

Quiero ser como Will (Wright)

29 de enero de 2008

Escrito por Miguel Santirso González

<http://miguelsantirso.es>

Capítulo 3: Colisiones

Al ver que ya estábamos llegando, me desabroché el cinturón y cogí mi abrigo del asiento de atrás del coche.

—Bueno tío —me dijo Rey mientras aparcaba el coche al lado del portal de Olaya—, ya sabes, siempre con protección. Acuérdate de lo que nos explicaron las monjas en el colegio —continuó, en un tono claramente sarcástico.

—No te preocupes por eso —le respondí riéndome—. Solo quedamos para seguir con lo del juego, esto no significa nada más.

Era mentira, claro. Ella me gustaba, mucho, y tenía más o menos claro que yo a ella también le gustaba (estaba seguro de que, como mínimo, le caía muy bien). Todo eso hacía que el hecho de ir a su casa por primera vez significara mucho más que una simple clase de desarrollo de videojuegos.

Crucé la calle y enseguida me planté delante de su portal. Espere unos segundos mientras respiraba hondo para tranquilizarme y pulsé el timbre durante el tiempo suficiente para asegurarme de que lo oía, aunque sin pasarme, no fuera a creer que estaba demasiado emocionado. Enseguida, alguien respondió:

—¿Quién es? —respondió una profunda voz masculina.

Eso me dejó bastante descolocado. Según me había dicho, sus padres se iban a ir durante el fin de semana, pero aún así conseguí encadenar unas cuantas palabras para preguntar por ella:

—Hola soy Miguel, ¿está Olaya?

—¿Eh? Ah, no, está equivocado. Ella vive en el 4ºD; esto es el 5ºD —respondió, para colgar justo a continuación.

Joder. Vaya susto. Después de maldecir a mi despistado cerebro y de volver a respirar hondo un par de veces, piqué de nuevo, aunque asegurándome de que tocaba el botón correcto.

—¿Sí? —respondió la alegre, e inconfundible, voz de Olaya.

—Hola —respondí aliviado—, soy yo. ¿Me abres?

Me abrió la puerta, y después de subir cuatro pisos de escaleras (el ascensor estaba ocupado por una de esas personas que cree poseerlo en exclusiva) llegué a su puerta de casa. Toqué el timbre y esperé. Al poco rato, se abrió la puerta:

—Ah, hola. Tú debes ser el amigo friki de Olaya —balbuceó el homínido con cara, cuerpo y mente de gorila que me abrió la puerta.

Antes de que me diera tiempo a responder alguna insensatez, apareció Olaya por detrás, le pasó un brazo por el hombro y me saludó:

—¡Hola Miguel! Te presento a mi novio, Carlos —dijo, como si nada.

En ese momento, una riada de extrañas emociones comenzó a bloquear cualquier posibilidad de pensar, de moverme o, desde luego, de hablar. Ante todo esto, la primera reacción de mi cuerpo fue intentar desmayarse, aunque finalmente conseguí dominar la situación y retomé el control de mi cuerpo. Por suerte, todo esto duró menos de un inapreciable segundo.

—Hola Carlos, encantado —dije, al tiempo que le extendía la mano e intentaba sonreír de alguna forma.

Me estrechó la mano con demasiada fuerza (o quizás yo con demasiada poca) y me invitó a entrar en casa. Olaya me dio dos besos y me acompañó hasta su habitación. Al llegar, me senté en una de las dos sillas que había delante del ordenador e intenté calmarme mientras ellos dos hablaban.

—Bueno, Oli. Yo voy a dar una vuelta mientras os quedáis con vuestras frikadas —dijo el tal Carlos.

—Vale —respondió ella—. Supongo que nos llevará toda la tarde... Tráete algo para cenar, ¿vale?

—Vale... ¿Te quieres quedar a cenar con nosotros después? —me preguntó.

—Ehm —hice como que lo pensaba un instante—. No, mejor no. Ya quedé para salir después y no quiero llegar tarde.

—De acuerdo —respondió—. ¡Hasta luego, Oli!

Se dieron un beso para despedirse y Olaya y yo nos quedamos solos. Yo estaba mirando las fotos de los dos que había por allí. Estaba empezando a sentirme realmente mal.

—¡Bueno, Miguel! ¿Qué te parece? ¿Es majo, no?

—Me parece fantástico —dije sin ningún entusiasmo—. Se os ve muy felices.

—Sí, la verdad es que sí... ¡Pero bueno, venga! No perdamos el tiempo, ¡vamos a programar un videojuego!

Dios, la verdad es que era lo último que quería hacer, ahora lo único que me apetecía hacer era sentarme solo en algún sitio, poner cara de pena y dejar pasar el tiempo. Aún así, no podía, debía intentar disimularlo, y eso hacía que me sintiera aún peor. Le pedí que abriera el Visual Studio:

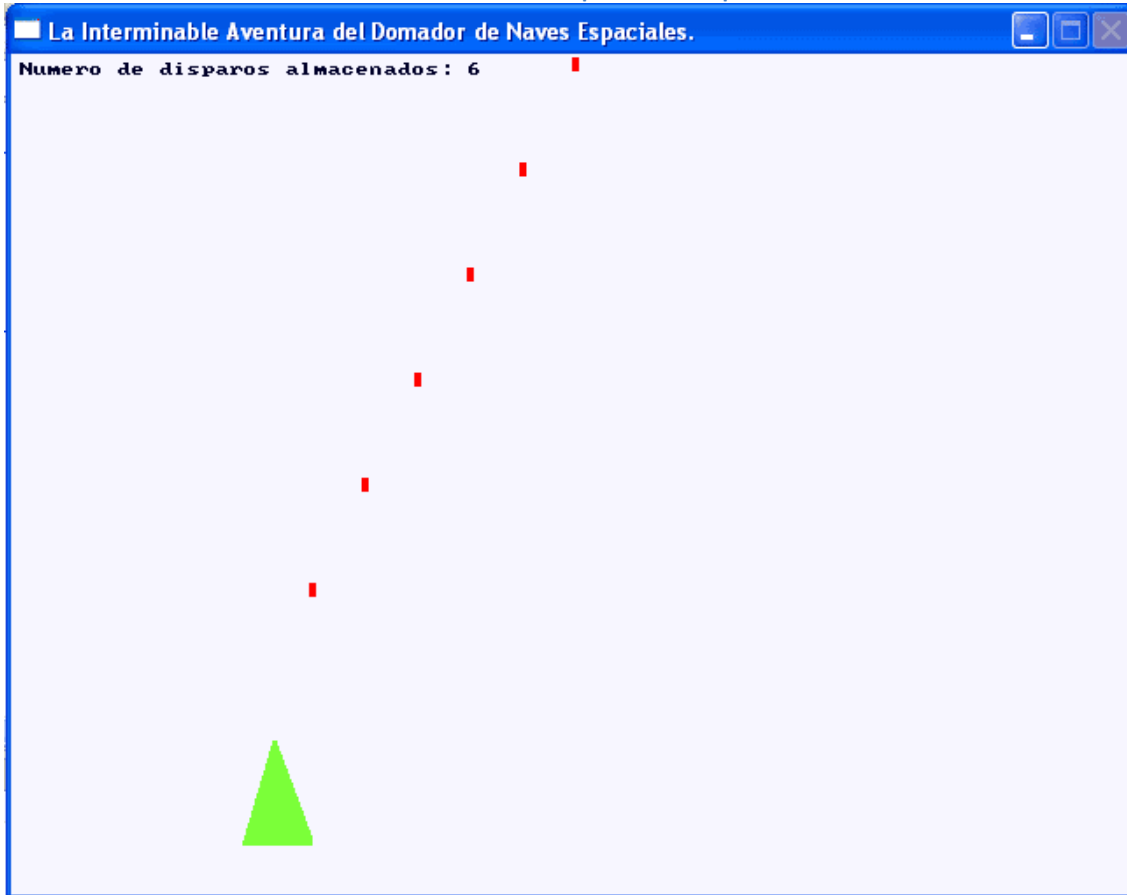
—Bueno, supongo que ya lo tendrás todo preparado... Abre el Visual Studio, a ver si corrigiste aquello que dejamos sin hacer el último día —me refería a hacer que los disparos no salieran tan seguidos como en la versión del último día.

