Pausa hasta el siguiente "fotograma" del juego --
    Hardware.Pausa(20);
}


public static bool Colision(ElemGrafico e1, ElemGrafico e2)
{
    // No se debe chocar con un elemento oculto
    if ((e1.visible == false) || (e2.visible == false))
        return false;
    // Ahora ya compruebo coordenadas
    if ((e1.x + e1.ancho > e2.x)
          && (e1.x < e2.x + e2.ancho)
          && (e1.y + e1.alto > e2.y)
          && (e1.y < e2.y + e2.alto))
        return true;
    else
        return false;
}


public static bool Colision(
    int x1, int y1, int xFin1, int yFin1,
    int x2, int y2, int xFin2, int yFin2)
{
    if ((xFin2 > x1)
          && (x2 < xFin1)
          && (yFin2 > y1)
          && (y2 < yFin1))
        return true;
    else
        return false;
}


public static bool EsPosibleMover(int x, int y, int xFin, int yFin)
{
    // Veo si choca con algún elemento del fondo
    for (int fila = 0; fila < altoFondo; fila++)  // Fondo
        for (int col = 0; col < anchoFondo; col++)
            if ((fondo[fila, col] == 1) &&  Colision(
                     x,y,xFin,yFin,  // Posicion del personaje
                     margenXFondo + col * anchoCasillaFondo,  // x inicial de casilla actual de fondo
                     margenYFondo + fila * altoCasillaFondo,  // y inicial
                     margenXFondo + (col+1) * anchoCasillaFondo,  // x final
                     margenYFondo + (fila+1) * altoCasillaFondo))  // y final
                return false;

    // Si no ha chocado con ninguno, es posible moverse a esa posición
```

```
            return true;
        }



        public static void Main()
        {

            InicializarJuego();

            while (!juegoTerminado)
            {
                InicializarPartida();
                MostrarPresentacion();

                // ------ Bucle de juego ------
                while (!partidaTerminada)
                {
                    Dibujar();
                    ComprobarTeclas();
                    MoverElementos();
                    ComprobarColisiones();
                    PausaFotograma();
                } // Fin del bucle de juego
            }     // Fin de partida
        }         // Fin de Main
}
```

Este planteamiento permite gastar muy poca memoria, porque la imagen repetitiva del fondo se carga sólo una vez. A cambio, complica otras posibilidades, como la de incluir los premios y obstáculos mortales como parte del "mapa" del fondo. Por eso, también podemos usar un planteamiento alternativo: a partir del "mapa" del fondo, crear un array de elementos gráficos, y comprobar colisiones directamente con todos esos elementos gráficos.

Si lo planteamos así, la función "ComprobarTeclas" no cambiará, pero sí lo hará "EsPosibleMover", que comprobará colisiones y también ampliaremos "InicializarJuego" para preparar ese nuevo array de elementos gráficos.

Vamos a empezar por ver los cambios en InicializarJuego: crearíamos un nuevo array, que ni siquiera hace falta que sea tan grande como el fondo, sino que puede estar formado por las casillas que no estén vacías:

```
// Cuento la cantidad de elementos reales en el fondo
elementosFondo = 0;
for (int fila = 0; fila < altoFondo; fila++)  // Fondo
    for (int col = 0; col < anchoFondo; col++)
        if (fondo[fila, col] != 0)
            elementosFondo++;
// Y reservo espacio para el array que los contiene
fondos = new ElemGrafico[elementosFondo];
// y para cada uno de uno de ellos
int posicFondo = 0;
for (int fila = 0; fila < altoFondo; fila++)  // Fondo
    for (int col = 0; col < anchoFondo; col++)
        if (fondo[fila, col] != 0)
        {
            fondos[posicFondo].x =  margenXFondo + col * anchoCasillaFondo;
            fondos[posicFondo].y =  margenYFondo + fila * altoCasillaFondo;
            fondos[posicFondo].imagen = new Imagen("pared.png");
            fondos[posicFondo].ancho = anchoCasillaFondo;
            fondos[posicFondo].alto = altoCasillaFondo;
            fondos[posicFondo].visible =  true;
```

```
            posicFondo++;
        }
```

En vez de crear una nueva función "Colision" alternativa, podemos hacer que "EsPosibleMover" sea la que compruebe si los rectángulos de esos nuevos elementos gráficos se solapan con las coordenadas que le indiquemos:

```
public static bool EsPosibleMover(int x, int y, int xFin, int yFin)
{
    // Veo si choca con algún elemento del fondo
    for (int i = 0; i < elementosFondo; i++)
        if ((fondos[i].x + fondos[i].ancho > x)
          && (fondos[i].x < xFin)
        && (fondos[i].y + fondos[i].alto > y)
          && (fondos[i].y < yFin))
        return false;

    // Si no ha chocado con ninguno, es posible moverse a esa posición
    return true;
}
```

Y ya que estamos, como tenemos un array de elementos gráficos, podemos simplificar la parte de "Dibujar" que se encarga de mostrar el fondo:

```
for (int i = 0; i < elementosFondo; i++)  // Imágenes del fondo
  fondos[i].imagen.DibujarOculta(fondos[i].x, fondos[i].y);
```

Con estos cambios, esta versión alternativa del fuente sería:

```
// Primer mini-esqueleto de juego en modo gráfico
// Versión "h"

using System;
using System.Threading; // Para Thread.Sleep

public class Juego05h
{
    public struct ElemGrafico
    {
        public int x;
        public int y;
        public int xInicial;
        public int yInicial;
        public int ancho;
        public int alto;
        public int incrX;
        public int incrY;
        public Imagen imagen;
        public bool visible;
    }

    static byte anchoFondo = 20;
    static byte altoFondo = 16;
    static short margenXFondo = 80;
    static byte margenYFondo = 30;
    static byte anchoCasillaFondo = 32;
    static byte altoCasillaFondo = 32;
    static Imagen imgPared;
    static public byte[,] fondo =
    {
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
```

```
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
    };

    static ElemGrafico personaje;
    static Fuente tipoDeLetra;

    static int numPremios, numEnemigos, numObstaculos;

    static ElemGrafico[] obstaculos;
    static ElemGrafico[] enemigos;
    static ElemGrafico[] premios;
    static ElemGrafico[] fondos;
    static int elementosFondo;

    static bool juegoTerminado;
    static int vidas;
    static int puntos;
    static bool partidaTerminada;
    static Random generador;

    static Imagen fondoPresentacion;
    static Imagen fondoAyuda;
    static Imagen fondoCreditos;


    public static void InicializarJuego()
    {
        // Entrar a modo grafico 800x600
        bool pantallaCompleta = false;
        Hardware.Inicializar(800, 600, 24, pantallaCompleta);

        // Resto de inicializacion
        tipoDeLetra = new Fuente("FreeSansBold.ttf", 18);
        juegoTerminado = false;
        numPremios = 10;
        numEnemigos = 10;
        numObstaculos = 20;
        obstaculos = new ElemGrafico[numObstaculos];
        enemigos = new ElemGrafico[numEnemigos];
        premios = new ElemGrafico[numPremios];
        generador = new Random();

        // Cuento la cantidad de elementos reales en el fondo
        elementosFondo = 0;
        for (int fila = 0; fila < altoFondo; fila++)  // Fondo
```

```
            for (int col = 0; col < anchoFondo; col++)
                if (fondo[fila, col] != 0)
                    elementosFondo++;
        // Y reservo espacio para el array que los contiene
        fondos = new ElemGrafico[elementosFondo];
        // y para cada uno de uno de ellos
        int posicFondo = 0;
        for (int fila = 0; fila < altoFondo; fila++)  // Fondo
            for (int col = 0; col < anchoFondo; col++)
                if (fondo[fila, col] != 0)
                {
                    fondos[posicFondo].x =  margenXFondo + col * anchoCasillaFondo;
                    fondos[posicFondo].y =  margenYFondo + fila * altoCasillaFondo;
                    fondos[posicFondo].imagen = new Imagen("pared.png");
                    fondos[posicFondo].ancho = anchoCasillaFondo;
                    fondos[posicFondo].alto = altoCasillaFondo;
                    fondos[posicFondo].visible =  true;
                    posicFondo++;
                }

        // Cargo imágenes de elementos
        personaje.imagen = new Imagen("personaje.png");

        for (int i = 0; i < numObstaculos; i++)  // Obstaculos
            obstaculos[i].imagen = new Imagen("obstaculo.png");

        for (int i = 0; i < numEnemigos; i++)  // Enemigos
            enemigos[i].imagen = new Imagen("enemigo.png");

        for (int i = 0; i < numPremios; i++)  // Premios
            premios[i].imagen = new Imagen("premio.png");

        imgPared = new Imagen("pared.png");

        // Y cargo las imagenes de la presentación, ayuda y créditos
        fondoPresentacion = new Imagen("present.jpg");
        fondoAyuda = new Imagen("present.jpg");
        fondoCreditos = new Imagen("present.jpg");
    }


    public static void InicializarPartida()
    {
        // En cada partida, hay que reiniciar ciertas variables
        vidas = 3;
        puntos = 0;
        partidaTerminada = false;

        personaje.xInicial = 400;
        personaje.yInicial = 300;
        personaje.x = personaje.xInicial;
        personaje.y = personaje.yInicial;
        personaje.visible = true;
        personaje.ancho = 32;
        personaje.alto = 30;
        personaje.incrX = 10;
        personaje.incrY = 10;

        // Genero las posiciones de los elementos al azar
```

```
    for (int i = 0; i < numObstaculos; i++)  // Obstaculos
    {
        obstaculos[i].visible = true;
        obstaculos[i].ancho = 38;
        obstaculos[i].alto = 22;
        // Al colocar un obstáculo, compruebo que no choque
        // con el personaje, para que la partida
        // no acabe nada más empezar
        do
        {
            obstaculos[i].x = generador.Next(50, 700);
            obstaculos[i].y = generador.Next(30, 550);
        } while (Colision(obstaculos[i], personaje));
    }

    for (int i = 0; i < numEnemigos; i++)  // Enemigos
    {
        enemigos[i].incrX = 5;
        enemigos[i].visible = true;
        enemigos[i].ancho = 36;
        enemigos[i].alto = 42;
        enemigos[i].x = generador.Next(50, 700);
        // Para la Y, compruebo que no sea del rango de
        // la del personaje, para que la partida
        // no acabe nada más empezar
        do
        {
            enemigos[i].y = generador.Next(30, 550);
        } while ((enemigos[i].y + enemigos[i].alto > personaje.y)
            && (enemigos[i].y < personaje.y + personaje.alto));
    }

    for (int i = 0; i < numPremios; i++)  // Premios
    {
        premios[i].x = generador.Next(50, 700);
        premios[i].y = generador.Next(30, 550);
        premios[i].visible = true;
        premios[i].ancho = 34;
        premios[i].alto = 18;
    }
}


public static void MostrarPresentacion()
{
    bool finPresentacion = false;

    do
    {
        // ---- Pantalla de presentación --
        Hardware.BorrarPantallaOculta(0, 0, 0);

        // Fondo de la presentación
        fondoPresentacion.DibujarOculta(0, 0);

        // Marcador
        Hardware.EscribirTextoOculta("Jueguecillo",
                  340, 200, // Coordenadas
                  255, 255, 255, // Colores
```

```
                    tipoDeLetra);

          Hardware.EscribirTextoOculta("Escoja una opción:",
                    310, 300, // Coordenadas
                    200, 200, 200, // Colores
                    tipoDeLetra);

          Hardware.EscribirTextoOculta("J.- Jugar una partida",
                    150, 390, // Coordenadas
                    200, 200, 200, // Colores
                   tipoDeLetra);

          Hardware.EscribirTextoOculta("A.- Ayuda",
                    150, 430, // Coordenadas
                    200, 200, 200, // Colores
                    tipoDeLetra);

          Hardware.EscribirTextoOculta("C.- Créditos",
                    150, 470, // Coordenadas
                    200, 200, 200, // Colores
                    tipoDeLetra);

          Hardware.EscribirTextoOculta("S.- Salir",
                    150, 510, // Coordenadas
                    200, 200, 200, // Colores
                    tipoDeLetra);

          Hardware.VisualizarOculta();

          Hardware.Pausa(20);

          if (Hardware.TeclaPulsada(Hardware.TECLA_A))
               MostrarAyuda();

          if (Hardware.TeclaPulsada(Hardware.TECLA_C))
               MostrarCreditos();

          if (Hardware.TeclaPulsada(Hardware.TECLA_J))
               finPresentacion = true;

          if (Hardware.TeclaPulsada(Hardware.TECLA_S))
          {
               finPresentacion = true;
               partidaTerminada = true;
               juegoTerminado = true;
          }
     } while (!finPresentacion);
}


public static void MostrarAyuda()
{
     string[] textosAyuda =
     {
       "Recoge los premios",
       "Evita los obstáculos y los enemigos",
       "Usa las flechas de cursor para mover"
     };
```

```
        // ---- Pantalla de presentación --
        Hardware.BorrarPantallaOculta(0, 0, 0);

        // Fondo de la presentación
        fondoAyuda.DibujarOculta(0, 0);

        // Marcador
        Hardware.EscribirTextoOculta("Ayuda",
                  340, 200, // Coordenadas
                  255, 255, 255, // Colores
                 tipoDeLetra);

        // Textos repetitivos
        short posicYtexto = 280;
        foreach (string texto in textosAyuda)
        {
            Hardware.EscribirTextoOculta(texto,
                       150, posicYtexto, // Coordenadas
                       200, 200, 200, // Colores
                      tipoDeLetra);
            posicYtexto += 30;
        }

        Hardware.EscribirTextoOculta("ESC- volver",
                  650, 530, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);
        Hardware.VisualizarOculta();

        do
        {
            Hardware.Pausa(20);
        } while (!Hardware.TeclaPulsada(Hardware.TECLA_ESC));
    }


    public static void MostrarCreditos()
    {
        // ---- Pantalla de presentación --
        Hardware.BorrarPantallaOculta(0, 0, 0);

        // Fondo de la presentación
        fondoCreditos.DibujarOculta(0, 0);

        // Marcador
        Hardware.EscribirTextoOculta("Creditos",
                  250, 200, // Coordenadas
                  255, 255, 255, // Colores
                 tipoDeLetra);

        Hardware.EscribirTextoOculta("Por Nacho Cabanes, 2011",
                   250, 300, // Coordenadas
                   200, 200, 200, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("ESC- volver",
                  650, 530, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);
```

```
        Hardware.VisualizarOculta();

        do
        {
            Hardware.Pausa(20);
        } while (!Hardware.TeclaPulsada(Hardware.TECLA_ESC));
    }


    public static void Dibujar()
    {
        // -- Dibujar --
        Hardware.BorrarPantallaOculta(0, 0, 0);

        // Marcador
        Hardware.EscribirTextoOculta("Vidas        Puntos",
                  0, 0, // Coordenadas
                  255, 255, 255, // Colores
                 tipoDeLetra);

        Hardware.EscribirTextoOculta(Convert.ToString(vidas),
                  70, 0, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);

        Hardware.EscribirTextoOculta(Convert.ToString(puntos),
                  190, 0, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);

        for (int i = 0; i < elementosFondo; i++)  // Imágenes del fondo
            fondos[i].imagen.DibujarOculta(fondos[i].x, fondos[i].y);

        for (int i = 0; i < numObstaculos; i++)  // Obstáculos
            obstaculos[i].imagen.DibujarOculta(obstaculos[i].x, obstaculos[i].y);

        for (int i = 0; i < numEnemigos; i++)  // Enemigos
            enemigos[i].imagen.DibujarOculta(enemigos[i].x, enemigos[i].y);

        for (int i = 0; i < numPremios; i++)  // Premios
            if (premios[i].visible)
                premios[i].imagen.DibujarOculta(premios[i].x, premios[i].y);

        personaje.imagen.DibujarOculta(
            personaje.x, personaje.y);

        // Finalmente, muestro en pantalla
        Hardware.VisualizarOculta();
    }


    public static void ComprobarTeclas()
    {
        // -- Leer teclas y calcular nueva posición --
        if (Hardware.TeclaPulsada(Hardware.TECLA_ESC))
            partidaTerminada = true;

        if (Hardware.TeclaPulsada(Hardware.TECLA_DER)
            && EsPosibleMover(personaje.x + personaje.incrX, personaje.y,
```