—Sí, sí que lo hice. No fue demasiado difícil.

—Bien —respondí—. A ver, enséñame como lo hiciste.

Al decírselo, ejecutó primero el juego para enseñarme que efectivamente funcionaba.

Ilustración 1: Ahora los disparos salen espaciados.



—Sí, parece que funciona —le dije—. A ver, enséñame el código.

—Vale —respondió, obediente—. Mira, solo tuve que cambiar un poco la clase Jugador. Simplemente le añadí una variable entera para contar cuantos frames pasan entre un disparo y otro.

Abrió los archivos Jugador.cpp y Jugador.h mientras me lo decía y enseguida me enseñó los cambios:

Ilustración 2: Declaración de la nueva variable en Jugador.h

```
protected:
    BITMAP * Grafico_Nave; // Almacenará el gráfico de la nave.

    int Pos_X; // Posición de la nave en el eje horizontal
    int Pos_Y; // Posición de la nave en el eje vertical

    int Velocidad; // Velocidad de movimiento de la nave

    int Temporizador_Disparo; // Temporizador para separar los disparos de la nave.
};
```

Ilustración 3: Inicializar la nueva variable en el constructor de Jugador y definir la separación

```
Jugador.h / Jugador.cpp main.cpp
Jugador Actualiza()

extern BITMAP * Grafico_Disparo;
extern list<Disparo *> Disparos;

#define SEPARACION_DISPARIOS int(10)

Jugador::Jugador(void)
{
}

Jugador::Jugador(BITMAP * _Grafico) : Grafico_Nave(_Grafico), Velocidad(3), Temporizador_Disparo(0)
{
    // Situamos la nave centrada en x y con 25 pixeles de margen desde abajo
    Pos_X = Resolucion_X/2 - Grafico_Nave->w/2; // Grafico_Nave->w = ancho del gráfico
    Pos_Y = Resolucion_Y - Grafico_Nave->h - 25; // Grafico_Nave->h = altura del gráfico
}
```

Ilustración 4: Cambios en la función Actualiza() de Jugador.cpp

```
Jugador.h / Jugador.cpp main.cpp
Jugador Actualiza()

// Dibujamos en la pantalla el objeto.
draw_sprite(Objetivo, Grafico_Nave, Pos_X, Pos_Y);
}

void Jugador::Actualiza()
{
    if(key[KEY_SPACE] && Temporizador_Disparo == 0)
    {
        Temporizador_Disparo = SEPARACION_DISPARIOS;
        Disparos.push_back(new Disparo(
            Grafico_Disparo,
            Pos_X + Grafico_Nave->w/2,
            Pos_Y - Grafico_Nave->h/2));
    }

    if(Temporizador_Disparo > 0)
        Temporizador_Disparo--;

    // Mover la nave en función de las teclas pulsadas y de la velocidad
    if(key[KEY_LEFT])
        Pos_X -= Velocidad;
}
```

—Bueno, veo que has encontrado la solución más sencilla. Veo que has usado un define para indicar la separación de los disparos. Muy bien, eso es muy elegante y útil.

—Sí, lo vi en algún ejemplo que estuve mirando por internet y me pareció lo más adecuado...

—Pues sí, la verdad es que está bastante bien. Aún así, a mí me hubiera gustado más dejarlo como parámetro de la clase para poder ajustarlo mientras está funcionando el juego. Imagina que quisieras poner varias armas, con diferentes frecuencias de disparo. Si lo hubieras hecho con una variable en lugar de con un define, sería sencillísimo.

—Tienes razón, ¿quieres que lo cambie? —preguntó.

—No, no. Da igual. De momento nos da igual, y si más adelante queremos cambiarlo, es un momento —le respondí.

—De acuerdo... ¿Qué hacemos ahora? Yo quiero poner ya los enemigos —dijo con bastante ánimo.

—Pues vale. Eso es lo que haremos ahora... ¿Te arreglas tú o quieres que te guíe un poco? —le dije, con la esperanza de no tener que esforzarme demasiado...

—Bueno —respondió—, creo que podré ir haciéndolo mirando por el código del jugador. Creo que tendrán bastante en común.

—De acuerdo, adelante —le dije sin gran entusiasmo.

Siguiendo el procedimiento del último día, agregó una nueva clase de C++ al proyecto con el nombre “Enemigo”. Después, abrió los archivos “Jugador.h” y “Enemigo.h” y empezó a copiar la estructura básica de la clase Jugador en la clase Enemigo. Al final terminó con un archivo “Enemigo.h” con este aspecto:

Ilustración 5: Declaración inicial de la clase enemigo (Enemigo.h)

```

Enemigo.h | Jugador.h
Enemigo
#pragma once

struct BITMAP;

class Enemigo
{
public:
    Enemigo(void);
    Enemigo(BITMAP *_Grafico);

    ~Enemigo(void);

    void Dibuja(BITMAP * Objetivo); // Dibuja el enemigo en el BITMAP Objetivo
    void Actualiza();              // Actualiza la lógica del enemigo

protected:
    BITMAP * Grafico_Enemigo; // Almacenará el gráfico del enemigo

    int Pos_X; // Posición del enemigo en el eje horizontal
    int Pos_Y; // Posición del enemigo en el eje vertical

    int Velocidad; // Velocidad de movimiento del enemigo
};

```

—¿Voy bien? —preguntó, al terminar.

—Sí, de momento todo perfecto.

—Está bien. Ahora voy a intentar que salga un enemigo en la pantalla.

Entonces abrió los archivos “Jugador.cpp” y “Enemigo.cpp” y, tomando la clase Jugador como referencia, fue escribiendo el código de la clase Enemigo. Al rato, se quedó pensando mientras miraba a lo que había hecho hasta ese momento:

Ilustración 6: Estado del archivo Enemigo.cpp

```

Jugador.cpp Enemigo.cpp Enemigo.h Jugador.h
Enemigo Actualiza()
#include <allegro.h>
#include "Enemigo.h"

extern int Resolucion_X;
extern int Resolucion_Y;

Enemigo::Enemigo(void)
{
}

Enemigo::Enemigo(BITMAP * Grafico) : Grafico_Enemigo(Grafico), Velocidad(2)
{
    // Situar la nave centrada en la parte superior de la pantalla (provisional)
    Pos_X = Resolucion_X/2 - Grafico_Enemigo->w/2;
    Pos_Y = 50;
}

Enemigo::~Enemigo(void)
{
}

void Enemigo::Dibuja(BITMAP *Objetivo)
{
    // Dibujamos en la pantalla el objeto.
    draw_sprite(Objetivo, Grafico_Enemigo, Pos_X, Pos_Y);
}

void Enemigo::Actualiza()
{
}
    
```

—¿Qué opinas? —preguntó, sin saber muy bien qué hacer a continuación.

—Bueno, está bien —respondí—. Aún así, te recomiendo que hagas que salga en pantalla un enemigo. Para comprobar que todo va bien, más que nada...

—Sí, casi mejor. Voy a crear un objeto de tipo de enemigo igual que creo uno del tipo Jugador.

Mientras lo decía, abrió el main.cpp, fue buscando todas las apariciones de la variable "Puntero_Jugador" (que almacena un puntero al objeto del jugador) e hizo lo mismo con una nueva variable "Puntero_Enemigo".

Ilustración 7: Inclusión de "Enemigo.h" en el main.cpp para poder utilizar la nueva clase.

```

main.cpp Jugador.cpp Enemigo.cpp Enemigo.h Jugador.h
(Ámbito global) Actualiza()
#include <allegro.h>
#include "Jugador.h"
#include "Enemigo.h"
#include "Disparo.h"

#include <list>
    
```

Ilustración 8: Declaración del puntero al enemigo (esto es provisional, para probarlo)

```

main.cpp | Jugador.cpp | Enemigo.cpp | Enemigo.h | Jugador.h
(Ámbito global) Actualizar()
// Objetos especiales
Jugador *Puntero_Jugador; // Puntero al jugador
Enemigo *Puntero_Enemigo; // Puntero al enemigo
list<Disparo *> Disparos; // Lista de punteros a los disparos
    
```

Ilustración 9: Actualizar el enemigo de prueba (esto realmente no hace nada porque Actualiza está vacía)

```

main.cpp | Jugador.cpp | Enemigo.cpp | Enemigo.h | Jugador.h
(Ámbito global) Actualizar()
// Actualiza la lógica del juego
void Actualizar()
{
    Puntero_Jugador->Actualizar();
    Puntero_Enemigo->Actualizar();

    list<Disparo *>::iterator it;
    
```

Ilustración 10: Llamada a la función Dibuja del enemigo de prueba.

```

main.cpp | Jugador.cpp | Enemigo.cpp | Enemigo.h | Jugador.h
(Ámbito global) Actualizar()
// DIBUJAR TODO
Puntero_Jugador->Dibuja(swap_screen); // Dibujar la nave del jugador.
Puntero_Enemigo->Dibuja(swap_screen);

// Dibujar los disparos
    
```

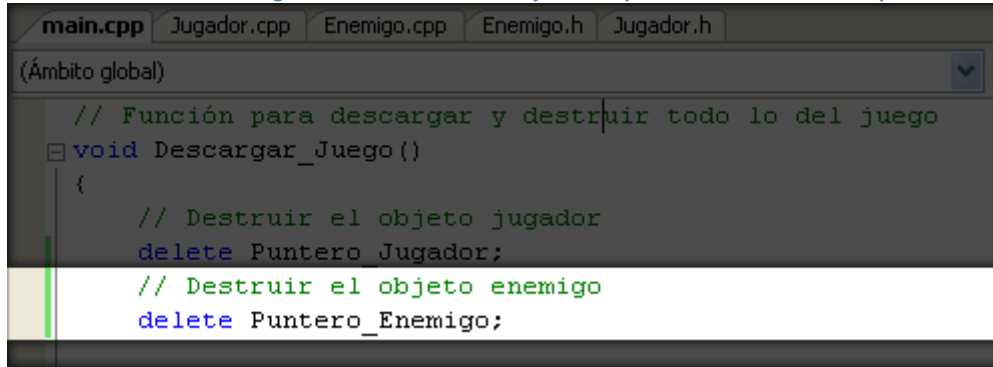
Ilustración 11: Crear el nuevo enemigo. Usamos el gráfico de la nave para poder probarlo enseguida.

```

main.cpp | Jugador.cpp | Enemigo.cpp | Enemigo.h | Jugador.h
(Ámbito global) Cargar_Juego()
// Función para cargar e inicializar todo el juego
void Cargar_Juego()
{
    // Cargar el gráfico del jugador
    Grafico_Jugador = load_bitmap("graficos\\jugador.tga", NULL);
    // Cargar el gráfico de los disparos
    Grafico_Disparo = load_bitmap("graficos\\disparo.tga", NULL);

    // Crear el objeto Jugador
    Puntero_Jugador = new Jugador(Grafico_Jugador);
    // Crear un objeto Enemigo (para probar)
    Puntero_Enemigo = new Enemigo(Grafico_Jugador);
}
    
```

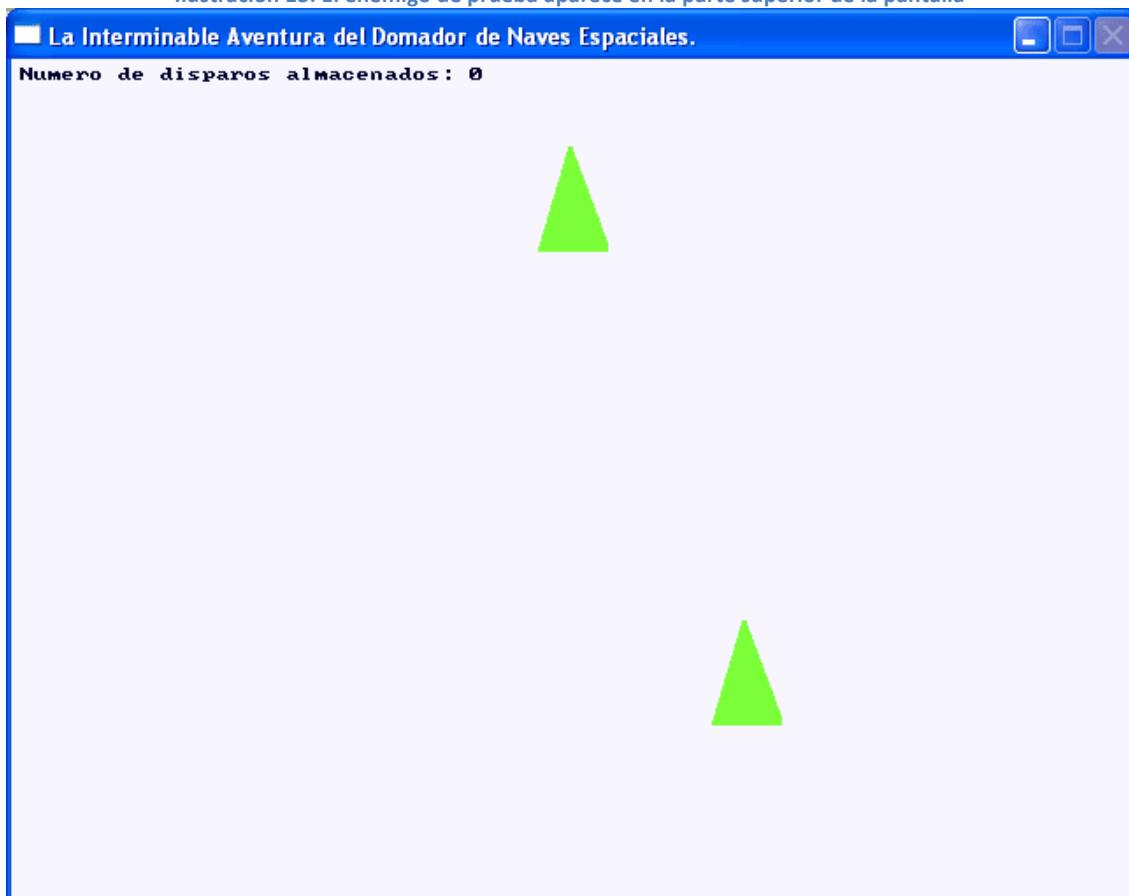
Ilustración 12: Eliminar el enemigo. Escribimos el "delete" justo después de escribir el "new" para no olvidarlo.



```
main.cpp Jugador.cpp Enemigo.cpp Enemigo.h Jugador.h
(Ámbito global)
// Función para descargar y destruir todo lo del juego
void Descargar_Juego()
{
    // Destruir el objeto jugador
    delete Puntero Jugador;
    // Destruir el objeto enemigo
    delete Puntero_Enemigo;
```

Después de realizar estos cambios, compiló y ejecutó el programa para comprobar que todo iba bien. Por suerte, parecía que empezaba a necesitar menos ayuda...