```
                personaje.x + personaje.ancho + personaje.incrX, personaje.y + personaje.alto))
            personaje.x += personaje.incrX;
    if (Hardware.TeclaPulsada(Hardware.TECLA_IZQ)
        && EsPosibleMover(personaje.x - personaje.incrX, personaje.y,
                personaje.x + personaje.ancho - personaje.incrX, personaje.y + personaje.alto))
            personaje.x -= personaje.incrX;
    if (Hardware.TeclaPulsada(Hardware.TECLA_ARR)
        && EsPosibleMover(personaje.x, personaje.y - personaje.incrY,
                personaje.x + personaje.ancho, personaje.y + personaje.alto - personaje.incrY))
            personaje.y -= personaje.incrY;
    if (Hardware.TeclaPulsada(Hardware.TECLA_ABA)
        && EsPosibleMover(personaje.x, personaje.y + personaje.incrY,
                personaje.x + personaje.ancho, personaje.y + personaje.alto + personaje.incrY))
            personaje.y += personaje.incrY;
}


public static void MoverElementos()
{
    // -- Mover enemigos, entorno --
    for (int i = 0; i < numEnemigos; i++)  // Enemigos
    {
        enemigos[i].x = enemigos[i].x + enemigos[i].incrX;
        if (((int)enemigos[i].x <= 50)
                || ((int)enemigos[i].x >= 700))
            enemigos[i].incrX = -enemigos[i].incrX;
    }
}


public static void ComprobarColisiones()
{
    // -- Colisiones, perder vidas, etc --
    for (int i = 0; i < numObstaculos; i++)  // Obstáculos
    {
        if (Colision(obstaculos[i], personaje))
        {
            vidas--;
            if (vidas == 0)
                partidaTerminada = true;
            personaje.x = personaje.xInicial;
            personaje.y = personaje.yInicial;
        }
    }

    for (int i = 0; i < numPremios; i++)  // Premios
    {
        if (Colision(premios[i], personaje))
        {
            puntos += 10;
            premios[i].visible = false;
        }
    }

    for (int i = 0; i < numEnemigos; i++)  // Enemigos
    {
        if (Colision(enemigos[i], personaje))
        {
            vidas--;
```

```
            if (vidas == 0)
                partidaTerminada = true;
            personaje.x = personaje.xInicial;
            personaje.y = personaje.yInicial;
        }
    }

}


public static void PausaFotograma()
{
    // -- Pausa hasta el siguiente "fotograma" del juego --
    Hardware.Pausa(20);
}


public static bool Colision(ElemGrafico e1, ElemGrafico e2)
{
    // No se debe chocar con un elemento oculto
    if ((e1.visible == false) || (e2.visible == false))
        return false;
    // Ahora ya compruebo coordenadas
    if ((e1.x + e1.ancho > e2.x)
          && (e1.x < e2.x + e2.ancho)
          && (e1.y + e1.alto > e2.y)
          && (e1.y < e2.y + e2.alto))
        return true;
    else
        return false;
}


public static bool EsPosibleMover(int x, int y, int xFin, int yFin)
{
    // Veo si choca con algún elemento del fondo
    for (int i = 0; i < elementosFondo; i++)
        if ((fondos[i].x + fondos[i].ancho > x)
          && (fondos[i].x < xFin)
        && (fondos[i].y + fondos[i].alto > y)
          && (fondos[i].y < yFin))
        return false;

    // Si no ha chocado con ninguno, es posible moverse a esa posición
    return true;
}


public static void Main()
{

    InicializarJuego();

    while (!juegoTerminado)
    {
        InicializarPartida();
        MostrarPresentacion();

        // ------ Bucle de juego ------
```

```
            while (!partidaTerminada)
            {
                Dibujar();
                ComprobarTeclas();
                MoverElementos();
                ComprobarColisiones();
                PausaFotograma();
            } // Fin del bucle de juego
        }     // Fin de partida
    }         // Fin de Main
}
```

Ejercicio propuesto (1): Mejora esta versión del juego, para que el fondo no esté formado por un único tipo de casilla, sino por dos o más.

Ejercicio propuesto (2): Haz que los premios que se pueden recoger no aparezcan en posiciones al azar, sino que sean parte también del mapa del nivel, además de los ladrillos del fondo. (Si no te sale, no te preocupes, pronto veremos cómo hacerlo)

30. Un primer juego completo en modo gráfico.

Tenemos ya un esqueleto de juego que hace un poco de todo, pero que todavía no es muy jugable: es "demasiado difícil" y "demasiado repetitivo", además de que tiene otras muchas carencias que de momento no nos van a preocupar:

- Sólo hay una "pantalla" que recorrer.
- Sólo hay un tipo de enemigos, un tipo de obstáculos, un tipo de premios...
- No hay ningún movimiento "avanzado" (no podemos saltar, ni caer de las plataformas).
- No hay animaciones en los movimientos.
- No tiene sonido.
- No se puede jugar con joystick ni gamepad.
- ...

Vamos a ignorar esas carencias por ahora, porque las iremos solucionando más adelante, y a juntar todo lo que hemos visto, para crear un juego sencillo pero "real", con un nivel de dificultad creciente. El funcionamiento será el siguiente:

- Al comenzar el juego, se nos mostrará una pantalla de presentación, que nos permitirá ver una pequeña ayuda, una pantalla de créditos o comenzar una nueva partida.
- En la partida, nuestro personaje aparecerá en una primera pantalla, en la que habrá paredes que no podremos atravesar, un premio que podremos recoger, pero no habrá ningún obstáculo ni ningún enemigo.
- Cuando recojamos ese premio, pasaremos a un segundo nivel, que tendrá las mismas paredes, pero dos premios en (nuevas) posiciones al azar, un obstáculo fijo y un enemigo móvil.
- Cuando recojamos esos dos premios, pasaremos a un tercer nivel, también con las mismas paredes, pero tres premios, dos obstáculos fijo y dos enemigos móviles.
- En general, cuando recojamos todos los premios de un nivel, pasaremos a un siguiente nivel, que tendrá un premio más que el anterior (hasta un máximo de 20), un enemigo móvil más que el anterior (hasta un máximo de 15) y un obstáculo fijo más que el anterior (hasta un máximo de 20).
- Obtendremos 10 puntos por cada premio recogido.
- Si tocamos un obstáculo o un enemigo, perderemos una vida. Cuando perdamos las tres vidas iniciales, terminará la partida y volveremos a la pantalla de presentación.

El fuente completo podría ser algo como:

```
// Primer mini-juego en modo gráfico: Freddy One

using System;
using System.Threading; // Para Thread.Sleep

public class FreddyOne
{
    public struct ElemGrafico
    {
        public int x;
        public int y;
        public int xInicial;
        public int yInicial;
        public int ancho;
        public int alto;
```

```
    public int incrX;
    public int incrY;
    public Imagen imagen;
    public bool visible;
}

static byte anchoFondo = 24;
static byte altoFondo = 17;
static short margenXFondo = 10;
static byte margenYFondo = 30;
static byte anchoCasillaFondo = 32;
static byte altoCasillaFondo = 32;
static Imagen imgPared;
static public byte[,] fondo =
{
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
};

static ElemGrafico personaje;
static Fuente tipoDeLetra;

static int numPremios, numEnemigos, numObstaculos;

static ElemGrafico[] obstaculos;
static ElemGrafico[] enemigos;
static ElemGrafico[] premios;
static ElemGrafico[] fondos;
static int elementosFondo;

static bool juegoTerminado;
static int vidas;
static int puntos;
static bool partidaTerminada;
static Random generador;

static Imagen fondoPresentacion;
static Imagen fondoAyuda;
static Imagen fondoCreditos;
static Imagen logoPresentacion;

// Cantidad de elementos que se ven en pantalla en el nivel actual
static int premiosVisibles;
static int enemigosVisibles;
```

```
static int obstaculosVisibles;

// Cantidad de premios que quedan para cambiar de nivel
static int premiosRestantes;


public static void InicializarJuego()
{
     // Entrar a modo grafico 800x600
     bool pantallaCompleta = false;
     Hardware.Inicializar(800, 600, 24, pantallaCompleta);

     // Resto de inicializacion
     tipoDeLetra = new Fuente("FreeSansBold.ttf", 18);
     juegoTerminado = false;
     numPremios = 20;
     numEnemigos = 12;
     numObstaculos = 20;
     obstaculos = new ElemGrafico[numObstaculos];
     enemigos = new ElemGrafico[numEnemigos];
     premios = new ElemGrafico[numPremios];
     generador = new Random();

     // Cuento la cantidad de elementos reales en el fondo
     elementosFondo = 0;
     for (int fila = 0; fila < altoFondo; fila++)  // Fondo
          for (int col = 0; col < anchoFondo; col++)
               if (fondo[fila, col] != 0)
                    elementosFondo++;
     // Y reservo espacio para el array que los contiene
     fondos = new ElemGrafico[elementosFondo];
     // y para cada uno de uno de ellos
     int posicFondo = 0;
     for (int fila = 0; fila < altoFondo; fila++)  // Fondo
          for (int col = 0; col < anchoFondo; col++)
               if (fondo[fila, col] != 0)
               {
                    fondos[posicFondo].x =  margenXFondo + col * anchoCasillaFondo;
                    fondos[posicFondo].y =  margenYFondo + fila * altoCasillaFondo;
                    fondos[posicFondo].imagen = new Imagen("pared.png");
                    fondos[posicFondo].ancho = anchoCasillaFondo;
                    fondos[posicFondo].alto = altoCasillaFondo;
                    fondos[posicFondo].visible =  true;
                    posicFondo++;
               }

     // Cargo imágenes de elementos
     personaje.imagen = new Imagen("personaje.png");

     for (int i = 0; i < numObstaculos; i++)  // Obstaculos
          obstaculos[i].imagen = new Imagen("obstaculo.png");

     for (int i = 0; i < numEnemigos; i++)  // Enemigos
          enemigos[i].imagen = new Imagen("enemigo.png");

     for (int i = 0; i < numPremios; i++)  // Premios
          premios[i].imagen = new Imagen("premio.png");

     imgPared = new Imagen("pared.png");
```

```
    // Y cargo las imagenes de la presentación, ayuda y créditos
    fondoPresentacion = new Imagen("present.jpg");
    logoPresentacion = new Imagen("freddyRotulo.png");
    fondoAyuda = new Imagen("ayuda.jpg");
    fondoCreditos = new Imagen("creditos.jpg");
}


public static void InicializarPartida()
{
    // En cada partida, hay que reiniciar ciertas variables
    vidas = 3;
    puntos = 0;
    partidaTerminada = false;

    personaje.xInicial = 400;
    personaje.yInicial = 300;
    personaje.x = personaje.xInicial;
    personaje.y = personaje.yInicial;
    personaje.visible = true;
    personaje.ancho = 32;
    personaje.alto = 30;
    personaje.incrX = 10;
    personaje.incrY = 10;

    // Genero las posiciones de los elementos al azar
    for (int i = 0; i < numObstaculos; i++)  // Obstaculos
    {
        obstaculos[i].visible = false;
        obstaculos[i].ancho = 38;
        obstaculos[i].alto = 22;
        // Al colocar un obstáculo, compruebo que no choque
        // con el personaje, para que la partida
        // no acabe nada más empezar
        do
        {
            obstaculos[i].x = generador.Next(50, 700);
            obstaculos[i].y = generador.Next(30, 550);
        } while (Colision(obstaculos[i], personaje));
    }

    for (int i = 0; i < numEnemigos; i++)  // Enemigos
    {
        enemigos[i].incrX = 5;
        enemigos[i].visible = false;
        enemigos[i].ancho = 36;
        enemigos[i].alto = 42;
        enemigos[i].x = generador.Next(50, 700);
        // Para la Y, compruebo que no sea del rango de
        // la del personaje, para que la partida
        // no acabe nada más empezar
        do
        {
            enemigos[i].y = generador.Next(30, 550);
        } while ((enemigos[i].y + enemigos[i].alto > personaje.y)
            && (enemigos[i].y < personaje.y + personaje.alto));
    }
```

```
    for (int i = 0; i < numPremios; i++)  // Premios
        do
        {
            premios[i].x = generador.Next(50, 700);
            premios[i].y = generador.Next(30, 550);
            premios[i].visible = false;
            premios[i].ancho = 34;
            premios[i].alto = 18;
        }
        while (! EsPosibleMover(premios[i].x, premios[i].y,
            premios[i].x+premios[i].ancho, premios[i].y+premios[i].alto));

    premiosVisibles = 1;
    premios[0].visible = true;
    enemigosVisibles = 0;
    obstaculosVisibles = 0;
    premiosRestantes = 1;
}


public static void MostrarPresentacion()
{
    bool finPresentacion = false;

    do
    {
        // ---- Pantalla de presentación --
        Hardware.BorrarPantallaOculta(0, 0, 0);

        // Fondo de la presentación
        fondoPresentacion.DibujarOculta(0, 0);

        logoPresentacion.DibujarOculta(60, 100);

        Hardware.EscribirTextoOculta("Escoja una opción:",
                  310, 300, // Coordenadas
                  200, 200, 200, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("J.- Jugar una partida",
                  250, 390, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);

        Hardware.EscribirTextoOculta("A.- Ayuda",
                  250, 430, // Coordenadas
                  200, 200, 200, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("C.- Créditos",
                  250, 470, // Coordenadas
                  200, 200, 200, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("S.- Salir",
                  250, 510, // Coordenadas
                  200, 200, 200, // Colores
                  tipoDeLetra);
```

```
            Hardware.VisualizarOculta();

            Hardware.Pausa(20);

            if (Hardware.TeclaPulsada(Hardware.TECLA_A))
                MostrarAyuda();

            if (Hardware.TeclaPulsada(Hardware.TECLA_C))
                MostrarCreditos();

            if (Hardware.TeclaPulsada(Hardware.TECLA_J))
                finPresentacion = true;

            if (Hardware.TeclaPulsada(Hardware.TECLA_S))
            {
                finPresentacion = true;
                partidaTerminada = true;
                juegoTerminado = true;
            }
        } while (!finPresentacion);
    }


    public static void MostrarAyuda()
    {
        string[] textosAyuda =
        {
          "Recoge los premios",
          "Evita los obstáculos y los enemigos",
          "Usa las flechas de cursor para mover",
          "Cuando recojas todos los premios",
          "  avanzarás de nivel.",
          "Comienzas con 3 vidas.",
          "Si tocas un obstáculo o un enemigo,.",
          "  perderás una de ellas."
        };

        // ---- Pantalla de presentación --
        Hardware.BorrarPantallaOculta(0, 0, 0);

        // Fondo de la presentación
        fondoAyuda.DibujarOculta(0, 0);

        // Marcador
        Hardware.EscribirTextoOculta("Freddy One - Ayuda",
                  300, 100, // Coordenadas
                  255, 255, 255, // Colores
                 tipoDeLetra);

        // Textos repetitivos
        short posicYtexto = 250;
        foreach (string texto in textosAyuda)
        {
            Hardware.EscribirTextoOculta(texto,
                       150, posicYtexto, // Coordenadas
                       200, 200, 200, // Colores
                      tipoDeLetra);
            posicYtexto += 30;
        }
```

```
        Hardware.EscribirTextoOculta("ESC- volver",
                  650, 530, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);

        // También muestro imágenes aclaradoras
        premios[0].imagen.DibujarOculta(350, 252);
        obstaculos[0].imagen.DibujarOculta(510, 280);
        enemigos[0].imagen.DibujarOculta(560, 260);

        Hardware.VisualizarOculta();

        do
        {
            Hardware.Pausa(20);
        } while (!Hardware.TeclaPulsada(Hardware.TECLA_ESC));
    }


    public static void MostrarCreditos()
    {
        // ---- Pantalla de presentación --
        Hardware.BorrarPantallaOculta(0, 0, 0);

        // Fondo de la presentación
        fondoCreditos.DibujarOculta(0, 0);

        // Marcador
        Hardware.EscribirTextoOculta("Freddy One - Créditos",
                  300, 100, // Coordenadas
                  255, 255, 255, // Colores
                 tipoDeLetra);