Ilustración 13: El enemigo de prueba aparece en la parte superior de la pantalla



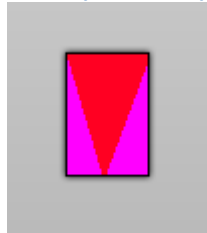
—Bien. Ya aparece en pantalla como queríamos —dijo al verlo en pantalla—. Pero ahora que lo pienso, esto apenas sirve para nada... Los enemigos funcionan muy diferente al Jugador; hay que hacer que salgan cada poco, que se muevan, que disparen...

—Sí, es verdad... Mira, puedes ir haciendo un gráfico diferente para los enemigos y así lo vas pensando un poco mientras tanto.

—De acuerdo —respondió.

En un momento, abrió el Paint.NET y creó un nuevo gráfico para los enemigos. Por no complicarse, hizo un triángulo de la misma forma que el del jugador solo que de color rojo y mirando hacia abajo.

Ilustración 14: Gráfico provisional para el enemigo



—Ya está —dijo mientras lo guardaba junto al resto de gráficos, con el nombre “enemigo.tga”—. Lo cargo en el juego, ¿vale?

—Vale —respondí secamente.

Entonces fue a main.cpp y escribió el código necesario para que el enemigo de pruebas utilizase el nuevo gráfico:

Ilustración 15: Código para cargar y descargar el nuevo gráfico de los enemigos

```

main.cpp | Jugador.cpp | Enemigo.cpp | Enemigo.h | Jugador.h
(Ámbito global) | Cargar_Juego()

// Función para cargar e inicializar todo el juego
void Cargar_Juego()
{
    // Cargar el gráfico del jugador
    Grafico_Jugador = load_bitmap("graficos\\jugador.tga", NULL);
    // Cargar el gráfico de los enemigos
    Grafico_Enemigo = load_bitmap("graficos\\enemigo.tga", NULL);
    // Cargar el gráfico de los disparos
    Grafico_Disparo = load_bitmap("graficos\\disparo.tga", NULL);

    // Crear el objeto Jugador
    Puntero_Jugador = new Jugador(Grafico_Jugador);
    // Crear un objeto Enemigo (para probar)
    Puntero_Enemigo = new Enemigo(Grafico_Enemigo);
}

// Función para descargar y destruir todo lo del juego
void Descargar_Juego()
{
    // Destruir el objeto jugador
    delete Puntero_Jugador;
    // Destruir el objeto enemigo
    delete Puntero_Enemigo;

    // Descargar todos los gráficos
    destroy_bitmap(Grafico_Jugador);
    destroy_bitmap(Grafico_Enemigo);
    destroy_bitmap(Grafico_Disparo);
}

```

Al terminar, ejecuté el juego para comprobar que, efectivamente, el enemigo salía ya con el nuevo gráfico. En cuanto vio que así era, me preguntó acerca de cómo gestionar el comportamiento de los enemigos.

—No se me ocurre cómo hacer que los enemigos vayan apareciendo automáticamente durante la partida... Podría utilizar un contador de “frames” y crear uno cada cierto tiempo, pero eso es un poco cutre; normalmente los enemigos suelen aparecer de forma un poco más aleatoria.

—Tú misma lo has dicho. Si quieres que los enemigos salgan de forma aleatoria, lo que debes usar es la función para generar aleatorios, es decir, la función “rand()”. No sé si te lo había dicho antes —continué explicando—, pero los aleatorios se usan muchísimo en la programación de videojuegos...

—¡Ah, claro! ¡Qué tonta! —respondió—. Entonces lo único que tengo que hacer es utilizar la función rand() para decidir, de forma aleatoria, cuando deben crearse nuevos enemigos...

—Eso es. Lo mejor es que crees una nueva función en el main, y que la llames desde el método Actualiza() del main. Yo pondría un parámetro a esa función para poder ajustar la probabilidad de que se cree un nuevo enemigo —le aconsejé—. También necesitarás una lista de enemigos, igual que la que tienes para los disparos. Ah, y haz que en el modo debug salga el número de enemigos, lo mismo que hicimos también con los disparos.

—De acuerdo, a ello voy.

Fue al main.cpp y, sin demasiada prisa, escribió todo el nuevo código para generar enemigos aleatoriamente. No fue demasiado difícil:

Ilustración 16: Declaración de la lista donde almacenaremos los punteros a los enemigos

```

Disparo.h main.cpp Jugador.cpp Enemigo.cpp Enemigo.h Jugador.h
(Ámbito global)
Jugador *Puntero_Jugador; // Puntero al jugador
Enemigo *Puntero_Enemigo; // Puntero al enemigo
list<Disparo *> Disparos; // Lista de punteros a los disparos
list<Enemigo *> Enemigos; // Lista de punteros a los enemigos
    
```

Ilustración 17: Nueva función que calcula cuando se debe crear un nuevo enemigo

```

Disparo.h main.cpp Jugador.cpp Enemigo.cpp Enemigo.h Jugador.h
(Ámbito global)
// Crea un nuevo enemigo con una probabilidad ajustable a través del parámetro
// A mayor valor de Separacion, menos enemigos creados
void Crea_Enemigos(int Separacion)
{
    if( rand()%Separacion == 1 )
    {
        Enemigos.push_back( new Enemigo(Grafico_Enemigo) );
    }
}
    
```

Ilustración 18: Llamada a la nueva función Crea_Enemigos y código para actualizar todos los enemigos

```

Disparo.h main.cpp Jugador.cpp Enemigo.cpp Enemigo.h Jugador.h
(Ámbito global)
// Actualiza la lógica del juego
void Actualizar()
{
    Puntero_Jugador->Actualiza();
    Puntero_Enemigo->Actualiza();
    Crea_Enemigos(60);

    // Actualizar todos los enemigos
    list<Enemigo *>::iterator it_enemigos;
    for(it_enemigos = Enemigos.begin(); it_enemigos != Enemigos.end(); it_enemigos++)
    {
        (*it_enemigos)->Actualiza();

        if((*it_enemigos)->Eliminar)
        {
            delete *it_enemigos;
            *it_enemigos = NULL;
        }
    }

    // Actualizar todos los disparos
    list<Disparo *>::iterator it;
    for(it = Disparos.begin(); it != Disparos.end(); it++)
    {
    
```

Ilustración 19: Escribir el número de enemigos almacenados cuando se está en modo debug

```

Disparo.h | main.cpp | Jugador.cpp | Enemigo.cpp | Enemigo.h | Jugador.h
(Ámbito global) [Actualizar()]

// Acciones especiales del modo debug
if (Debug)
{
    char Cadena_Numero[20];
    string Aux;

    Aux="Numero de disparos almacenados: ";
    _itoa_s(Disparos.size(), Cadena_Numero, 10);
    Aux+=Cadena_Numero;
    textout_ex(swap_screen, font, (char *)Aux.c_str(), 5, 5, 5, -1);

    Aux="Numero de enemigos almacenados: ";
    _itoa_s(Enemigos.size(), Cadena_Numero, 10);
    Aux+=Cadena_Numero;
    textout_ex(swap_screen, font, (char *)Aux.c_str(), 5, 15, 5, -1);
}

// DIBUJAR TODO

```

—Ya está creo... ¿Me falta algo? —preguntó.

—No sé —respondí—. Compruébalo tú misma.

Al escuchar mi respuesta me miró un poco raro, dándose cuenta, quizás, de que no estaba muy contento. Aún así, ejecutó el programa y vio que aparentemente no cambiaba nada.

—¿Qué pasa? Parece que no cambiamos nada —obviamente se equivocaba.

—Fíjate en el contador de enemigos —le dije mientras se lo señalaba, para mostrarle que iba incrementando su número poco a poco—. Estás dibujando todos los enemigos unos encima de otros. Por eso parece que solo hay uno.

—Ah, claro. ¿Qué hago entonces? ¿Cambio la clase enemigo para hacer que calcule una posición inicial aleatoria?

—Tienes que cambiar la clase enemigo, pero lo mejor es que “parametrices” la posición inicial, en lugar de calcularla en la clase Enemigo. Así podremos calcularla desde fuera y modificarla más fácilmente.

—¿Entonces lo añado como parámetros del constructor de la clase?

—Sí, eso será lo mejor.

No le costó mucho cambiar eso:

Ilustración 20: Modificar el Enemigo.h para indicar la nueva forma del constructor.

```

Disparo.h | main.cpp | Jugador.cpp | Enemigo.cpp | Enemigo.h* | Jugador.h
Enemigo
#pragma once

struct BITMAP;

class Enemigo
{
public:
    Enemigo(void);
    Enemigo(BITMAP * _Grafico, int X_Inicial, int Y_Inicial);

    ~Enemigo(void);

    void Dibuja(BITMAP * Objetivo); // Dibuja el enemigo en el BITMAP Objetivo
    
```

Ilustración 21: Modificaciones en el constructor (Enemigo.cpp) para que reciba los nuevos parámetros y los guarde en las variables de la clase.

```

Disparo.h | main.cpp | Jugador.cpp | Enemigo.cpp | Enemigo.h | Jugador.h
Enemigo
Enemigo(BITMAP * _Grafico)

extern int Resolucion_X;
extern int Resolucion_Y;

Enemigo::Enemigo(void)
{
}

Enemigo::Enemigo(BITMAP * Grafico, int X_Inicial, int Y_Inicial) : Grafico_Enemigo(Grafico),
    Velocidad(2), Eliminar(false)
{
    // Situar el enemigo en la posición que nos han pasado como parámetro
    Pos_X = X_Inicial;
    Pos_Y = Y_Inicial;
}
    
```

Ilustración 22: Establecer la posición inicial en el momento de crear los enemigos.

```

Disparo.h | main.cpp | Jugador.cpp | Enemigo.cpp | Enemigo.h | Jugador.h
(Ámbito global)
Crea_Enemigos(int Separacion)

// Crea un nuevo enemigo con una probabilidad ajustable a través del parámetro
// A mayor valor de Separacion, menos enemigos creados
void Crea_Enemigos(int Separacion)
{
    if( rand()%Separacion == 1 )
    {
        Enemigos.push_back( new Enemigo(Grafico_Enemigo, rand()%Resolucion_X, 15 ) );
    }
}
    
```

Al terminar, compilo y ejecuté de nuevo el juego. Ahora enseguida se vio qué era lo siguiente que había que hacer:

—Está bien —dijo—. Ya se van colocando aleatoriamente por arriba como queríamos. Pero se quedan quietos arriba...

—Claro, eso es porque no programamos nada para que se movieran —contesté.

—Bueno, claro, eso era evidente... Habrá que hacer que se muevan de alguna forma, ¿quieres que haga algo concreto?

—No sé, eso es cosa tuya. Después de todo, este es tu juego; yo solo estoy aquí para guiarte un poco...

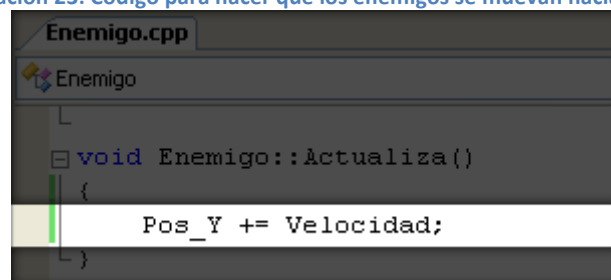
—De acuerdo —pensó durante un momento—. Bueno, supongo que de momento valdrá con que bajen rectos hacia abajo... Luego podemos añadir más tipos de enemigos, ¿no?

—Claro, esa es la idea en realidad. Para hacer que los enemigos se muevan, tienes que modificar el método “Actualiza()” de la clase enemigo.cpp —expliqué—, aunque bueno, supongo que ya lo sabrías, es muy parecido a lo que hicimos con los disparos.

—Vale.

Sin más conversación, se fue al código y lo hizo a la primera:

Ilustración 23: Código para hacer que los enemigos se muevan hacia abajo.



```
Enemigo.cpp
Enemigo
{
  void Enemigo::Actualiza()
  {
    Pos_Y += Velocidad;
  }
}
```

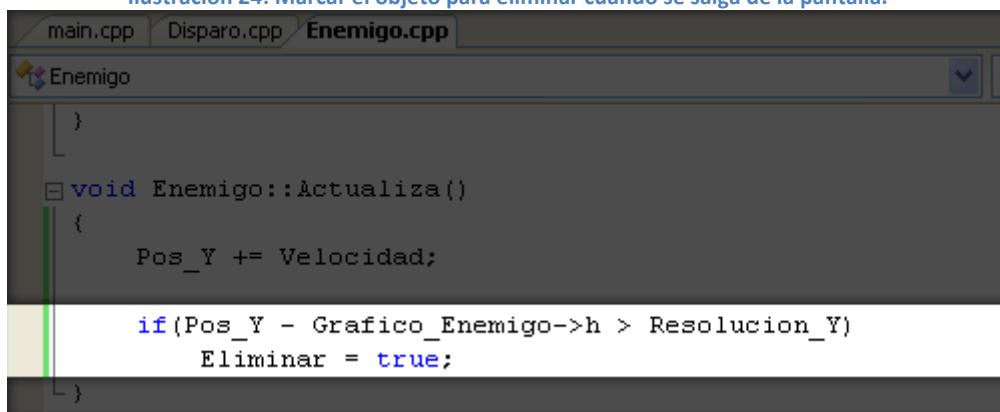
—Eso está bien —le dije—, pero si no añades nada más, te va a pasar lo mismo que nos pasó con los disparos. No estás destruyendo nunca los enemigos.

—Ah, es verdad. Voy a corregirlo ahora.

—Bueno, pero hay una cosa que seguramente no sepas arreglar —le advertí—. Aún así, inténtalo para que veas el error que te dará.

Y lo corrigió enseguida, fijándose en como lo habíamos solucionado para los disparos:

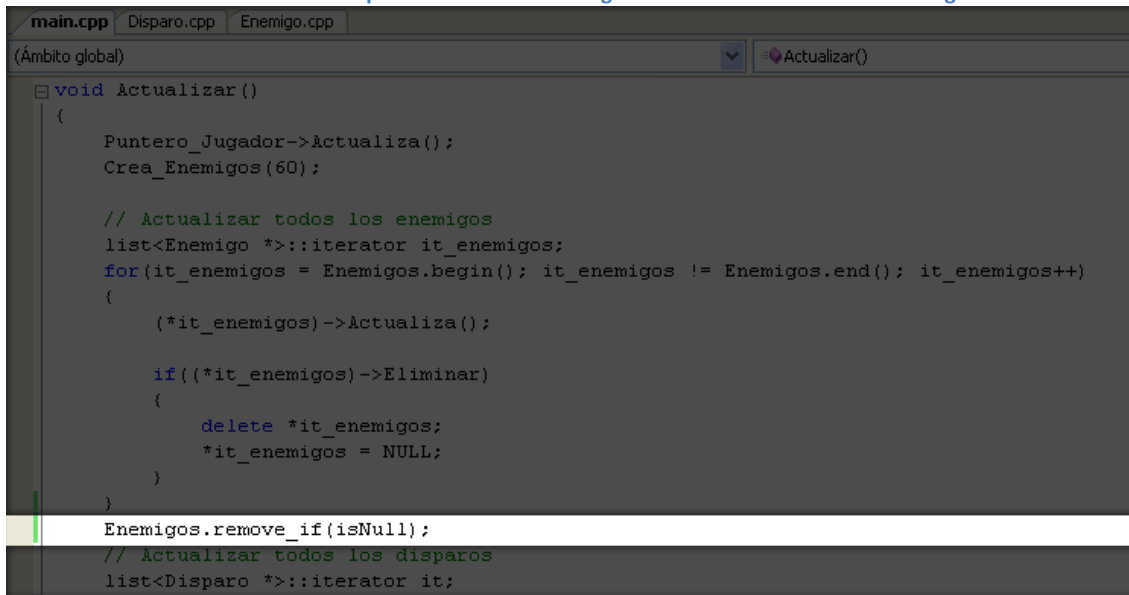
Ilustración 24: Marcar el objeto para eliminar cuando se salga de la pantalla.



```
main.cpp Disparo.cpp Enemigo.cpp
Enemigo
{
}
void Enemigo::Actualiza()
{
  Pos_Y += Velocidad;

  if(Pos_Y - Grafico_Enemigo->h > Resolucion_Y)
    Eliminar = true;
}
```

Ilustración 25: Línea para eliminar los enemigos destruidos de la lista de enemigos



```

main.cpp | Disparo.cpp | Enemigo.cpp
(Ámbito global)
void Actualizar()
{
    Puntero_Jugador->Actualiza();
    Crea_Enemigos(60);

    // Actualizar todos los enemigos
    list<Enemigo *>::iterator it_enemigos;
    for(it_enemigos = Enemigos.begin(); it_enemigos != Enemigos.end(); it_enemigos++)
    {
        (*it_enemigos)->Actualiza();

        if((*it_enemigos)->Eliminar)
        {
            delete *it_enemigos;
            *it_enemigos = NULL;
        }
    }

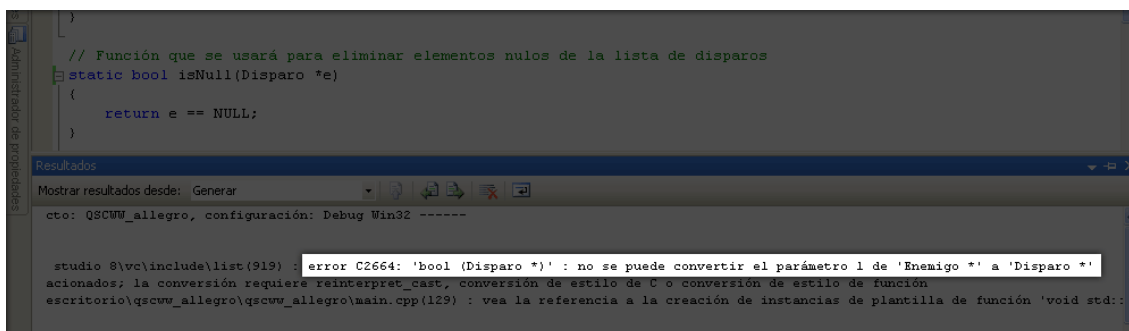
    Enemigos.remove_if(isNull);
    // Actualizar todos los disparos
    list<Disparo *>::iterator it;

```

—Yo creo que ya está —dijo al hacer los cambios—. ¿Dices que esto va a fallar?

—Sí. Estoy seguro de ello, aunque puedes probar si no te fías...

A pesar de que ella sabía perfectamente que iba a fallar (si no, yo nunca lo habría dicho con tanta seguridad), lo compiló para ver el error con sus propios ojos:



```

// Función que se usará para eliminar elementos nulos de la lista de disparos
static bool isNull(Disparo *e)
{
    return e == NULL;
}

Resultados
Mostrar resultados desde: Generar
cto: QSCWU_allegro, configuración: Debug Win32 -----

studio 8\vc\include\list(919) : error C2664: 'bool (Disparo *)' : no se puede convertir el parámetro 1 de 'Enemigo *' a 'Disparo *'
acionados; la conversión requiere reinterpret_cast, conversión de estilo de C o conversión de estilo de función
escritorio\qscwu_allegro\qscwu_allegro\main.cpp(129) : vea la referencia a la creación de instancias de plantilla de función 'void std::

```

—Dice que no se puede convertir un puntero a Enemigo a un puntero a Disparo —dijo al verlo—. Eso tiene sentido, el problema es que no sé donde estoy intentando hacer esa conversión.

—Bueno, es lógico porque lo que está fallando es, seguramente, la parte más complicada del código. El problema está en la función isNull, que ahora mismo solo puede recibir un puntero a un objeto de la clase Disparo. El problema es que estamos intentando usar esa misma función para discernir si un puntero a Enemigo es null. Acuérdate de lo que hacíamos con el “remove_if”.

—Ah, ya entiendo... ¿Tengo que hacer dos funciones, entonces? ¿Una “enemigoIsNull” y otra “disparoIsNull”?

—Bueno, hacer eso funcionaría, pero no es la mejor solución. Imagina que tuvieras muchos más tipos de objetos con los que quisieras hacer lo mismo —expliqué—. Hacer una función `isNull` para cada uno sería un jaleo.

—Entonces no sé cómo podemos arreglar eso...

—Es lógico en parte. Esto ya requiere un poco más de nivel en C++; lo más cómodo aquí es utilizar un puntero a `void`.

—¿Puntero a `void`? —preguntó, intrigada.

—Sí, un puntero a `void` es un “puntero a nada” que se puede usar como un “puntero a cualquier cosa”. Un puntero a `void` es, simplemente, una dirección de memoria que apunta a cualquier cosa. Por eso no se puede hacer casi nada con ello, salvo que lo conviertas a un puntero del tipo que sea.

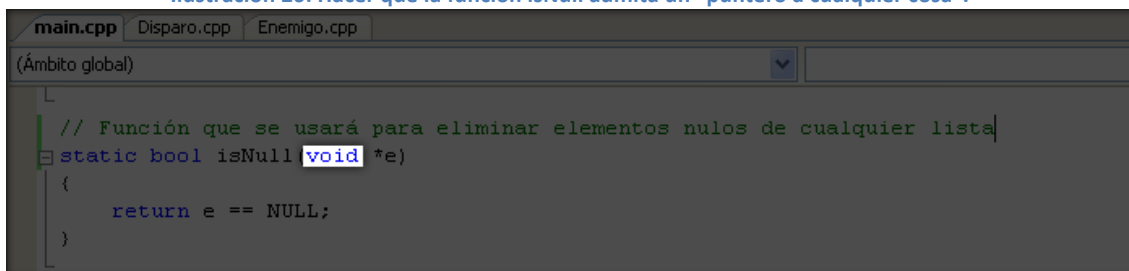
—Buf —dijo preocupada—. Vaya lío, ¿no?

—Es posible, pero tampoco te estreses con ello. En este caso lo único que vamos a hacer es comprobar si un puntero (una dirección de memoria) es `NULL` o no. Para eso da igual que el puntero sea a `void` o a `Enemigo`.

—De acuerdo, ¿cómo lo hago entonces?

—Muy sencillo —le dije, mientras cogía el teclado para realizar el pequeño cambio que hacía falta.