        Hardware.EscribirTextoOculta("Por Nacho Cabanes, 2011",
                   150, 200, // Coordenadas
                   200, 200, 200, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("(Curso 2011-2012, primer juego de ejemplo)",
                   150, 250, // Coordenadas
                   200, 200, 200, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("Imágenes \"in-game\", de Electro Freddy, (c) SoftSpot 1984",
                   150, 300, // Coordenadas
                   200, 200, 200, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("Resto de imágenes por Nacho Cabanes",
                   150, 350, // Coordenadas
                   200, 200, 200, // Colores
                  tipoDeLetra);

        Hardware.EscribirTextoOculta("Se puede distribuir libremente",
                   150, 400, // Coordenadas
                   200, 200, 200, // Colores
                  tipoDeLetra);
```

```
        Hardware.EscribirTextoOculta("ESC- volver",
                  650, 530, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);
        Hardware.VisualizarOculta();

        do
        {
            Hardware.Pausa(20);
        } while (!Hardware.TeclaPulsada(Hardware.TECLA_ESC));
    }


    public static void Dibujar()
    {
        // -- Dibujar --
        Hardware.BorrarPantallaOculta(0, 0, 0);

        // Marcador
        Hardware.EscribirTextoOculta("Vidas        Puntos",
                  0, 0, // Coordenadas
                  255, 255, 255, // Colores
                 tipoDeLetra);

        Hardware.EscribirTextoOculta(Convert.ToString(vidas),
                  70, 0, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);

        Hardware.EscribirTextoOculta(Convert.ToString(puntos),
                  190, 0, // Coordenadas
                  200, 200, 200, // Colores
                 tipoDeLetra);

        for (int i = 0; i < elementosFondo; i++)  // Imágenes del fondo
            fondos[i].imagen.DibujarOculta(fondos[i].x, fondos[i].y);

        for (int i = 0; i < numObstaculos; i++)  // Obstáculos
            if (obstaculos[i].visible)
                obstaculos[i].imagen.DibujarOculta(obstaculos[i].x, obstaculos[i].y);

        for (int i = 0; i < numEnemigos; i++)  // Enemigos
            if (enemigos[i].visible)
                enemigos[i].imagen.DibujarOculta(enemigos[i].x, enemigos[i].y);

        for (int i = 0; i < numPremios; i++)  // Premios
            if (premios[i].visible)
                premios[i].imagen.DibujarOculta(premios[i].x, premios[i].y);

        personaje.imagen.DibujarOculta(
            personaje.x, personaje.y);

        // Finalmente, muestro en pantalla
        Hardware.VisualizarOculta();
    }


    public static void ComprobarTeclas()
    {
```

```
    // -- Leer teclas y calcular nueva posición --
    if (Hardware.TeclaPulsada(Hardware.TECLA_ESC))
        partidaTerminada = true;

    if (Hardware.TeclaPulsada(Hardware.TECLA_DER)
        && EsPosibleMover(personaje.x + personaje.incrX,
               personaje.y,
               personaje.x + personaje.ancho + personaje.incrX,
               personaje.y + personaje.alto))
            personaje.x += personaje.incrX;

    if (Hardware.TeclaPulsada(Hardware.TECLA_IZQ)
        && EsPosibleMover(personaje.x - personaje.incrX, personaje.y,
                  personaje.x + personaje.ancho - personaje.incrX, personaje.y + personaje.alto))
            personaje.x -= personaje.incrX;
    if (Hardware.TeclaPulsada(Hardware.TECLA_ARR)
        && EsPosibleMover(personaje.x, personaje.y - personaje.incrY,
                  personaje.x + personaje.ancho, personaje.y + personaje.alto - personaje.incrY))
            personaje.y -= personaje.incrY;
    if (Hardware.TeclaPulsada(Hardware.TECLA_ABA)
        && EsPosibleMover(personaje.x, personaje.y + personaje.incrY,
                  personaje.x + personaje.ancho, personaje.y + personaje.alto + personaje.incrY))
            personaje.y += personaje.incrY;
}


public static void MoverElementos()
{
    // -- Mover enemigos, entorno --
    for (int i = 0; i < numEnemigos; i++)  // Enemigos
    {
        enemigos[i].x = enemigos[i].x + enemigos[i].incrX;
        if (((int)enemigos[i].x <= 50)
                || ((int)enemigos[i].x >= 700))
            enemigos[i].incrX = -enemigos[i].incrX;
    }
}


public static void ComprobarColisiones()
{
    // -- Colisiones, perder vidas, etc --
    for (int i = 0; i < numObstaculos; i++)  // Obstáculos
    {
        if (Colision(obstaculos[i], personaje))
        {
            vidas--;
            if (vidas == 0)
                partidaTerminada = true;
            personaje.x = personaje.xInicial;
            personaje.y = personaje.yInicial;
        }
    }

    for (int i = 0; i < numPremios; i++)  // Premios
    {
        if (Colision(premios[i], personaje))
        {
            puntos += 10;
```

```
                premios[i].visible = false;
                premiosRestantes--;
                if (premiosRestantes == 0)
                    AvanzarNivel();
            }
        }

        for (int i = 0; i < numEnemigos; i++)  // Enemigos
        {
            if (Colision(enemigos[i], personaje))
            {
                vidas--;
                if (vidas == 0)
                    partidaTerminada = true;
                personaje.x = personaje.xInicial;
                personaje.y = personaje.yInicial;
            }
        }

    }


    public static void PausaFotograma()
    {
        // -- Pausa hasta el siguiente "fotograma" del juego --
        Hardware.Pausa(20);
    }


    public static bool Colision(ElemGrafico e1, ElemGrafico e2)
    {
        // No se debe chocar con un elemento oculto
        if ((e1.visible == false) || (e2.visible == false))
            return false;
        // Ahora ya compruebo coordenadas
        if ((e1.x + e1.ancho > e2.x)
              && (e1.x < e2.x + e2.ancho)
              && (e1.y + e1.alto > e2.y)
              && (e1.y < e2.y + e2.alto))
            return true;
        else
            return false;
    }


    public static bool EsPosibleMover(int x, int y, int xFin, int yFin)
    {
        // Veo si choca con algún elemento del fondo
        for (int i = 0; i < elementosFondo; i++)
            if ((fondos[i].x + fondos[i].ancho > x)
              && (fondos[i].x < xFin)
              && (fondos[i].y + fondos[i].alto > y)
              && (fondos[i].y < yFin))
            return false;

        // Si no ha chocado con ninguno, es posible moverse a esa posición
        return true;
    }
```

```
public static void AvanzarNivel()
{
     // Borro la pantalla y aviso del nuevo nivel
     Hardware.Pausa(200);
     Hardware.BorrarPantallaOculta();
     Hardware.EscribirTextoOculta("Nivel: "+Convert.ToString(premiosVisibles+1),
               350, 300, /* Coordenadas */ 255, 255, 255, /* Colores */ tipoDeLetra);
     Hardware.VisualizarOculta();
     Hardware.Pausa(1000);

     // Activo un enemigo mas
     if (enemigosVisibles < numEnemigos)
     {
          enemigosVisibles++;
          for (int i = 0; i < enemigosVisibles; i++)
               enemigos[i].visible = true;
     }

     // Y un obstaculo mas
     if (obstaculosVisibles < numObstaculos)
     {
          obstaculosVisibles++;
          for (int i = 0; i < obstaculosVisibles; i++)
               obstaculos[i].visible = true;
     }

     // Y un premio mas, y los recoloco
     if (premiosVisibles < numPremios)
     {
          premiosVisibles++;
          for (int i = 0; i < premiosVisibles; i++)
               do
               {
                    premios[i].x = generador.Next(50, 700);
                    premios[i].y = generador.Next(30, 550);
                    premios[i].visible = true;
                    premios[i].ancho = 34;
                    premios[i].alto = 18;
               }
               while (!EsPosibleMover(premios[i].x, premios[i].y,
                         premios[i].x + premios[i].ancho, premios[i].y + premios[i].alto));
     }
     premiosRestantes = premiosVisibles;

     personaje.x = personaje.xInicial;
     personaje.y = personaje.yInicial;
}


public static void Main()
{

     InicializarJuego();

     while (!juegoTerminado)
     {
          InicializarPartida();
          MostrarPresentacion();
```

```
            // ------ Bucle de juego ------
            while (!partidaTerminada)
            {
                Dibujar();
                ComprobarTeclas();
                MoverElementos();
                ComprobarColisiones();
                PausaFotograma();
            } // Fin del bucle de juego
        }     // Fin de partida
    }         // Fin de Main
}
```

31. Añadiendo soporte de joystick

Casi cualquier biblioteca de juegos nos permitirá usar un joystick, o un gamepad, o un volante. En general, todos ellos se manejan básicamente igual: podremos saber si se ha inclinado en una dirección u otra (tendremos varios "ejes"), así como si se ha pulsado algún botón.

En el caso de SDL, que es la librería que usa nuestro juego "por debajo", existe una función "GetAxis", que nos dice cómo de inclinado está un eje, desde 0 (sin inclinar) hasta 32767 (totalmente inclinado en un sentido) o -32768 (totalmente inclinado en sentido contrario). Si se tratara de un joystick digital, en vez de analógico, no obtendríamos valores intermedios.

En nuestro juego, no necesitamos saber si está muy inclinado o poco inclinado, sino que nos basta saber si se ha pedido mover el personaje hacia un lado u otro, así que podemos ampliar la clase Hardware para crear funciones "JoystickDerecha" y similares, que podrían ser algo así: (no te preocupes si no entiendes por completo esta función, o el resto de la clase Hardware, que están más allá del propósito del curso en este punto)

```
public static bool JoystickDerecha()
{
    // Si no hay joystick, no hay más que hacer
    if (! existeJoystick)
        return false;

    // Leo valores (-32768 a 32767)
    int posX = Sdl.SDL_JoystickGetAxis(joystick, 0);
    if (posX > 16000)
        return true;

    // Si el valor del joystick no es suficiente,
    // supongo que no se ha movido
    return false;
}
```

La función "ComprobarTeclas" del juego, aplicando estas nuevas funciones, sería algo como:

```
if ((Hardware.TeclaPulsada(Hardware.TECLA_DER)
    || Hardware.JoystickDerecha())
    && EsPosibleMover(personaje.x + personaje.incrX, personaje.y,
            personaje.x + personaje.ancho + personaje.incrX,
            personaje.y + personaje.alto))
        personaje.x += personaje.incrX;
```

De forma parecida, podríamos crear una función "JoystickPulsado" en la clase Hardware, que permitiera comprobar si se ha pulsado un cierto botón

```
int posX = Sdl.SDL_JoystickGetAxis(joystick, 0);
```

Puedes descargar el juego completo, incluyendo fuentes, proyecto, imágenes y ejecutable, para probar los cambios por ti mismo.

Ejercicio propuesto: Mejora esta versión del juego, para que se pueda usar el joystick también para entrar al juego y para salir de las pantallas de créditos, etc.

32. Frontón (1): Pelota que rebota

Como acercamiento a la forma de descomponer en juego en una serie de clases que colaboran, vamos a crear un nuevo proyecto sencillo: un juego de "frontón", una pelota que rebota en una pared formada por ladrillos, al estilo del clásico Arkanoid.

Para centrarnos en la lógica y no perder tiempo en detalles adicionales, nuestro juego será en modo consola.

La apariencia final será algo como:

En esta primera entrega, vamos a hacer que una pelota rebote en los bordes de la pantalla. Después añadiremos los "ladrillos" y la posibilidad de que la pelota "los rompa". A continuación, incorporaremos una raqueta manejada por el usuario de forma simultánea. Para finalizar, añadiremos la colisión de la pelota con la raqueta, varias vidas, etc.

La lógica de esta primera entrega debería ser simple:

- Borrar la pantalla y dibujar la pelota en su posición actual
- Calcular la próxima posición, sumando un cierto incremento a la posición actual
- Si llegamos a un extremo de la pantalla, invertir el incremento correspondiente, para que la pelota se mueva en sentido contrario.
- Esperar un instante (por ejemplo, 40 milisegundos, para que el juego se mueva a 25 fotogramas por segundo)
- Repetir... indefinidamente.

Que se podría plantear así...

```
// Fronton - version 1: pelota que rebota

using System;
using System.Threading;

public class Fronton01
{
  public static void Main()
  {
    char pelota = 'O';
    int x = 40; // Posición de la pelota
    int y = 12;
    int incrX = 1; // Incremento (velocidad) de la pelota
    int incrY = -1;
    int minX = 0, maxX = 79; // Límites horizontales
    int minY = 0, maxY = 23; // Límites verticales
    bool terminado = false;

    do
    {
      // Dibujo la pelota
      Console.Clear();
      Console.SetCursorPosition(x, y);
```

```
        Console.Write(pelota);

        // Compruebo si llego al límite de la pantalla
        if ( (x <= minX) || (x >= maxX) )
          incrX = -incrX;
        if ( (y <= minY) || (y >= maxY) )
          incrY = -incrY;

        // Y paso a la siguiente posición
        x += incrX;
        y += incrY;

        // Y espero un poco hasta el siguiente fotograma
        Thread.Sleep( 40 );
      }
      while (!terminado);
    }
}
```

Ejercicio propuesto: Mejora esta versión del juego, para que se pueda abandonar la partida pulsando ESC.

33. Frontón (2): Ladrillos

Una vez que tenemos la pelota que rebota, el siguiente paso es añadir los ladrillos.

Los podríamos dibujar con varios WriteLine, pero parece más razonable crear un array de "strings", de modo que podamos modificar esa estructura que hay que dibujar. Podríamos incluso usar caracteres distintos, como:

```
string[] ladrillos =
{
   "-------=-------=-------=-------=----------------",
   "------=-=------=-=------=-=------=-=-------------",
   "-----=---=---=---=---=---=---=---=--------------",
   "----=-----=-=-----=-=-----=-=-----=-------------",
   "---=--------=-------=-------=-------=------------"
};
```

Y dibujaríamos este array completo nada más borrar la pantalla:

```
Console.Clear();
  foreach( string filaLadrillos in ladrillos)
    Console.WriteLine( filaLadrillos );
```

Pero es que borrarlo sería casi igual de fácil: si la pelota está dentro de las coordenadas de los ladrillos, y en una posición que que no sea un espacio en blanco, cambiamos esa casilla por un espacio y hacemos que la pelota rebote ("dando la vuelta" a su incremento vertical):

```
// Compruebo colisiones con el fondo
if ( ( y < ladrillos.Length)// Si estoy en vertical dentro del fondo
  && ( x < ladrillos[0].Length) // Y en horizontal
  && (ladrillos[y][x] != ' ')) // Y hay ladrillo
{
  incrY = - incrY;
  // Borrar: en 2 pasos, C# no permite modificar un carácter de un string
  ladrillos[y] = ladrillos[y].Remove(x, 1);
  ladrillos[y] = ladrillos[y].Insert(x, " ");
}
```

El fuente completo quedaría:

```
// Fronton - version 1: pelota que rebota y
// choca con los ladrillos (a veces falla en la
// primera línea)
```

```
using System;
using System.Threading;

public class Fronton02
{
  public static void Main()
  {
    char pelota = 'O';
    int x = 40; // Posición de la pelota
    int y = 12;
    int incrX = 1; // Incremento (velocidad) de la pelota
    int incrY = -1;
    int minX = 0, maxX = 79; // Límites horizontales
    int minY = 0, maxY = 23; // Límites verticales
    bool terminado = false;

    // Ladrillos del fondo
    string[] ladrillos =
      {
        "-------=-------=-------=-------=----------------",
        "------=-=-----=-=-----=-=-----=-=---------------",
        "-----=---=---=---=---=---=---=---=--------------",
        "----=-----=-=-----=-=-----=-=-----=-------------",
        "---=-------=-------=-------=-------=------------"
      };


    do
    {
      // Dibujo los ladrilos y la pelota
      Console.Clear();
      foreach( string filaLadrillos in ladrillos)
        Console.WriteLine( filaLadrillos );
      Console.SetCursorPosition(x, y);
      Console.Write(pelota);

      // Compruebo colisiones con el fondo
      if ( ( y < ladrillos.Length)// Si estoy en vertical dentro del fondo
      && ( x < ladrillos[0].Length) // Y en horizontal
      && (ladrillos[y][x] != ' ')) // Y hay ladrillo
      {
        incrY = - incrY;
        // Borrar: no puedo hacer "ladrillos[y][x] = ' ';"
        ladrillos[y] = ladrillos[y].Remove(x, 1);
        ladrillos[y] = ladrillos[y].Insert(x, " ");
      }