Ilustración 26: Hacer que la función `isNull` admita un “puntero a cualquier cosa”.



```
main.cpp | Disparo.cpp | Enemigo.cpp
(Ámbito global)
[
// Función que se usará para eliminar elementos nulos de cualquier lista
static bool isNull(void *e)
{
    return e == NULL;
}
```

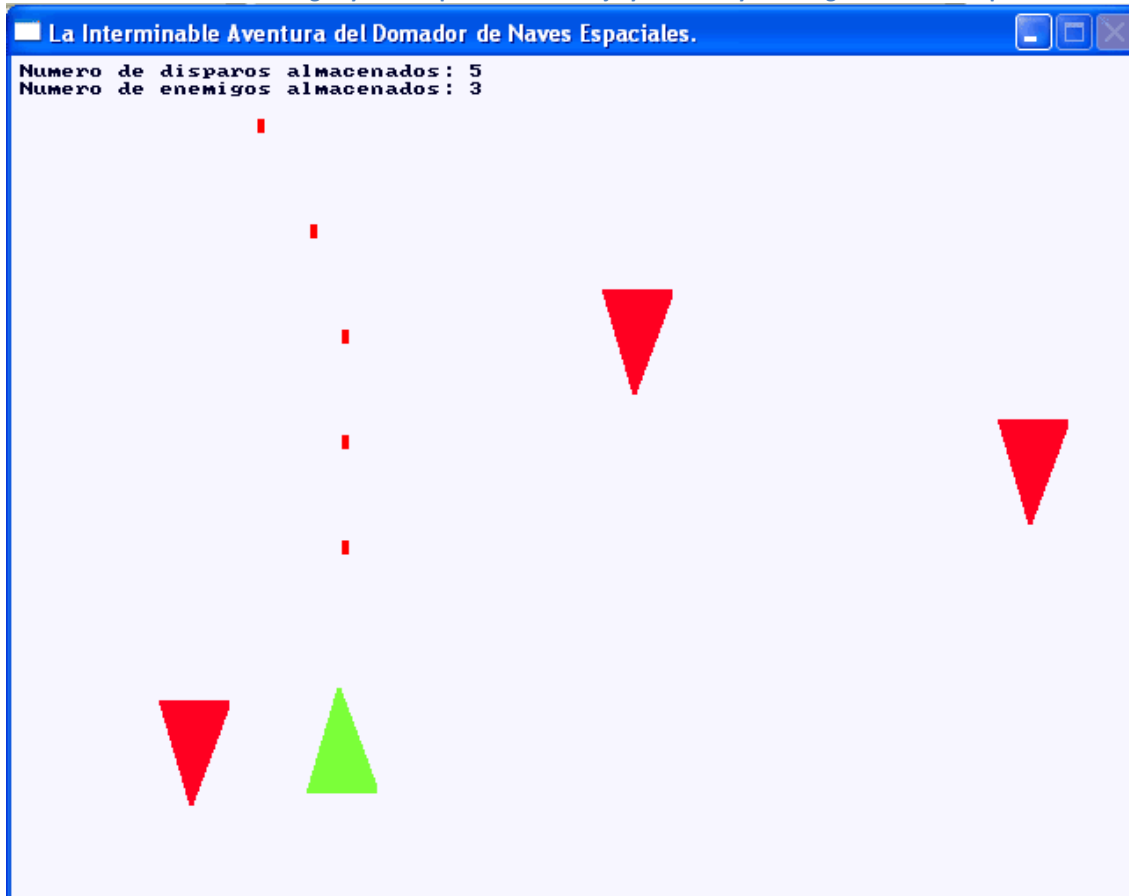
—Joder —dijo Olaya sorprendida—, menuda tontería... Desde luego, es muchísimo más sencillo que la solución que yo daba.

—Pues sí, la verdad es que cuanto más dominas el lenguaje, más trabajo te ahorras. Aún así, insisto en que lo importante es conseguir que las cosas funcionen, y tú lo habrías conseguido aún sin saber esto. Tampoco es crucial dominar completamente el lenguaje que estás usando; poco a poco vas aprendiendo estas cosas.

—Ya, ya. Pero mola programar así de bien —dijo sonriéndome.

Una vez hecho ese cambio, compiló de nuevo y, esta vez sí, el juego arrancó correctamente.

Ilustración 27: Los enemigos ya se desplazan hacia abajo y se destruyen al llegar al final de la pantalla.



—¡Ey, esto ya empieza a molar! —dijo mientras hacía como que jugaba a su propio juego, disparando y esquivando a los enemigos.

—Bueno, eso apenas es nada —le dije—. Todavía quedan muchas cosas que hacer...

—Ya, hombre, ya lo sé —respondió—. Oye, Miguel, no te noto tan emocionado como los otros días... ¿Pasa algo? —preguntó, mientras me miraba.

—No, nada, tranquila —intenté disimular—. Estoy bien.

—Bueno, lo que tú digas. ¡Ahora quiero poder matar a los enemigos! ¿Cómo se hace eso?

—Pues lo que tienes que hacer es una detección de colisiones entre los enemigos y los disparos. Es bastante parecido a lo que hicimos en el PONG con la pelota y las palas, por ejemplo.

—Sí, tiene sentido... Pero para eso habría que comprobar las colisiones de todos los disparos con todos los enemigos. ¿Eso no será demasiado lento?

—Hombre, lento seguro que no, porque el juego es muy sencillito y las máquinas de ahora son bastante potentes. Aún así, es verdad que esto admite un montón de optimización, pero si lo optimizáramos sería complicarnos mucho —le expliqué—. Yo creo que no merece la pena en este caso.

—Vale... Pero es que tengo bastantes dudas. Por ejemplo, ¿Dónde pongo el código para comprobar las colisiones? ¿En el método actualizar los enemigos valdría?

—Sí, claro. Ahí se podría hacer, pero no es el mejor sitio. Ten en cuenta que los enemigos no tienen acceso directo a la lista de disparos, que está en el main.cpp. En este caso podríamos hacer un extern y usar la lista desde ahí, porque la hemos declarado como global para todo el programa, pero eso no es una solución real; normalmente no tendríamos un acceso tan fácil a esa lista de enemigos.

—Entiendo... Pues no se me ocurre otra cosa —dijo, esperando mi solución al problema.

—Vale, te ayudaré un poco... Yo lo que haría, para no complicarme demasiado, sería hacer una función en el main.cpp que recibiera como parámetro un puntero a un Enemigo —comencé a explicarme—. Después, comprobaría las colisiones de ese enemigo con todos los disparos y, en caso de ser alguna positiva, marcaría para destruir tanto el enemigo como el disparo. Todo ello desde el main, que tiene acceso a toda la información necesaria.

Olaya se tomó un par de segundos para pensarlo y pareció satisfecha:

—Entendido. Parece bastante fácil, voy a intentarlo.

No le resultó tan fácil, sin embargo. Estuvo un rato mirando otras partes del código y probando, pero no consiguió obtener una solución aceptable...

—No sé —dijo, al ver que no avanzaba—. Sé que esta función va ahí —continuó mientras me señalaba a una función que había creado en el main—, pero no sé cómo obtener todos los parámetros que necesito de los disparos y los enemigos. Creo que es un poco complicado para hacerlo yo sola.

—Bah —le dije—. Seguro que con un poco más de tiempo lo acabarías sacando... Aún así te ayudaré, que no tenemos todo el día.

—Te escucho.