      // Compruebo si llego al límite de la pantalla
      if ( (x <= minX) || (x >= maxX) )
        incrX = -incrX;
      if ( (y <= minY) || (y >= maxY) )
        incrY = -incrY;

      // Y paso a la siguiente posición
      x += incrX;
      y += incrY;

      // Y espero un poco hasta el siguiente fotograma
      Thread.Sleep( 40 );
```

```
    }
    while (!terminado);
  }
}
```

Ejercicio propuesto: Mejora esta versión del juego, para que la pared de ladrillos ocupe todo el ancho de la pantalla.

34. Frontón (3): Raqueta simultánea

Dibujar una raqueta que se mueve a la vez que la pelota tampoco es especialmente difícil: no debemos usar directamente "ReadKey", o el juego se quedaría parado hasta que se pulse una tecla, pero podemos comprobar si hay una tecla pendiente con "KeyAvailable", y en ese caso sí leemos la tecla y vemos si es alguna de la que nos interesan:

```
if(Console.KeyAvailable)
{
     tecla=Console.ReadKey(false);
     if(tecla.Key == ConsoleKey.RightArrow
                  && xRaqueta < 79 - raqueta.Length)
          xRaqueta += 2;
     if(tecla.Key == ConsoleKey.LeftArrow
                  && xRaqueta > 2)
          xRaqueta -= 2;
}
```

En este ejemplo, nuestra raqueta se mueve de 2 en 2 y la pelota de en 1, para darnos alguna oportunidad de alcanzarla, y que así el juego sea un poco más jugable.

El resultado puede ser:

```
// Fronton - version 3: raqueta que se mueve de lado a lado

using System;
using System.Threading;

public class Fronton03
{
  public static void Main()
  {
    char pelota = 'O';
    int x = 40; // Posición de la pelota
    int y = 12;
    int xRaqueta = 39;  // Posición de la raqueta
    int yRaqueta = 22;
    int incrX = 1; // Incremento (velocidad) de la pelota
    int incrY = -1;
    int minX = 0, maxX = 79; // Límites horizontales
    int minY = 0, maxY = 23; // Límites verticales
    bool terminado = false;

    string raqueta = "_______";
    ConsoleKeyInfo tecla;

    // Ladrillos del fondo
    string[] ladrillos =
      {
        "-------=-------=-------=-------=----------------",
        "------=-=-----=-=-----=-=-----=-=---------------",
        "-----=---=---=---=---=---=---=---=--------------",
        "----=-----=-=-----=-=-----=-=-----=-------------",
        "---=-------=-------=-------=-------=------------"
      };
```

```
do
{
  // Dibujo los ladrillos y la pelota
  Console.Clear();
  foreach( string filaLadrillos in ladrillos)
    Console.WriteLine( filaLadrillos );
  Console.SetCursorPosition(x, y);
  Console.Write(pelota);
  Console.SetCursorPosition(xRaqueta, yRaqueta);
  Console.Write(raqueta);

  // Muevo la raqueta si se pulsa alguna tecla
  if(Console.KeyAvailable)
  {
       tecla=Console.ReadKey(false);
       if(tecla.Key == ConsoleKey.RightArrow
                    && xRaqueta < 79 - raqueta.Length)
            xRaqueta += 2;
       if(tecla.Key == ConsoleKey.LeftArrow
                    && xRaqueta > 2)
            xRaqueta -= 2;
  }

  // Compruebo colisiones con el fondo
  if ( ( y < ladrillos.Length)// Si estoy en vertical dentro del fondo
  && ( x < ladrillos[0].Length) // Y en horizontal
  && (ladrillos[y][x] != ' ')) // Y hay ladrillo
  {
    incrY = - incrY;
    // Borrar: no puedo hacer "ladrillos[y][x] = ' ';"
    ladrillos[y] = ladrillos[y].Remove(x, 1);
    ladrillos[y] = ladrillos[y].Insert(x, " ");
  }

  // Compruebo si llego al límite de la pantalla
  // Si se sale por abajo...
  if ( y >= maxY )
  {
    // Recolocar pelota y raqueta
    x = 40;
    y = 12;
    xRaqueta = 39;
    yRaqueta = 22;
    incrX = 1;
    incrY = -1;
  }
  // Si llega a un extremo sin salir por abajo: rebota
  if ( (x <= minX) || (x >= maxX) )
    incrX = -incrX;
  if ( (y <= minY) || (y >= maxY) )
    incrY = -incrY;

  // Y paso a la siguiente posición
  x += incrX;
  y += incrY;

  // Y espero un poco hasta el siguiente fotograma
  Thread.Sleep( 40 );
```

```
    }
    while (!terminado);
  }
}
```

Ejercicio propuesto: Este fuente puede fallar si la pelota rebota "hacia arriba" cuando está en la primera línea; corrígelo.

35. Frontón (4): Raqueta y pelota; varias vidas

Hacer que la pelota rebote en la raqueta también es sencillo: deberemos ver si se encuentra en la misma vertical (la misma coordenada Y) y en el mismo rango de coordenadas horizontales (entre la X inicial de la raqueta y la X final de la raqueta), y en ese caso cambiaremos su incremento vertical:

```
if (( y == yRaqueta) &&
  (x >= xRaqueta) && (x <= xRaqueta + raqueta.Length - 1) )
{
   incrY = -incrY;
}
```

Esto todavía no es jugable: la pelota rebota en la raqueta, pero nunca se pierde ninguna vida. Para añadir esta mejora, podemos comprobar si se llega a la parte inferior de la pantalla, y en ese caso recolocar la pelota (tanto en su posición inicial como con su velocidad inicial) y la raqueta, restar una vida, y, en caso de que se agoten todas las vidas, comenzar un partida nueva:

```
if ( y >= maxY )
{
  // Recolocar pelota y raqueta
  x = 40;
  y = 12;
  xRaqueta = 39;
  yRaqueta = 22;
  incrX = 1;
  incrY = -1;
  // Compruebo final de partida
  vidas --;
  if (vidas == 0)
  {
     // Reinicio datos
     vidas = 3;
     // Pantalla de presentación
     Console.Clear();
     Console.WriteLine("Pulsa intro para jugar...");
     Console.ReadLine();

  }
}
```

El fuente completo quedaría:

```
// Fronton - version 5: tres vidas

using System;
using System.Threading;

public class Fronton05
{
  public static void Main()
  {
    char pelota = 'O';
    int x = 40; // Posición de la pelota
    int y = 12;
    int xRaqueta = 39;  // Posición de la raqueta
```

```
int yRaqueta = 22;
int vidas = 3;
int incrX = 1; // Incremento (velocidad) de la pelota
int incrY = -1;
int minX = 0, maxX = 79; // Límites horizontales
int minY = 0, maxY = 23; // Límites verticales
bool terminado = false;

string raqueta = "________";
ConsoleKeyInfo tecla;

// Ladrillos del fondo
string[] ladrillos =
  {
    "-------=-------=-------=-------=----------------",
    "------=-=-----=-=-----=-=-----=-=---------------",
    "-----=---=---=---=---=---=---=---=--------------",
    "----=-----=-=-----=-=-----=-=-----=-------------",
    "---=-------=-------=-------=-------=------------"
  };


do
{
  // Dibujo los ladrillos y la pelota
  Console.Clear();
  foreach( string filaLadrillos in ladrillos)
    Console.WriteLine( filaLadrillos );
  Console.SetCursorPosition(x, y);
  Console.Write(pelota);
  Console.SetCursorPosition(xRaqueta, yRaqueta);
  Console.Write(raqueta);
  Console.SetCursorPosition(0, maxY);
  Console.Write("Vidas: {0}", vidas);

  // Muevo la raqueta si se pulsa alguna tecla
  if(Console.KeyAvailable)
  {
       tecla=Console.ReadKey(false);
       if(tecla.Key == ConsoleKey.RightArrow
                    && xRaqueta < 79 - raqueta.Length)
            xRaqueta += 2;
       if(tecla.Key == ConsoleKey.LeftArrow
                    && xRaqueta > 2)
            xRaqueta -= 2;
  }

  // Compruebo colisiones con el fondo
  if ( ( y < ladrillos.Length)// Si estoy en vertical dentro del fondo
  && ( x < ladrillos[0].Length) // Y en horizontal
  && (ladrillos[y][x] != ' ')) // Y hay ladrillo
  {
    incrY = - incrY;
    // Borrar: no puedo hacer "ladrillos[y][x] = ' ';"
    ladrillos[y] = ladrillos[y].Remove(x, 1);
    ladrillos[y] = ladrillos[y].Insert(x, " ");
  }
  // Y colisiones con la raqueta
  if (( y == yRaqueta) &&
```

```
          (x >= xRaqueta) && (x <= xRaqueta + raqueta.Length - 1) )
        {
           incrY = -incrY;
        }

        // Compruebo si llego al límite de la pantalla
        // Si se sale por abajo...
        if ( y >= maxY )
        {
          // Recolocar pelota y raqueta
          x = 40;
          y = 12;
          xRaqueta = 39;
          yRaqueta = 22;
          incrX = 1;
          incrY = -1;
          // Compruebo final de partida
          vidas --;
          if (vidas == 0)
          {
              // Reinicio datos
              vidas = 3;
              // Pantalla de presentación
              Console.Clear();
              Console.WriteLine("Pulsa intro para jugar...");
              Console.ReadLine();

          }
        }
        // Si llega a un extremo sin salir por abajo: rebota
        if ( (x <= minX) || (x >= maxX) )
          incrX = -incrX;
        if  (y <= minY)
          incrY = -incrY;

        // Y paso a la siguiente posición
        x += incrX;
        y += incrY;

        // Y espero un poco hasta el siguiente fotograma
        Thread.Sleep( 40 );
      }
      while (!terminado);
    }
}
```

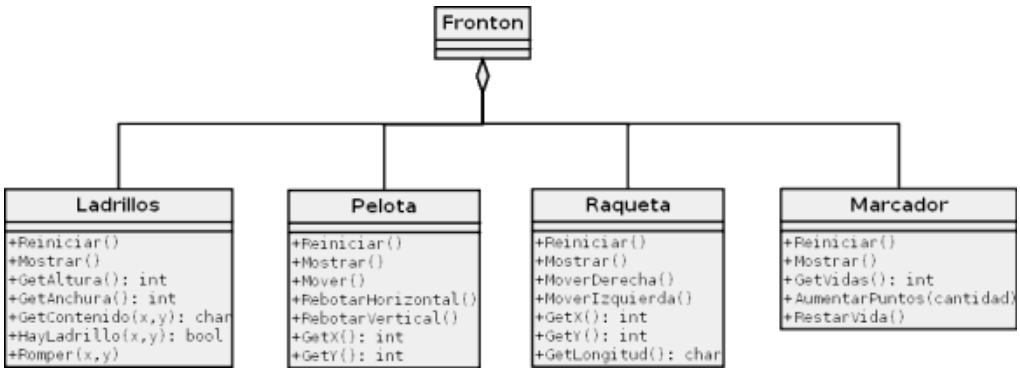
Ejercicio propuesto: Mejora este fuente para que vuelvan a aparecer los ladrillos cuando se hayan roto todos. Añade un marcador que muestre también los puntos que se van obteniendo.

36. Frontón (5): Descomposición en clases

Descomponer un programa en clases es una tarea que no suele ser sencilla, especialmente si el programa del que partimos es grande. En nuestro caso es sólo un fuente de 120 líneas, yy aun así va a ser trabajoso...

El primer paso es descomponer el problema, pensando qué "objetos" lo forman y cómo interaccionan entre ellos. Por ejemplo, en nuestro caso tenemos una "Pelota", una "Raqueta", una "Pared de ladrillos", un "Marcador". Pero esta descomposición no tiene por qué ser única: también se podría pensar que cada uno de los ladrillos fuera también un tipo de objeto independiente (una "clase"), o crear una "Presentación". El detallar más o menos depende de cada caso, de la complejidad de lo que queda por descomponer, incluso de los criterios de cada diseñador. Por supuesto, suele haber una clase que representa a toda la aplicación (en nuestro caso, al juego, "Frontón").

Una vez hemos decidido qué clases van a formar nuestro programa, hay que analizar qué tareas tendrá que realizar cada una de ellos (los "métodos"). Este paso también puede ayudar a descubrir clases que hubiéramos pasado por alto. Por ejemplo, en nuestro caso, la raqueta se podría "Mover a la derecha" y "Mover a la izquierda", mientras que la pelota simplemente tendría un "Mover", que la desplazase a la siguiente posición que le correspondiera. La "Pared de ladrillos podría tener un método que permitiera saber si hay un ladrillo en una cierta posición, o que permitiera romper un cierto ladrillo. Todas ellas tendrían también un método "Mostrar" o "Dibujar", para representarlas en pantalla en su posición actual y con su estado actual. También podrían tener un "Reiniciar" que llevara cada elemento a su posición inicial y con su estado inicial (la velocidad prevista, para la pelota, o todos los ladrillos de partida, para la pared).



El siguiente paso, que también nos puede ayudar a descubrir clases que hemos pasado por alto, es plantearnos cómo podría ser el "Main" del programa y de qué forma usaría cada una de esas clases, buscando que el programa resultante sea tan legible como nos sea posible. Por ejemplo, un fragmento podría ser así:

```
// Dibujo los ladrillos y la pelota
Console.Clear();
miPared.Mostrar();
miRaqueta.Mostrar();
miPelota.Mostrar();
miMarcador.Mostrar();

// Compruebo colisiones con el fondo
if (miPared.HayLadrillo(miPelota.GetX(), miPelota.GetY()))
{
     miPelota.RebotarVertical();
     miPared.Romper(miPelota.GetX(), miPelota.GetY());
     miMarcador.AumentarPuntos(10);
}
```

El fuente completo, incuyendo todas las clases en un único fichero (que es una práctica poco recomendable, pero que usaremos en este primer acercamiento) podría ser:

```
// Fronton - version 6: descompuesto en clases

using System;
using System.Threading;

class FrontonClases
{
     static Raqueta miRaqueta;
     static Ladrillos miPared;
     static Pelota miPelota;
     static Marcador miMarcador;

     public static void Main()
     {
          miRaqueta = new Raqueta();
          miPared = new Ladrillos();
          miPelota = new Pelota();
          miMarcador = new Marcador();

          int minX = 0, maxX = 79; // Límites horizontales
          int maxY = 23;           // Límites verticales
          bool terminado = false;
```

```
ConsoleKeyInfo tecla;

do
{
     // Dibujo los ladrillos y la pelota
     Console.Clear();
     miPared.Mostrar();
     miRaqueta.Mostrar();
     miPelota.Mostrar();
     miMarcador.Mostrar();

     // Muevo la raqueta si se pulsa alguna tecla
     if (Console.KeyAvailable)
     {
          tecla = Console.ReadKey(false);
          if (tecla.Key == ConsoleKey.RightArrow
                    && (miRaqueta.GetX() + miRaqueta.GetLongitud() - 1 < maxX))
               miRaqueta.MoverDerecha();
          if (tecla.Key == ConsoleKey.LeftArrow
                    && miRaqueta.GetX() > 1)
               miRaqueta.MoverIzquierda();
     }

     // Compruebo colisiones con el fondo
     if (miPared.HayLadrillo(miPelota.GetX(), miPelota.GetY()))
     {
          miPelota.RebotarVertical();
          miPared.Romper(miPelota.GetX(), miPelota.GetY());
          miMarcador.AumentarPuntos(10);
     }

     // Y colisiones con la raqueta
     if ((miPelota.GetX() >= miRaqueta.GetX()
          && miPelota.GetX() <= miRaqueta.GetX() + miRaqueta.GetLongitud() - 1)
          && miPelota.GetY() == miRaqueta.GetY())
          miPelota.RebotarVertical();

     // Compruebo si llego al límite de la pantalla
     // Si se sale por abajo...
     if (miPelota.GetY() >= maxY)
     {
          miPelota.Reiniciar();
          miRaqueta.Reiniciar();
          miMarcador.RestarVida();
          if (miMarcador.GetVidas() == 0)
          {
                // Reinicio datos
                miMarcador.Reiniciar();
                miPared.Reiniciar();
                // Pantalla de presentación
                Console.Clear();
                Console.WriteLine("Pulsa intro para jugar...");
                Console.ReadLine();
          }
     }
     // // Si llega a un extremo sin salir por abajo: rebota
     if ((miPelota.GetX() <= minX)
               || (miPelota.GetX() >= miPared.GetAnchura()-1))
          miPelota.RebotarHorizontal();
```

```
            if (miPelota.GetY() >= maxY)
                miPelota.RebotarVertical();

            // Y paso a la siguiente posición
            miPelota.Mover();

            // Y espero un poco hasta el siguiente fotograma
            Thread.Sleep(40);

        } while (! terminado);
    }
}

// ---------------------------------------------------------------

class Ladrillos
{
    private string[] filasLadrillos =
    { "-------=-------=-------=-------=-------=-------=-------=-------=---",
      "------=-=-----=-=-----=-=-----=-=-----=-=-----=-=-----=-=-----=----",
      "-----=---=---=---=---=---=---=---=---=---=---=---=---=---=---=-----",
      "----=-----=-----=-=-----=-=-----=-=-----=-=-----=-=-----=-=------",
      "---=-------=-------=-------=-------=-------=-------=-------=-------"
    };

    public void Reiniciar()
    {
        filasLadrillos[0] = "-------=-------=-------=-------=-------=-------=-------=-------=---";
        filasLadrillos[1] = "------=-=-----=-=-----=-=-----=-=-----=-=-----=-=-----=-=-----=----";
        filasLadrillos[2] = "-----=---=---=---=---=---=---=---=---=---=---=---=---=---=---=-----";
        filasLadrillos[3] = "----=-----=-----=-=-----=-=-----=-=-----=-=-----=-=-----=-=------";
        filasLadrillos[4] = "---=-------=-------=-------=-------=-------=-------=-------=-------";
    }

    public void Mostrar()
    {
        Console.SetCursorPosition(0, 0);
        foreach (string filaActual in filasLadrillos)
            Console.WriteLine(filaActual);
    }

    public int GetAltura()
    {
        return filasLadrillos.Length;
    }

    public int GetAnchura()
    {
        return filasLadrillos[0].Length;
    }

    public char GetContenido(int x, int y)
    {
        return filasLadrillos[y][x];
    }

    public bool HayLadrillo(int x, int y)
    {
        if ( y >= GetAltura() ) return false;
        if ( x >= GetAnchura() ) return false;
```

```
        if (filasLadrillos[y][x] == ' ') return false;
        return true;
    }

    public void Romper(int x, int y)
    {
        filasLadrillos[y] = filasLadrillos[y].Remove(x, 1);
        filasLadrillos[y] = filasLadrillos[y].Insert(x, " ");
    }
}

// ------------------------------------------------------------

class Pelota
{
    private int x, y;
    private char simbolo;
    int incrX, incrY;

    public Pelota()
    {
        simbolo = 'O';
        Reiniciar();
    }

    public void Reiniciar()
    {
        x = 40;
        y = 12;
        incrX = 1;
        incrY = -1;
    }

    public void Mostrar()
    {
        Console.SetCursorPosition(x,y);
        Console.Write(simbolo);
    }

    public void Mover()
    {
        x += incrX;
        y += incrY;
        // Posible fallo al rebotar en los extremos
        if (x<0) x=0;
        if (y<0) y=0;
    }

    public void RebotarHorizontal()
    {
        incrX = -incrX;
    }

    public void RebotarVertical()
    {
        incrY = -incrY;
    }

    public int GetX()
    {
```

```
            return x;
        }

        public int GetY()
        {
            return y;
        }
    }

    // -----------------------------------------------------------

    class Raqueta
    {
        private int x;
        private int y;
        private string dibujo;

        public Raqueta()
        {
            dibujo = "_________";
            Reiniciar();
        }

        public void Reiniciar()
        {
            x = 39;
            y = 22;
        }

        public void Mostrar()
        {
            Console.SetCursorPosition(x, y);
            Console.Write(dibujo);
        }

        public void MoverDerecha()
        {
            x += 2;
        }

        public void MoverIzquierda()
        {
            x -= 2;
        }

        public int GetX()
        {
            return x;
        }

        public int GetY()
        {
            return y;
        }

        public int GetLongitud()
        {
            return dibujo.Length;
        }
    }
```

```
// ------------------------------------------------------------

class Marcador
{
     private int x, y;
     private int puntos, vidas;

     public Marcador()
     {
          x = 0;
          y = 23;
          Reiniciar();
     }

     public void Reiniciar()
     {
          puntos = 0;
          vidas = 3;
     }

     public void Mostrar()
     {
          Console.SetCursorPosition(x, y);
          Console.Write("Vidas: {0}    Puntos: {1}", vidas, puntos);
     }

     public int GetVidas()
     {
          return vidas;
     }

     public void AumentarPuntos(int cantidad)
     {
          puntos += cantidad;
     }

     public void RestarVida()
     {
          vidas--;
     }
}
```

Ejercicio propuesto: Mejora este fuente para que cada elemento tenga su propio color (distinto para la pelota y para la raqueta o cada tipo de ladrillo).

37. Descomposición en clases del juego en modo gráfico.

Nuestro "arkanoid" en modo consola ha pasado de ocupar 120 líneas, cuando era un único fuente, a casi 300 línwas cuando lo hemos descompuesto en clases.

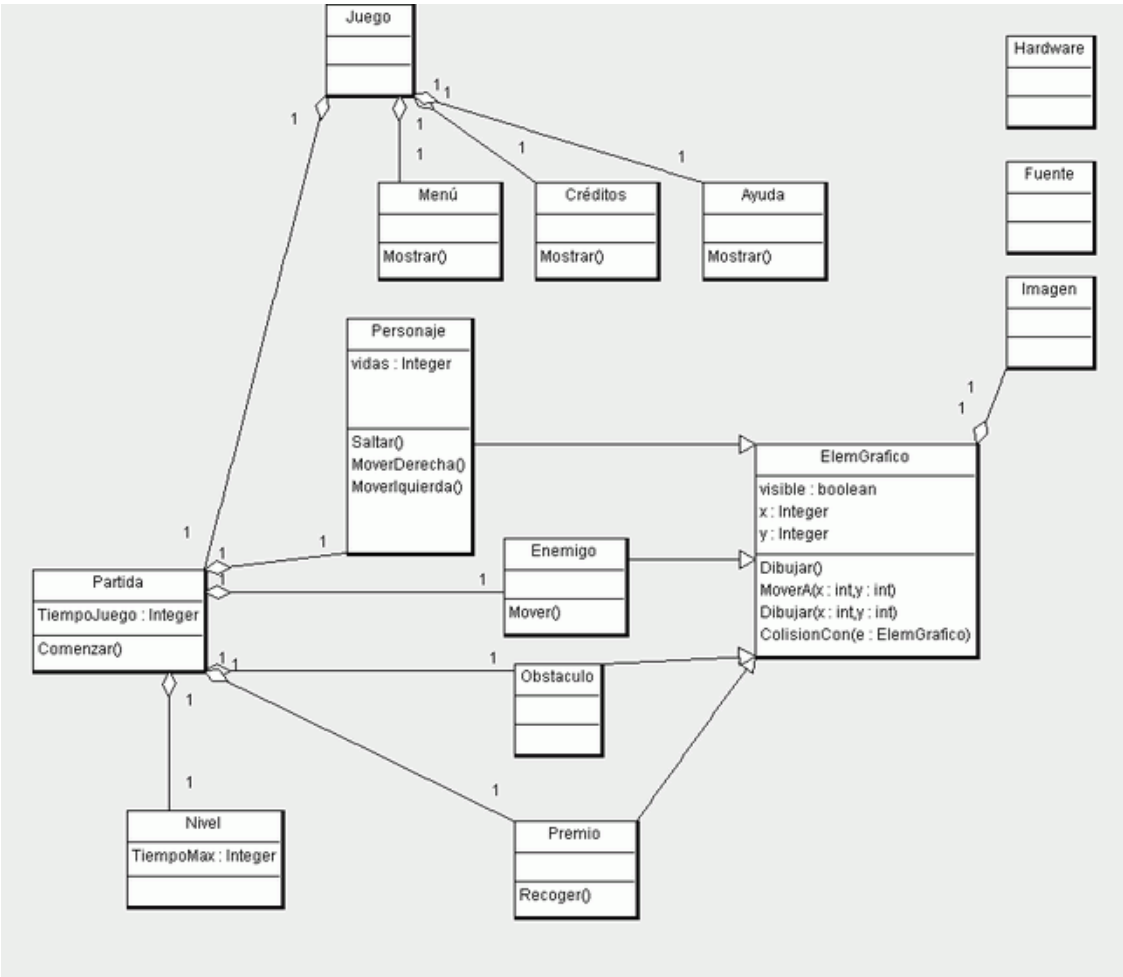
Esto suele ocurrir. Descomponer en clases un programa existente permite obtener fuentes indiviuales más pequeños, y, por tanto, más legibles y con una misión más clara. También permite repartir el trabajo entre grupos de programadores, para que sea más fácil desarrollar en paralelo. Pero no es una tarea fácil, y es habitual que el resultado "sea más grande" que el original (en cuanto a "número de líneas totales de programa").

Descomponer nuestro juego en modo gráfico va a ser bastante más trabajoso, porque era un proyecto de un tamaño mucho mayor que el "frontón". A cambio, a partir de este momento será más fácil ampliarlo y localizar fallos, así que vamos a hacerlo...

Esta vez podríamos pensar en clases como Descomponer nuestro juego en modo gráfico va a ser bastante más trabajoso, porque era un proyecto de un tamaño mucho mayor que el "frontón". A cambio, a partir de este momento será más fácil ampliarlo y localizar fallos, así que vamos a hacerlo...

Por una parte, podríamos distinguir la clase "Juego", que engloba a todo el proyecto, de una clase "Partida", que sería una sesión concreta de juego. El "Juego" podría contener un menú inicial, una pantalla de ayuda, otra de créditos (información sobre los desarrolladores). La "Partida" necesitaría un personaje, varios enemigos, varios premios, varios obstáculos, y tendremos que recorrer una serie de habitaciones, que podríamos llamar "niveles" de juego. Más adelante añadiremos posibilidades adicionales, como que nuestro personaje pueda saltar, o caer desde las alturas, o disparar... pero de momento, vamos a comenzar "imitando" las funcionalidades de la versión anterior del juego.

El diagrama de clases, sin detallar apenas métodos, y sin incluir otras clases que es posible que nuestro juego necesite más adelante, podría ser:



El fuente se va a convertir en 15 clases, con más de 1.500 líneas de código (muchas de las cuales serán comentarios, claro), así que no tiene sentido verlo entero, pero sí algún fragmento.

Por ejemplo, un "Elemento gráfico" ya no será un struct, sino una clase, en la que los campos del struct se convertirán en atributos, y además tendrá algún método, como los que permitan dibujarlo en pantalla o comprobar si hay colisión con otro elemento gráfico:

```
public class ElemGrafico
{
    // Atributos

    protected bool visible;
    protected int x;
    protected int y;
    // ...

    // Constructor: Carga la imagen que representara a este elemento grafico
    // Ancho y alto prefijados
```

```
    public ElemGrafico(string nombre)
    {
        imagen = new Imagen(nombre);
        visible = true;
        ancho = 32;
        alto = 32;
    }

    // Constructor: Carga la imagen que representara a este elemento grafico
    // Ancho y alto indicados por el usuario
    public ElemGrafico(string nombre, int nuevoAncho, int nuevoAlto)
    {
        imagen = new Imagen(nombre);
        visible = true;
        ancho = nuevoAncho;
        alto = nuevoAlto;
    }


    // Dibuja el elemento gráfico en sus coordenadas actuales
    public  void DibujarOculta()
    {
        if (visible)
            imagen.DibujarOculta(x, y);
    }

    //...
```

Y un "Enemigo" será un tipo de "Elemento gráfico", que además será capaz de moverse por sí mismo (calcular la siguiente posición de su movimiento):

```
public class Enemigo : ElemGrafico
{
    public Enemigo()
        : base("enemigo.png", 36, 42)
    {
        incrX = 5;
    }

    //  Mover el personaje a su siguiente posición (movimiento automático)
    public  void Mover()
    {
        if (x < 50)
            incrX = 5;

        if (x > 700)
            incrX = -5;

        x += incrX;
    }
}
```

Lo que antes eran las funciones "MostrarPresentacion", "MostrarAyuda" y "MostrarCreditos" ahora se convertirán en el método "Mostrar" de cada una de esas clases. La clase "Partida" absorberá toda la lógica de una sesión de juego, que ahora se verá simplificada porque podemos delegar en clases auxiliares. Por ejemplo, el método "Dibujar", que antes ocupaba 42 líneas, ahora se convertirá en:

```
public void Dibujar()
{
    Hardware.BorrarPantallaOculta(0, 0, 0);
    miMarcador.DibujarOculta();
      // Elementos del nivel: fondo, premios y enemigos
```

```
    miNivel.DibujarOculta();
    miPersonaje.DibujarOculta();
    Hardware.VisualizarOculta();
}
```

Puedes descargar el juego completo, incluyendo fuentes, proyecto, imágenes y ejecutable, para probar los cambios por ti mismo.

38. Varios niveles.

Ahora mismo, tenemos una clase "Nivel", que oculta todos los detalles de la pantalla de juego que estamos recorriendo, y que contiene internamente un array de dos dimensiones para representar las casillas repetitivas de fondo:

```
byte[,] fondo =
{
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
};
```

Una primera forma de conseguir que el juego pueda tener varios niveles es convertir ese aray de 2 dimensiones en uno de 3 dimensiones, en el que el primer índice nos permitirá acceder a una pantalla u otra. Como ya existía un método "Avanzar", todos los cambios se podrían hacer dentro de la clase "Nivel", sin que afecte para nada al resto de las clases que forman el juego.

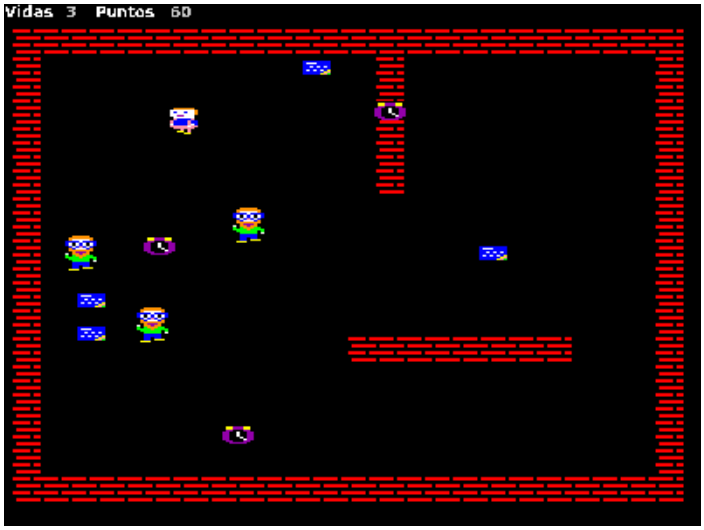
Vamos por partes...

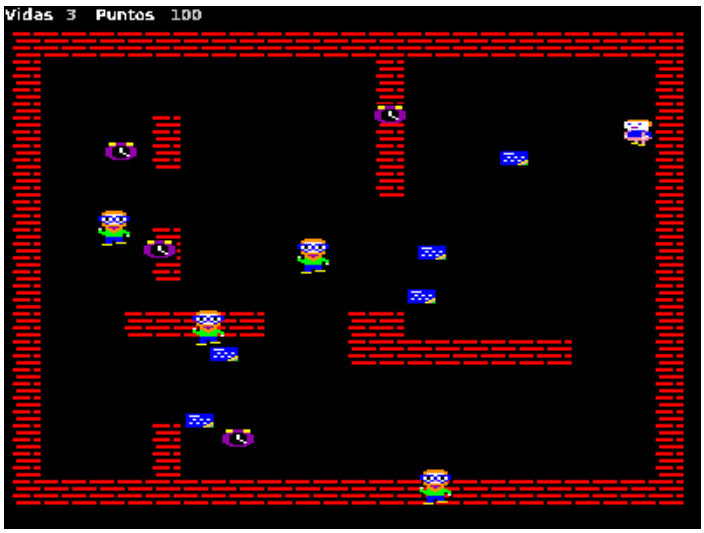
En primer lugar, usaremos un array de tres dimensiones, en vez de uno de dos:

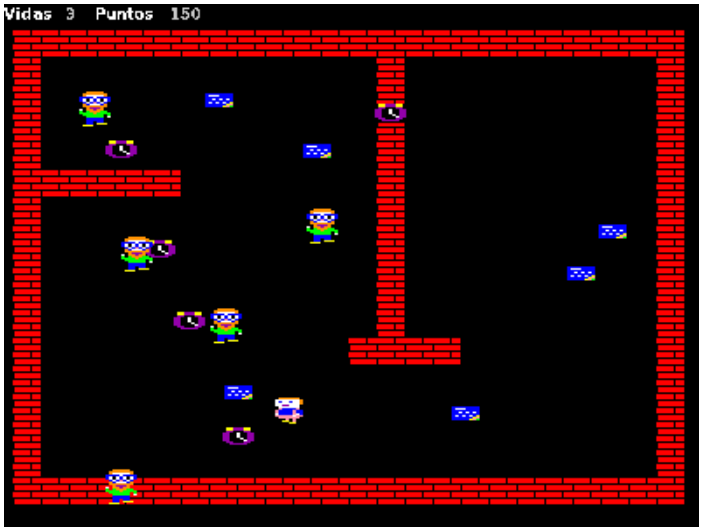
```
public byte[,,] fondo =
{
    {
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
```

```
    },
    {
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,1,1,1,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
    },
    {
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
        {1,1,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
    }
};
```

Que se convertirá en tres pantallas como éstas:







En segundo lugar, la función "Avanzar" deberá cambiar el número de nivel actual (que empezará siendo cero). Deberemos prever que el usuario puede llegar a recorrer todos, y cuando esto ocurra, habrá que dar la partida por terminada o volver al nivel 0. La segunda opción sería simplemente así:

```
if (nivel < 2)
     nivel++;
else
     nivel = 0;
```

Si el método "Dibujar" accediera directamente al array "fondo", ya (casi) habríamos terminado: bastaría cambiar "fondo[fila,columna]" por "fondo[nivel,fila,columna]". Pero nuestro "Dibujar" es así:

```
public void DibujarOculta()
     {
          for (int i = 0; i < elementosFondo; i++)  // Imágenes del fondo
               fondos[i].DibujarOculta();

          // ...
```

Es decir, estamos accediendo a un array auxiliar, formado por "elementos gráficos", que habíamos creado para simplificar las colisiones, así que tendremos que volver a generar ese array. Eso, que antes se hacía dentro del constructor, ahora lo podríamos llevar a una función auxiliar, que se llamara tanto desde el constructor como desde "Avanzar":

```
public void RellenarArrayDeFondo()
{

     elementosFondo = 0;
```

```
        for (int fila = 0; fila < altoFondo; fila++)  // Fondo
            for (int col = 0; col < anchoFondo; col++)
                if (fondo[nivel, fila, col] != 0)
                    elementosFondo++;
        // Y reservo espacio para el array que los contiene
        fondos = new ElemGrafico[elementosFondo];

        int posicFondo = 0;
        for (int fila = 0; fila < altoFondo; fila++)  // Fondo
            for (int col = 0; col < anchoFondo; col++)
                if (fondo[nivel, fila, col] != 0)
                {
                    fondos[posicFondo] = new ElemGrafico("pared.png", anchoCasillaFondo, altoCasillaFondo);
                    fondos[posicFondo].MoverA(margenXFondo + col * anchoCasillaFondo,
                        margenYFondo + fila * altoCasillaFondo);
                    fondos[posicFondo].SetVisible(true);
                    posicFondo++;
                }
    }
```

La clase "Nivel" completa podría quedar así:

```
/// <summary>
///  Nivel: el nivel actual de juego
///  @author Nacho Cabanes
///  @see Menu Juego
/// </summary>

/* ---------------------------------------------------
    Parte de FreddyTwo

    Versiones hasta la fecha:

    Num.    Fecha       Por / Cambios
    -----------------------------------------------------
    0.01  03-Feb-2012  Nacho Cabanes
                          Versión inicial: desglosado a partir de
                            FreddyOne (Juego.cs)
    0.02  04-Feb-2012  Cristina Giner Olcina y Jorge Bernabeu Mira
                          Varios niveles consecutivos
  ----------------------------------------------------- */

using System;  // Para Random

public class Nivel
{
    // Atributos
    Random generador;

    //private int tiempoMax;
    private int premiosRestantes;
    private int premiosVisibles, enemigosVisibles, obstaculosVisibles;

    byte anchoFondo = 24;
    byte altoFondo = 17;
    short margenXFondo = 10;
    byte margenYFondo = 30;
    byte anchoCasillaFondo = 32;
    byte altoCasillaFondo = 32;
    Imagen imgPared;
```

```
int numPremios, numEnemigos, numObstaculos;

Obstaculo[] obstaculos;
Enemigo[] enemigos;
Premio[] premios;
ElemGrafico[] fondos;
int elementosFondo;
int nivel;


public byte[,,] fondo =
{
    {
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
    },
    {
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,1,1,1,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
    },
    {
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,1,1,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
```

```
            {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
            {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
            {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
        }
};


public Nivel()
{
    numPremios = 20;
    numEnemigos = 12;
    numObstaculos = 20;
    nivel = 0;
    obstaculos = new Obstaculo[numObstaculos];
    enemigos = new Enemigo[numEnemigos];
    premios = new Premio[numPremios];
    generador = new Random();

    // y para cada uno de uno de ellos
    RellenarArrayDeFondo();

    // Preparo elementos
    for (int i = 0; i < numObstaculos; i++)  // Obstaculos
    {
        obstaculos[i] = new Obstaculo();
    }


    for (int i = 0; i < numEnemigos; i++)  // Enemigos
    {
        enemigos[i] = new Enemigo();
    }

    for (int i = 0; i < numPremios; i++)  // Premios
    {
        premios[i] = new Premio();
    }
    // Al empezar el primer nivel, al menos un precio debe estar visible
    premios[0].SetVisible(true);
    premiosRestantes = 1;
    imgPared = new Imagen("pared.png");


}


//funcion auxiliar
public void RellenarArrayDeFondo()
{

    elementosFondo = 0;
    for (int fila = 0; fila < altoFondo; fila++)  // Fondo
        for (int col = 0; col < anchoFondo; col++)
```

```
                if (fondo[nivel, fila, col] != 0)
                    elementosFondo++;
        // Y reservo espacio para el array que los contiene
        fondos = new ElemGrafico[elementosFondo];

        int posicFondo = 0;
        for (int fila = 0; fila < altoFondo; fila++)  // Fondo
            for (int col = 0; col < anchoFondo; col++)
                if (fondo[nivel, fila, col] != 0)
                {
                    fondos[posicFondo] = new ElemGrafico("pared.png", anchoCasillaFondo, altoCasillaFondo);
                    fondos[posicFondo].MoverA(margenXFondo + col * anchoCasillaFondo,
                        margenYFondo + fila * altoCasillaFondo);
                    fondos[posicFondo].SetVisible(true);
                    posicFondo++;
                }
}


/// Deja todo como es necesario para una nueva partida
public void Reiniciar()
{
    premiosVisibles = 1;
    enemigosVisibles = 0;
    obstaculosVisibles = 0;

    for (int i = 0; i < numObstaculos; i++)  // Obstaculos
    {
        obstaculos[i].MoverA(generador.Next(50, 700), generador.Next(30, 550));
        obstaculos[i].SetVisible(false);
    }


    for (int i = 0; i < numEnemigos; i++)  // Enemigos
    {
        enemigos[i].MoverA(generador.Next(50, 700), generador.Next(30, 550));
        enemigos[i].SetVisible(false);
    }

    for (int i = 0; i < numPremios; i++)  // Premios
    {
        do
        {
            premios[i].MoverA(generador.Next(50, 700),
                generador.Next(30, 550));
            premios[i].SetVisible(true);
        }
        while (!EsPosibleMover(premios[i].GetX(), premios[i].GetY(),
                    premios[i].GetXFin(), premios[i].GetYFin()));
        premios[i].SetVisible(false);
    }
    // Al empezar el primer nivel, al menos un precio debe estar visible
    premios[0].SetVisible(true);
    premiosRestantes = 1;
}


public void DibujarOculta()
{
```

```
        for (int i = 0; i < elementosFondo; i++)  // Imágenes del fondo
            fondos[i].DibujarOculta();

        for (int i = 0; i < numObstaculos; i++)  // Obstáculos
            obstaculos[i].DibujarOculta();

        for (int i = 0; i < numEnemigos; i++)  // Enemigos
            enemigos[i].DibujarOculta();

        for (int i = 0; i < numPremios; i++)  // Premios
            premios[i].DibujarOculta();

    }

    public bool EsPosibleMover(int x, int y, int xFin, int yFin)
    {
        // Veo si choca con algún elemento del fondo
        for (int i = 0; i < elementosFondo; i++)
            if (fondos[i].ColisionCon(x,y,xFin,yFin))
                return false;

        // Si no ha chocado con ninguno, es posible moverse a esa posición
        return true;
    }

    public int ObtenerPuntosPosicion(int x, int y, int xFin, int yFin)
    {
        // Veo si recoge algún premio
        for (int i = 0; i < numPremios; i++)
            if (premios[i].ColisionCon(x, y, xFin, yFin))
            {
                premios[i].SetVisible(false);
                premiosRestantes--;
                if (premiosRestantes == 0)
                    Avanzar();
                return 10;
            }

        // O si toca algún obstáculo
        for (int i = 0; i < numObstaculos; i++)
            if (obstaculos[i].ColisionCon(x, y, xFin, yFin))
                return -1;

        // O si choca con algún enemigo
        for (int i = 0; i < numEnemigos; i++)
            if (enemigos[i].ColisionCon(x, y, xFin, yFin))
                return -1;

        // Si llego hasta aqui, es que no hay puntos que devolver
        return 0;

    }



    public int GetNumEnemigos()
    {
        return numEnemigos;
    }
```

```
    public Enemigo GetEnemigo(int num)
    {
        return enemigos[num];
    }


    public void Avanzar()
    {
        // Borro la pantalla y aviso del nuevo nivel
        Hardware.Pausa(200);
        Hardware.BorrarPantallaOculta();
        Hardware.VisualizarOculta();
        Hardware.Pausa(1000);

        // Cambio los "ladrillos" del fondo
        if (nivel < 2)
            nivel++;
        else
            nivel = 0;

        RellenarArrayDeFondo();


        // Activo un enemigo mas
        if (enemigosVisibles < numEnemigos)
        {
            enemigosVisibles++;
            for (int i = 0; i < enemigosVisibles; i++)
                enemigos[i].SetVisible(true);
        }

        // Y un obstaculo mas
        if (obstaculosVisibles < numObstaculos)
        {
            obstaculosVisibles++;
            for (int i = 0; i < obstaculosVisibles; i++)
                obstaculos[i].SetVisible(true);
        }

        // Y un premio mas, y los recoloco
        if (premiosVisibles < numPremios)
        {
            premiosVisibles++;
            for (int i = 0; i < premiosVisibles; i++)
                do
                {
                    premios[i].MoverA(generador.Next(50, 700),
                        generador.Next(30, 550));
                    premios[i].SetVisible(true);
                }
                while (!EsPosibleMover(premios[i].GetX(), premios[i].GetY(),
                        premios[i].GetXFin(), premios[i].GetYFin()));
        }
        premiosRestantes = premiosVisibles;
    }

} /* Fin de la clase Nivel */
```

Ejercicio propuesto: Amplía este fuente para que tenga 5 niveles, en vez de 3.

39. Un personaje animado.

Para que el movimiento de un personaje parezca más real, deberíamos tener distintas imágenes, una para cada "fotograma" de ese movimiento.

La forma más sencilla sería que cada "Elemento gráfico" no tuviera una única imagen, sino un "array" de imágenes:

```
protected Imagen[] secuencia;
```

Como no siempre estaremos mostrando el mismo fotograma de esa secuencia, deberemos llevar un contador para saber cuál es el fotograma actual:

```
protected byte fotogramaActual;
```

Además, ahora el método que se encarga de dibujar tendrá que saber si debe mostrar una imagen tomada de este array o no, para lo que nos interesará tener un "booleano" auxiliar:

```
protected bool contieneSecuencia = false;
```

De modo que "DibujarOculta" quedaría así:

```
public void DibujarOculta()
{
     if (!visible) return;

     if (contieneSecuencia)
          secuencia[fotogramaActual].DibujarOculta(x, y);
     else
          imagen.DibujarOculta(x, y);
}
```

Nos falta un constructor que no cargue una imagen, sino que prepare las cosas para cargar una secuencia: (o dentro de poco tiempo, varias):

```
public ElemGrafico(int nuevoAncho, int nuevoAlto)
{
     contieneSecuencia = true;

     visible = true;
     ancho = nuevoAncho;
     alto = nuevoAlto;
}
```

Y el método encargado de cargar las secuencias podría recibir un array de strings, que fueran los nombres de cada una de las imágenes que forman esa secuencia:

```
public void CargarSecuencia(string[] nombres)
{
     if (!contieneSecuencia)
          return;

     byte tamanyo = (byte)nombres.Length;
     secuencia = new Imagen[tamanyo];
     for (byte i = 0; i < nombres.Length; i++)
          secuencia[i] = new Imagen(nombres[i]);
}
```

Sólo nos falta el método encargado de pasar de una imagen a la siguiente dentro de esa secuencia. Lo podemos llamar "SiguienteFotograma":

```
public void SiguienteFotograma()
{
     if (!contieneSecuencia)
          return;

     if (fotogramaActual < secuencia.Length - 1)
          fotogramaActual++;
     else
          fotogramaActual = 0;
}
```

Con eso ya tenemos toda la infraestructura necesaria para que un "elemento gráfico" esté formado por una secuencia de imágenes. Ahora vamos a aplicarlo a la clase "Enemigo":

En primer lugar, el constructor se basará en el de ElemGrafico que no carga imagen, sino que prepara para cargar una secuencia. Además, cargará una secuencia formada por dos foogramas:

```
public Enemigo()
     : base(36, 42)
{
     incrX = 5;
     CargarSecuencia(new string[] {
          "enemigo1.png", "enemigo2.png" });
}
```

Y el método "Mover", además de calcular la siguiente posición del movimiento, cambiará de un fotograma visible al siguiente:

```
public void Mover()
{
     SiguienteFotograma();

     if (x < 50)
          ...
```

Puedes descargar el juego completo, incluyendo fuentes, proyecto, imágenes y ejecutable, para ver los fuentes completos y probar el resultado.

Ejercicio propuesto: Aplica estos cambios al "Personaje", para que también intercambie al menos entre dos fotogramas.

40. Varias animaciones distintas.

Si queremos que un personaje pueda tener distintas secuencias de movimiento, es decir, que no se vean los mismos fotogramas cuando se mueva en una dirección o en otra, podemos ampliar el esquema anterior para que no sea un único array de imágenes.

La primera aproximación podría ser pensar en un array de dos dimensiones, de modo que el primer índice sirva para saber qué secuencia estamos utilizando y el segundo índice sea para el fotograma actual dentro de esa secuencia:

```
protected Imagen[,] secuencia;
```

Pero esto tiene la limitación de que todas las secuencias tendrían que tener la misma cantidad de fotogramas. Una alternativa más versátil es utilizar un "array de arrays", para que las secuencias puedan ser de distinta longitud:

```
protected Imagen[][] secuencia;
```

Además, ahora nos interesará saber en qué dirección se mueve el personaje, para saber cuál debe ser la siguiente imagen a mostrar:

```
protected byte direccion;
```

Para nombrar las direcciones, sería preferible usar "constantes" en vez de números, que ayuden a que el fuente sea más legible:

```
protected byte cantidadDirecciones = 4;
public const byte ABAJO = 0;
public const byte ARRIBA = 1;
public const byte DERECHA = 2;
```

```
public const byte IZQUIERDA = 3;
```

De modo que en el constructor reservaríamos espacio para ese array:

```
secuencia = new Imagen[cantidadDirecciones][];
```

Y "CargarSecuencia" debería tener en cuenta la dirección a la que corresponde esa serie de imágenes:

```
public void CargarSecuencia(byte direcc, string[] nombres)
{
  byte tamanyo = (byte) nombres.Length;
  secuencia[direcc] = new Imagen[tamanyo];
  for (byte i=0; i< nombres.Length; i++)
    secuencia[direcc][i] = new Imagen(nombres[i]);
}
```

En la mayoría de métodos no hay apenas cambios. Basta poner "secuencia[direccion]" donde antes ponía "secuencia". Por ejemplo, en SiguienteFotograma se comprobaría la longitud de la secuencia actual haciendo:

```
public void SiguienteFotograma()
{
     if (!contieneSecuencia)
          return;

     if (fotogramaActual < secuencia[direccion].Length - 1)
          fotogramaActual++;
     else
          fotogramaActual = 0;
}
```

Y nos hará falta un método "CambiarDirección" que pase a tomar las imágenes de una secuencia distinta (y que volver a comenzar desde el fotograma 0, para evitar problemas si una secuencia es más larga que otra):

```
public void CambiarDireccion(byte nuevaDir)
{
     if (!contieneSecuencia)
          return;

     if (direccion != nuevaDir)
     {
          direccion = nuevaDir;
          fotogramaActual = 0;
     }

}
```

Un ejemplo de uso, desde el constructor de la clase Personaje, podría ser:

```
public Personaje()
     : base(32, 23)
{
     x = 400;
     y = 300;
     visible = true;
     incrX = 10;
     incrY = 10;
     CargarSecuencia(DERECHA,new string[] { "personajed1.png", "personajed2.png" });
     CargarSecuencia(IZQUIERDA, new string[] { "personajei1.png", "personajei2.png" });
     CambiarDireccion(DERECHA);
}
```

Y cambiaríamos de dirección cuando fuera necesario. Por ejemplo, al pulsar la tecla Izquierda podría ocurrir lo siguiente:

```
public void MoverIquierda()
```

```
{
     CambiarDireccion(IZQUIERDA);
     SiguienteFotograma();
     x -= incrX;
}
```

Ejercicio propuesto: Aplica estos cambios al juego, para que el personaje realmente intercambie entre dos fotogramas en cada sentido.

41. Una tabla de records.

En casi cualquier juego existe una "tabla de records", que guarda las mejores puntuaciones. Las características habituales son:

- Se guarda una cantidad prefijada de puntuaciones (típicamente 10).
- Para cada puntuación, se guarda también el nombre (o las iniciales) del jugador que la consiguió.
- Las puntuaciones (y sus nombres asociados) deben estar ordenadas, desde la más alta a la más baja.
- Las puntuaciones deben guardarse en fichero automáticamente cuando termina la partida (o incluso antes, cuando se introduzca un nuevo record).
- Las puntuaciones se deben leer de fichero cada vez que el juego se pone en marcha. Cada puntuación nueva que se intenta introducir deberá quedar en su posición correcta, desplazando "hacia abajo" a las que sean inferiores, o incluso no entrando en la tabla de records si se trata de una puntuación más baja que todas las que ya existen en ella.
- Para conseguirlo, podemos crear una clase auxiliar, que también se podrá aplicar a cualquier otro juego. Esa clase debería tener un método para "Cargar" desde fichero (o el constructor se podría encargar), otro para "Grabar" en fichero, otro para "Introducir" un nuevo record (su puntuación y su nombre asociado) y otro para "Obtener" un record (tanto la puntuación como el nombre, de modo que las podamos mostrar en pantalla).

Una forma sencilla de conseguirlo, usando simplemente un array para los puntos y otro distinto para los nombres, en vez de un array de struct o de objetos, podría ser ésta:

```
using System;
using System.IO;

public class TablaRecords
{

     const int NUM_RECORDS = 10;
     int[] puntos;
     string[] nombres;

     public TablaRecords()
     {
          puntos = new int[NUM_RECORDS];
          nombres = new string[NUM_RECORDS];
          Cargar();
     }

     public void Cargar()
     {
          if (!File.Exists("records.dat"))
               return;

          StreamReader fichero = File.OpenText("records.dat");

          for (int i = 0; i < NUM_RECORDS; i++)
          {
               nombres[i] = fichero.ReadLine();
               string puntuacion = fichero.ReadLine();
               if (puntuacion != null)
                    puntos[i] = Convert.ToInt32(puntuacion);
          }
          fichero.Close();
     }
```

```
    public void Grabar()
    {
        StreamWriter fichero = File.CreateText("records.dat");
        for (int i = 0; i < NUM_RECORDS; i++)
        {
            fichero.WriteLine(nombres[i]);
            fichero.WriteLine(puntos[i]);
        }
        fichero.Close();
    }

    public int GetNumRecords()
    {
        return NUM_RECORDS;
    }

    public void Introducir(int puntuacion, string nombre)
    {
        // Compruebo "si cabe"
        if (puntos[NUM_RECORDS - 1] > puntuacion)
            return;

        // Busco la posición correcta
        int posicion = NUM_RECORDS - 1;
        while ((posicion > 0) && (puntuacion > puntos[posicion-1]))
        {
            posicion--;
        }
        // Inserto en esa posición (desplazando siguientes)
        for (int i = NUM_RECORDS - 1; i > posicion; i--)
        {
            puntos[i] = puntos[i - 1];
            nombres[i] = nombres[i - 1];
        }
        puntos[posicion] = puntuacion;
        nombres[posicion] = nombre;

        // Guardar cambios
        Grabar();
    }

    public void Obtener(int posicion, out int puntuacion,
                        out string nombre)
    {
        puntuacion = puntos[posicion - 1];
        nombre = nombres[posicion - 1];
    }
}
```

Y para probarla, lo razonable sería hacerlo "desde fuera del juego", con un programa específico, incluso en modo texto, que fuera añadiendo datos y tomándolos de la tabla de records, para comprobar de la forma más simple y a la vez más exhaustiva que se comporta correctamente:

```
using System;

public class PruebaTablaRecords
{

    public static void Main()
```

```
    {

        TablaRecords t = new TablaRecords();

        t.Introducir(200, "Juan");
        t.Introducir(100, "Pedro");
        t.Grabar();
        t.Introducir(150, "Alberto");
        t.Cargar();

        int maximo = t.GetNumRecords();
        for (int i = 1; i <= maximo; i++)
        {
            string nombre;
            int puntos;
            t.Obtener(i, out puntos, out nombre);
            Console.WriteLine("{0}: {1} = {2}",
                              i, nombre, puntos);
        }

    }
}
```

Las llamadas a "Cargar" y "Grabar" desde el programa de prueba son sólo para asegurarnos de que no "pasan cosas raras". En una implementación real, una vez que todo esté probado, sería razonable que tanto "Cargar" como "Grabar" fueran métodos privados, sólo para uso de la propia clase "TablaRecords" y no desde fuera de ella.

Ejercicio propuesto: Incluye esta clase en el juego. Añade cada record a la tabla al final de una partida (con un nombre prefijado). Crea una pantalla similar a la de "Créditos" para mostrar el contenido de la tabla de records.

42. Leer niveles desde fichero.

Hasta ahora, la descripción de los niveles que recorreremos en el juego se hace mediante un array, prefijado en el código fuente:

```
public byte[,] fondo =
{
    {
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
    },
    {
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
```

```
        {1,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1},
        ...
```

Ahora vamos a llevarlo a un fichero, como forma de que el juego se pueda ampliar sin necesidad de tocar el código. El primer paso será volcar a un fichero el contenido de este array. Para ello, tenemos que escoger un formato. Por ejemplo, podríamos usar un fichero de texto, en el que cada bloque de varias líneas represente un nivel: una cantidad predefinida de líneas, cada una con una cantidad de letras prefijada. Para mayor legibilidad, podemos incluir una línea en blanco entre un nivel y otro. Y para que nos sea sencillo de manipular, podemos incluir una primera línea que nos diga la cantidad de niveles que contiene el fichero:

```
3

111111111111111111111111
100000000000010000000001
100000000000010000000001
100000000000010000000001
100000000000010000000001
100000000000010000000001
100000000000000000000001
100000000000000000000001
100000000000000000000001
100000000000000000000001
100000000000000000000001
100000000000111111110001
100000000000000000000001
100000000000000000000001
100000000000000000000001
100000000000000000000001
111111111111111111111111

111111111111111111111111
100000000000010000000001
100000000000010000000001
100001000000010000000001
...
```

Ahora podemos crear una función encargada de leer ese fichero, y rellenar con él los datos del array:

```
private void leerDesdeFichero()
{
     StreamReader ficheroMapas = File.OpenText("niveles.dat");
     string linea;

     linea = ficheroMapas.ReadLine();
     int numeroNiveles = Convert.ToInt32(linea);

     fondo = new byte[numeroNiveles, altoFondo, anchoFondo];

     for (int nivelActual = 0; nivelActual < numeroNiveles; nivelActual++)
     {
          string separador = ficheroMapas.ReadLine();

          for (int lineaActual = 0; lineaActual < altoFondo; lineaActual++)
          {
               linea = ficheroMapas.ReadLine();
               for (int columnaActual = 0; columnaActual < anchoFondo; columnaActual++)
                    fondo [nivelActual, lineaActual, columnaActual] =
                         Convert.ToByte(linea[columnaActual]-48);
          }
     }
     ficheroMapas.Close();
}
```

Sólo queda un cambio por hacer: dejar el array vacío y hacer que el constructor comience leyendo los datos desde fichero:

```
public byte[, ,] fondo;

public Nivel()
{
     leerDesdeFichero();

     numPremios = 20;
     ...
```

Si nos molesta eso de que cualquiera pueda modificar el fichero, hay una alternativa sencilla: encriptarlo. El método "leerDesdeFichero" debería ir desencriptando los datos a medida que los lee, y podríamos crear una utilidad adicional que tomase un fichero "legible" (creado con cualquier editor de texto) y lo encriptara, para dificultar su modificación por terceras personas.

Podemos aprovechar para que el fichero guarde información extra, como la posición inicial de nuestro personaje, o la cantidad de enemigos y sus posiciones. Pero eso lo haremos un poco más adelante...

Ejercicio propuesto: Amplía el juego con dos niveles más, hasta llegar a un total de cinco, y comprueba que se comportan correctamente.

43. Añadiendo gravedad.

Si nuestro personaje está "expuesto a las leyes de la gravedad", debería "caer" cuando no haya suelo bajo sus pies...

Aparentemente, la acción de "caer" se parecerá mucho a la de "mover hacia abajo" de nuestro personaje, pero hay un par de diferencias, una de las cuales es poco importante y otra que lo es bastante más:

- Por una parte, es habitual que el personaje "cambie de fotograma" cuando está andando, pero no cuando cae.
- Por otra parte (y esto sí es importante), el personaje baja cuando pulsamos cierta tecla (o movemos el joystick/gamepad), pero si está cayendo, debe seguir haciéndolo aunque nosotros no hagamos nada.

Por lo tanto, podríamos crear un método "caer" que hiciera que el personaje bajase pero sin cambiar de fotograma:

```
public void Caer()
{
     y += incrY;
}
```

Pero este método podemos necesitar llamarlo de forma repetitiva. De igual modo que el "enemigo" tenía un método "mover" que se encargaba de su movimiento automático, vamos a crear un "mover" que gestione los movimientos que tenga que hacer nuestro personaje incluso cuando nosotros no toquemos ninguna tecla: caer y, dentro de poco, saltar.

```
public void Mover()
{
     if (miPartida.EsPosibleMover(x,y+incrY,
               x+ancho,y+alto+incrY))
          Caer();
}
```

Este "mover" deberá ser llamado desde la "partida", en concreto desde la función encargada de "mover elementos":

```
public void MoverElementos()
{
     // -- Mover enemigos, entorno --
     for (int i = 0; i < miNivel.GetNumEnemigos(); i++)
          miNivel.GetEnemigo(i).Mover();

     miPersonaje.Mover();
}
```

Y ya sólo quedan pequeños retoques como ese "EsPosibleMover" de partida, para que el personaje (o incluso los enemigos) puedan saber si es posible desplazarse a cierta posición o hay algún obstáculo que lo impide).

¿Y de dónde sale ese "miPartida" que aparece en "Personaje"? Como la partida contiene al personaje, puede acceder a sus datos y sus métodos (que sean públicos), pero no al contrario: el personaje no sabe "a qué partida pertenece". Podemos

solucionarlo creando un atributo "miPartida", al que se dará valor con una función "IndicarPartida" del personaje o bien desde su constructor:

```
public class Personaje : ElemGrafico
{
     private Partida miPartida;

     public Personaje(Partida p)
          : base(4, 32, 23)  // Elemento gráfico con 4 direcciones
     {
          miPartida = p;
          ...
```

Y la partida le daría valor inicial cuando crea al personaje:

```
public Partida()
{
     miMarcador = new Marcador();
     miPersonaje = new Personaje(this);
     miNivel = new Nivel();
}
```

Como hemos hecho varias ampliaciones de una cierta complejidad, puedes descargar una versión actualizada del juego completo, incluyendo fuentes, proyecto, imágenes y ejecutable, para ver la apariencia de los fuentes completos y poder probar el resultado con facilidad.

Ejercicio propuesto: Mejora esta estructura, para que el personaje no se pueda mover hacia los lados mientras cae.

44. Un personaje que salta en vertical.

Hacer que el personaje puede "saltar" supone varios cambios:

- Por una parte, cuando haya comenzado a saltar deberá moverse solo, al igual que ocurría cuando caía, lo que implica que necesitaremos un método "Saltar", que comience la secuencia del salto, pero también usaremos el método "Mover", que se encargará de continuar esa secuencia (cuando corresponda).
- Además, la secuencia de salto será más complicada que (por ejemplo) el movimiento "normal" del personaje o que su caída: el salto (hacia los lados)suele ser un movimiento parabólico. Además, en muchos juegos, el personaje salta en vertical si sólo se pulsa la tecla de saltar, y lo hace de forma parabólica a derecha o izquierda si se pulsa la tecla de salto junto con una de las teclas de dirección.

Una primera forma de conseguirlo sería usar realmente la ecuación de la parábola, pero suele ser más eficiente y más versátil "precalcular" la secuencia de movimientos y guardarlos en un array. Realmente, ni siquiera es necesario hacer "con cálculos reales", sino que podemos imitar la aceleración de la gravedad, preparando movimientos más rápidos (más "largos") al principio y al final de la parábola, y movimientos más lentos en el punto medio, algo como:

```
int[] pasosSaltoArriba = {
     -8, -8, -8, -6, -6, -5, -5, -4, -4, -4, -3, -3, -2, -1, -1, 0, 0, 0,
     0, 0, 0, 1, 1, 2, 3, 3, 4, 4, 4, 5, 5, 6, 6, 8, 8, 8 };
```

Esa es la secuencia de salto casi completa: falta tener en cuenta que cuando el personaje deja de avanzar en forma parabólica, ya sea porque realmente llegue a su destino o porque choque un obstáculo, deberíamos comprobar si tiene suelo por debajo, para que caiga en caso contrario.

En este primer acercamiento, nos centraremos en que el personaje salte (y caiga, claro) en vertical, y en el próximo apartado lo ampliaremos para que también salte hacia los lados.

Vamos con ello...

En primer lugar, necesitaremos un método "Saltar" dentro del personaje, que pondrá en marcha la secuencia de salto:

```
// Hace que el personaje comience a saltar
public void Saltar()
{
          saltando = true;
}
```

(Obviamente, ese "saltando" será un atributo privado del Personaje)

Por otra parte, dentro de la "Partida", habrá que comprobar si se pulsa una cierta tecla (por ejemplo, Espacio) para llamar a ese método:

```
public void ComprobarTeclas()
{
        ...

        if (Hardware.TeclaPulsada(Hardware.TECLA_ESP) || Hardware.JoystickPulsado(0) )
                miPersonaje.Saltar();

}
```

Además, el método "Mover" del personaje, tendrá que comprobar si está saltando, y hacer algo en ese caso. Una primera aproximación podría ser disminuir en una unidad la posición "y" del personaje, de modo que cuando pulsemos Espacio, empezaría a moverse lentamente hacia arriba hasta que saliera de la pantalla para nunca más volver...

```
public void Mover()
{
        if (saltando)
                y--;

        else
                if (miPartida.EsPosibleMover(x,y+incrY,
                                x+ancho,y+alto+incrY))
                        Caer();
}
```

Por supuesto, lo anterior no es cierto del todo: en cuanto pulsemos ESC, termina la partida y podemos volver a empezar... o incluso salir del juego para hacer que el salto sea un poco más real.

En cuanto hayamos comprobado que todo funciona hasta aquí, podemos utilizar un array con lo que se tiene que desplazar en cada "fotograma" de la secuencia de salto, como ya habíamos comentado más arriba. Nos vendrá bien tener también un contador que nos permitar saber en cual de todos esos "pasos" nos encontramos actualmente:

```
// Atributos
bool saltando;
int[] pasosSalto =
        { -10, -10, -8, -8, -6, -6, -4, -2, -1, -1, 0,
          0, 1, 1, 2, 4, 6, 6, 8, 8, 10, 10  };
int pasoSaltoActual;
```

De modo que un "Mover" más real debería tomar el siguiente de esos "saltos" para aplicárselo al personaje, reiniciar toda la secuencia cuando ha terminado, y, a partir de entonces (pero no antes) comprobar si debe caer:

```
public void Mover()
{
        if (saltando)
        {
                // Siguiente incremento del salto
                y += pasosSalto[pasoSaltoActual];
                // Si no ha terminado la secuencia
                if (pasoSaltoActual < pasosSalto.Length-1)
                        pasoSaltoActual++; // preparo el siguiente
                // Y si ha terminado, reinicio
                else
                {
                        pasoSaltoActual = 0;
                        saltando = false;
                }
        }
        else
                if (miPartida.EsPosibleMover(x, y + incrY,
                                x + ancho, y + alto + incrY))
                        Caer();
}
```

Ejercicio propuesto: Aplica todos estos cambios. Comprueba qué ocurre si se pulsa la tecla hacia arriba mientras el personaje está saltando... y soluciónalo.

45. Salto hacia los lados.

Ahora mismo, nuestro personaje puede saltar. Además, si mantenemos pulsada una flecha hacia un lado mientras saltamos, el personaje saltará hacia ese lado.

Pero eso tiene algunos efectos laterales no deseables:

- Podemos movernos hacia un lado y hacia otro mientras salta, consiguiendo que el personaje cambie de dirección en el aire, algo que resulta "poco real" y que la mayoría de juegos no permite.
- Si pulsamos la flecha arriba, podemos encadenar varios saltos, y así alcanzar alturas mucho mayores.

Esos efectos son fáciles de evitar. Los métodos "MoverArriba" y similares del personaje deberán comprobar primero si está saltando, para no permitir que se mueva en ese caso:

```
public void MoverArriba()
{
     if (saltando)
          return;

     SiguienteFotograma();
     y -= incrY;
}
```

Pero querremos permitir que sí se pueda saltar hacia el lado, de forma más controlada. Para eso, podemos crear un método "SaltarDerecha" y otro "SaltarIzquierda", que añadan un movimiento horizontal al salto:

```
public void SaltarDerecha()
{
     saltando = true;
     incrXsalto = 4;
}
```

Mientras que en el "Saltar" (en vertical) habrá que dejar claro que no habrá movimiento horizontal:

```
public void Saltar()
{
     saltando = true;
     incrXsalto = 0;
}
```

Y si queremos que se pueda saltar a derecha e izquierda, deberemos comprobar desde "Partida" si se pulsan simultáneamente las teclas de salto y las de desplazamiento:

```
public void ComprobarTeclas()
{
     // -- Leer teclas y calcular nueva posición --
     if (Hardware.TeclaPulsada(Hardware.TECLA_ESC))
          partidaTerminada = true;

     if (((Hardware.TeclaPulsada(Hardware.TECLA_DER)
               && Hardware.TeclaPulsada(Hardware.TECLA_ESP)))
          || (Hardware.JoystickDerecha() && Hardware.JoystickPulsado(0)))
          miPersonaje.SaltarDerecha();

     else if (((Hardware.TeclaPulsada(Hardware.TECLA_IZQ)
               && Hardware.TeclaPulsada(Hardware.TECLA_ESP)))
          || (Hardware.JoystickIzquierda() && Hardware.JoystickPulsado(0)))
          miPersonaje.SaltarIzquierda();

     ...
```

(Eso habrá que hacerlo antes de comprobar cada una de las teclas individuales).

Si quieres contrastar tu versión con la "oficial", puedes descargar todo el juego, con fuentes y ejecutable.

Ejercicio propuesto: Rediseña los niveles, de modo que se puedan recorrer saltando.

46. Premios como parte del mapa.

Ahora que nuestro personaje no se mueve libremente por el mapa, sino que tiene que saltar, ya no nos podemos permitir que los premios aparezcan al azar, porque quizá no sean alcanzables.

Una solución sencilla suele ser incluirlos como parte del mapa de fondo, para que su posición esté prefijada. Cuando nuestro personaje toque uno de ellos, podremos eliminarlos del mapa para que no se sigan dibujando.

El mapa podría ser algo como esto:

```
111111111111111111111111
1            1          1
1            1          1
1       0    1          1
1            1          1
1            2          1
1                  X    1
1                       1
1                111    1
1   E           21 0    1
1                       1
1           22111111    1
1                       1
1        111            1
1   P            0      1
1      21               1
111111111111111111111111
```

Por ejemplo, en ese mapa, 1 y 2 podrían ser distintos tipos de "ladrillos" de fondo, X podrían ser los premios que hay que recoger, O serían obstáculos mortales, E podría ser la posición inicial de los enemigos, e incluso P podría ser la posición inicial del personaje (que no usaremos todavía en esta entrega).

Los cambios en el fuente serán menos grandes de los que esperamos: ya teníamos un array de elementos de fondo, otro de premios, otro de enemigos... La única diferencia de planteamiento es que antes algunos de ellos tenían tramaño prefijado, mientras que ahora se tendrán que rellenar "al vuelo" a partir de lo que se ha detallado en el mapa.

Como los arrays tienen tamaño prefijado, el primer paso es saber cuál debe ser ese tamaño, para poder reservar espacio, por lo que daremos una primera pasada (en la próxima versión usaremos ArrayList, para evitar esta pasada inicial):

```
for (int fila = 0; fila < altoFondo; fila++)
     for (int col = 0; col < anchoFondo; col++)
     {
          // Elementos del fondo: 1 al 9
          if ((fondo[nivel, fila, col] >= '1') && (fondo[nivel, fila, col] <= '9'))
               elementosFondo++;
          // Premios: X
          if (fondo[nivel, fila, col] == 'X')
               numPremios++;
          // Enemigos: E
          if (fondo[nivel, fila, col] == 'E')
               numEnemigos++;
          // Obstáculos: O
          if (fondo[nivel, fila, col] == 'O')
               numObstaculos++;
          ...
```

Y en una segunda pasada, tras reservar espacio, rellenaremos los arrays:

```
// Y reservo espacio para el array que los contiene
fondos = new ElemGrafico[elementosFondo];
obstaculos = new Obstaculo[numObstaculos];
enemigos = new Enemigo[numEnemigos];
```

```
premios = new Premio[numPremios];

int posicFondo = 0, contEnemigos = 0, contObstaculos = 0, contPremios = 0;
for (int fila = 0; fila < altoFondo; fila++)  // Fondo
     for (int col = 0; col < anchoFondo; col++)
          switch (fondo[nivel, fila, col])
          {
               case '1':
                    fondos[posicFondo] = new ElemGrafico("pared.png", anchoCasillaFondo, altoCasillaFondo);
                    fondos[posicFondo].MoverA(margenXFondo + col * anchoCasillaFondo,
                         margenYFondo + fila * altoCasillaFondo);
                    fondos[posicFondo].SetVisible(true);
                    posicFondo++;
                    break;
               case '3':
                    fondos[posicFondo] = new ElemGrafico("pared3.png", anchoCasillaFondo, altoCasillaFondo);
                    fondos[posicFondo].MoverA(margenXFondo + col * anchoCasillaFondo,
                         margenYFondo + fila * altoCasillaFondo);
                    fondos[posicFondo].SetVisible(true);
                    posicFondo++;
                    break;
               case 'X':
                    premios[contPremios] = new Premio();
                    premios[contPremios].MoverA(margenXFondo + col * anchoCasillaFondo,
                         margenYFondo + fila * altoCasillaFondo);
                    premios[contPremios].SetVisible(true);
                    contPremios++;
                    break;
          ...
```

Y a la hora de reiniciar, ya no habrá que colocarlos en posiciones al azar, sino volver a tomar los datos que preveía el mapa:

```
public void Reiniciar()
{
     nivel = 0;
     RellenarArrayDeFondo();
}
```

Las rutinas de dibujado, de comprobación de colisiones, etc. no deberían necesitar cambios.

Si "te atascas" intentando implementar estos cambios, aquí tienes la "versión oficial", con fuentes y ejecutable.

Ejercicio propuesto: Usa "ArrayList" en vez de arrays, para no tener que dar dos pasadas.

47. Mejorando con estructuras dinámicas.

Nuestro mapa ya tiene información no sólo sobre el fondo, sino también sobre premio, obstáculos mortales, enemigos... pero como usamos "arrays", que deben tener tamaño prefijado, necesitamos dar dos pasadas para rellenarlos: en la primera contamos elementos y en la segunda los almacenamos.

Una alternativa más razonable es usar estructuras dinámicas, que crezcan automáticamente, e ir incluyendo en ellas todos estos elementos. Por ejemplo, en el caso del lenguaje C#, podemos usar ArrayList.

A la hora de rellenar, los cambios no serían grandes: usar Add para añadir en vez de corchetes (y eliminamos la primera pasada para contar, claro):

```
public void RellenarArrayDeFondo()
{
     // Y reservo espacio para el array que los contiene
     fondos = new ArrayList();
     obstaculos = new ArrayList();
     enemigos = new ArrayList();
     premios = new ArrayList();

     int posicFondo = 0, contEnemigos = 0, contObstaculos = 0, contPremios = 0;
```

```
for (int fila = 0; fila < altoFondo; fila++)  // Fondo
     for (int col = 0; col < anchoFondo; col++)
          switch (fondo[nivel, fila, col])
          {
               case '1':
                    fondos.Add(new ElemGrafico("pared.png", anchoCasillaFondo, altoCasillaFondo));
                    ((ElemGrafico)fondos[posicFondo]).MoverA(margenXFondo + col * anchoCasillaFondo,
                         margenYFondo + fila * altoCasillaFondo);
                    ((ElemGrafico)fondos[posicFondo]).SetVisible(true);
                    posicFondo++;
                    break;
               ...
```

Y al dibujar, comprobar colisiones, etc., los cambios también son mínimos: en vez de recorrer todas esas estructuras hasta el contador que teníamos antes, ahora lo haremos hasta su tamaño real (que podemos saber con "fondos.Count", "premios.Count", etc):

```
public void DibujarOculta()
{
     for (int i = 0; i < fondos.Count; i++)  // Imágenes del fondo
          ((ElemGrafico)fondos[i]).DibujarOculta();
     ...
```

Son cambios sencillos, pero un así, aquí tienes todo el proyecto, incluyendo fuentes y ejecutable.

Ejercicio propuesto: Amplía este esqueleto, para que tenga más tipos distintos de casillas de fondo.

48. Segundo juego completo en modo gráfico.

Llega el momento de juntar todo lo que hemos visto, para crear un "juego de plataformas clásico", en el que el usuario tenga que recoger premios, esquivar enemigos y avanzar de pantalla...

Este apartado estará disponible dentro de poco tiempo (fecha prevista: 17 de marzo).

Cambios entre versiones

- 0.47, de 14-Mar-2012: Añadido el apartado 47.
- 0.46, de 14-Mar-2012: Añadido el apartado 46.
- 0.45, de 02-Mar-2012: Añadido el apartado 45. Cambiados los tipos de letra, usando Google WebFonts, para que sea un poco más atractivo y los fuentes un poco más legibles (se distingan mejor los ceros de la letra "o", por ejemplo).
- 0.44, de 28-Feb-2012: Añadido el apartado 44.
- 0.43, de 24-Feb-2012: Añadido el apartado 43.
- 0.42, de 23-Feb-2012: Añadido el apartado 42.
- 0.41, de 22-Feb-2012: Añadido el apartado 41.
- 0.40, de 21-Feb-2012: Añadido el apartado 40.
- 0.39, de 20-Feb-2012: Añadido el apartado 39. Cambiada la numeración de los próximos apartados previstos.
- 0.38, de 04-Feb-2012: Añadido el apartado 38 y el planteamiento del 39, el 40, el 41 y el 42.
- 0.37, de 03-Feb-2012: Añadido el apartado 37 y el planteamiento del 38.
- 0.36, de 02-Feb-2012: Añadido el apartado 36.
- 0.35, de 01-Feb-2012: Añadido el apartado 35.
- 0.34, de 31-Ene-2012: Añadido el apartado 34. No se muestra el panel lateral de acceso rápido, para no quitar mucha superficie visible al texto del curso.
- 0.33, de 30-Ene-2012: Añadido el apartado 33.
- 0.32, de 29-Ene-2012: Añadido el apartado 32.
- 0.31, de 28-Ene-2012: Añadido el apartado 31.
- 0.30, de 22-Dic-2011: Añadido el fuente correspondiente al apartado 30.
- 0.29, de 08-Dic-2011: Añadidos dos fuentes alternativos (y su explicación) para el apartado 29.
- 0.28, de 03-Dic-2011: Añadido el fuente del apartado apartado 27 y el planteamiento de los apartados 29 y 30.
- 0.27, de 30-Nov-2011: Añadido el apartado 28 y la mayor parte del tema 27. Optimizado el tamaño (peso) de las imágenes de los apartados 25 y 26, para que carguen más rápido. Creada una nueva versión PDF (0.27).
- 0.26, de 29-Nov-2011: Completado el apartado 24.
- 0.25, de 28-Nov-2011: Completado el apartado 23.
- 0.24, de 26-Nov-2011: Completado el apartado 22.

- 0.23, de 23-Nov-2011: Añadidas las imágenes del apartado 26, y el planteamiento de los apartados 22 y 24.
- 0.22, de 22-Nov-2011: Añadido el apartado 25 y parte del texto (todavía sin imágenes) del apartado 26.
- 0.21, de 19-Nov-2011: Añadido el apartado 21, junto con el correspondiente proyecto para Visual Studio 2010, para MonoDevelop 2.4 o para compilar desde línea de comandos.
- 0.20, de 18-Nov-2011: Añadido el apartado 20, ampliado el apartado 19 con una variante del juego que usa "StringBuilder" y "foreach", creada una nueva versión PDF (0.19).
- 0.19, de 11-Nov-2011: Añadido un apartado 19 que no estaba previsto (juego del ahorcado en modo texto), renumerados los posteriores, ampliada la planificación hasta el apartado 35.
- 0.18, de 09-Nov-2011: Añadido el apartado 18 y el planteamiento de los apartados 21, 22 y 23.
- 0.17, de 06-Nov-2011: Añadido el apartado 17 y el planteamiento de los apartados 19 y 20. Incluidas imágenes en los apartados 17, 13 y 6.
- 0.16, de 04-Nov-2011: Añadido el apartado 16 y el planteamiento de los apartados 17 y 18
- 0.15, de 03-Nov-2011: Añadido el apartado 15
- 0.14, de 30-Oct-2011: Añadido el apartado 14, creada una nueva versión PDF (0.14)
- 0.13, de 28-Oct-2011: Añadido el apartado 13 y el planteamiento de los apartados 14 y 15
- 0.12, de 24-Oct-2011: Añadido el apartado 12 y el planteamiento del apartado 13
- 0.11, de 23-Oct-2011: Añadido el apartado 11
- 0.10, de 23-Oct-2011: Añadido el apartado 10
- 0.09, de 22-Oct-2011: Añadido el apartado 9
- 0.08, de 21-Oct-2011: Añadido el apartado 8, creada una primera versión PDF (0.07)
- 0.07, de 21-Oct-2011: Añadido el apartado 7, creado el apartado de Herramientas
- 0.06, de 21-Oct-2011: Añadido el apartado 6
- 0.05, de 20-Oct-2011: Añadido el apartado 5
- 0.04, de 20-Oct-2011: Añadido el apartado 4
- 0.03, de 20-Oct-2011: Añadido el apartado 3
- 0.02, de 19-Oct-2011: Añadido el apartado 2
- 0.01, de 18-Oct-2011: Creada la estructura básica, incluido el apartado 1