—De acuerdo. Vamos a empezar por modificar las clases Enemigo y Disparo de tal forma que nos dejen obtener la información que necesitamos para comprobar las colisiones desde el main (estas funciones se llaman “observadores”). Además, vamos a añadir un método auxiliar en los Enemigos que nos ayudará mucho a la hora de comprobar las colisiones. Añade lo siguiente:

Ilustración 28: Declaraciones de los nuevos observadores de la clase Disparo

```

main.cpp Disparo.h Disparo.cpp Enemigo.h Enemigo.cpp
Disparo
#pragma once

struct BITMAP;

class Disparo
{
public:
    Disparo(void);
    Disparo(BITMAP * Grafico, int x, int y);
    ~Disparo(void);

    void Dibuja(BITMAP * Objetivo); // Dibuja en el BITMAP Objetivo
    void Actualiza(); // Actualiza la lógica

    // Funciones para acceder a la geometría del objeto (necesario para las colisiones)
    int Obtener_PosX();
    int Obtener_PosY();
    int Obtener_Ancho();
    int Obtener_Alto();

    bool Eliminar; // Indica si el disparo debe ser eliminado.
protected:
    BITMAP * Grafico_Disparo; // Almacenará el gráfico

    int Pos_X; // Posición en el eje horizontal
    int Pos_Y; // Posición en el eje vertical

    int Velocidad; // Velocidad de movimiento
};
    
```

Ilustración 29: Implementación de los nuevos observadores de la clase Disparo

```

main.cpp Disparo.h Disparo.cpp Enemigo.h Enemigo.cpp
Disparo
void Disparo::Actualiza()
{
    Pos_Y -= Velocidad;

    if(Pos_Y + Grafico_Disparo->h <= 0)
        Eliminar = true;
}

int Disparo::Obtener_PosX()
{
    return Pos_X;
}

int Disparo::Obtener_PosY()
{
    return Pos_Y;
}

int Disparo::Obtener_Ancho()
{
    return Grafico_Disparo->w;
}

int Disparo::Obtener_Alto()
{
    return Grafico_Disparo->h;
}
    
```

Ilustración 30: Declaración de los observadores y de la función auxiliar en Enemigo.h

```

main.cpp | Disparo.h | Disparo.cpp | Enemigo.h | Enemigo.cpp
Enemigo
#pragma once

struct BITMAP;

class Enemigo
{
public:
    Enemigo(void);
    Enemigo(BITMAP * _Grafico, int X_Inicial, int Y_Inicial);

    ~Enemigo(void);

    void Dibuja(BITMAP * Objetivo); // Dibuja el enemigo en el BITMAP Objetivo
    void Actualiza(); // Actualiza la lógica del enemigo

    // Funciones para acceder a la geometría del objeto (necesario para las colisiones)
    int Obtener_PosX();
    int Obtener_PosY();
    int Obtener_Ancho();
    int Obtener_Alto();

    bool Colision_Punto(int x, int y); // Retorna cierto si el punto (x,y) está dentro del gráfico del objeto

    bool Eliminar; // Indica si el enemigo debe ser eliminado.
protected:

```

Ilustración 31: Código de las nuevas funciones en Enemigo.cpp

```

main.cpp | Disparo.h | Disparo.cpp | Enemigo.h | Enemigo.cpp
Enemigo
Colision_Punto(int x, int y)

int Enemigo::Obtener_PosX()
{
    return Pos_X;
}

int Enemigo::Obtener_PosY()
{
    return Pos_Y;
}

int Enemigo::Obtener_Ancho()
{
    return Grafico_Enemigo->w;
}

int Enemigo::Obtener_Alto()
{
    return Grafico_Enemigo->h;
}

// Comprueba si un punto (x,y) "está dentro" del gráfico del enemigo
bool Enemigo::Colision_Punto(int x, int y)
{
    // Declarar e inicializar las cuatro esquinas del gráfico
    int x1, x2, y1, y2;
    x1 = Pos_X;
    x2 = Pos_X + Grafico_Enemigo->w;
    y1 = Pos_Y;
    y2 = Pos_Y + Grafico_Enemigo->h;

    // Si el punto está dentro del rectángulo
    if (x1 < x && x < x2 && y1 < y && y < y2)
    {
        return true;
    }
    return false;
}

```

—Vale, dijo ella. Creo que, teniendo esto, podré seguir yo sola...

—Me alegro —respondí—. ¿Qué es lo siguiente que vas a hacer?

—Ahora lo veo fácil: Voy a crear una función en el main.cpp que utilice estas funciones para comprobar todas las colisiones entre disparos y enemigos. Lo que me dijiste antes, vamos.

—Pues venga. Todo tuyo —dije, aliviado de no tener que explicárselo también.

En ese momento, se puso a ello y no tardó mucho en escribir la función de la que habíamos hablado:

Ilustración 32: Función que comprueba si existe una colisión entre un enemigo y cualquier disparo.

```

main.cpp | Disparo.h | Disparo.cpp | Enemigo.h | Enemigo.cpp
(Ámbito global) | Comprobar_Colision_Enemigo_Disparo(Enemigo* enemigo)
}
}
// Comprueba si existe una colisión entre un enemigo y todos los disparos en pantalla
// En caso afirmativo, destruye tanto el enemigo como el disparo.
void Comprobar_Colision_Enemigo_Disparo(Enemigo* enemigo)
{
    // Recorrer todos los disparos almacenados
    list<Disparo *>::iterator it;
    for(it = Disparos.begin(); it != Disparos.end(); it++)
    {
        // Obtener los valores necesarios para formar el rectángulo del disparo
        int x1 = (*it)->Obtener_PosX();
        int x2 = (*it)->Obtener_PosX() + (*it)->Obtener_Ancho();
        int y1 = (*it)->Obtener_PosY();
        int y2 = (*it)->Obtener_PosY() + (*it)->Obtener_Alto();
        // Comprobar si alguna de las cuatro esquinas del disparo colisiona con el enemigo
        if(enemigo->Colision_Punto(x1, y1) ||
            enemigo->Colision_Punto(x1, y2) ||
            enemigo->Colision_Punto(x2, y1) ||
            enemigo->Colision_Punto(x2, y2))
        {
            // Existe colisión. Se elimina tanto el disparo como el enemigo
            enemigo->Eliminar = true;
            (*it)->Eliminar = true;
        }
    }
}

// Actualiza la lógica del juego
void Actualizar()
{
    Puntero_Jugador->Actualiza();
}

```

—¿Crees que está bien? —preguntó al terminar.

—Sí. En principio lo veo todo bien... Aunque bueno, debes saber que estas colisiones son lo más simple que se puede hacer —hice un esfuerzo por explicarle esto. Me pareció bastante importante—. En proyectos más complejos, se suele optimizar mucho más y se tienen que tener otras muchas cosas en cuenta. Por ejemplo, si los disparos fueran muy rápido podrían “saltar” por encima de los enemigos sin que las colisiones se detectaran... En resumen, que el tema de las colisiones puede complicarse muchísimo.

—No hace falta que lo jures... Hasta ahora es lo que más me cuesta entender —reconoció.

—Lo sé. Por eso, quiero insistir en que cojas lápiz y papel cada vez que te entren dudas en un tema de estos. Realmente ayuda verlo dibujado.

—También he podido comprobar eso más de una vez...

—Bien. Ahora solo falta que llames a esta función nueva por cada enemigo que haya en pantalla. El mejor sitio para hacerlo es en el bucle que ya tienes en main.cpp para actualizar los enemigos.

Ilustración 33: Llamada a la función que comprueba las colisiones entre los enemigos y los disparos

```

main.cpp | Disparo.h | Disparo.cpp | Enemigo.h | Enemigo.cpp
(Ámbito global) Actualizar()
// Actualiza la lógica del juego
void Actualizar()
{
    Puntero_Jugador->Actualizar();
    Crea_Enemigos(60);

    // Actualizar todos los enemigos
    list<Enemigo *>::iterator it_enemigos;
    for(it_enemigos = Enemigos.begin(); it_enemigos != Enemigos.end(); it_enemigos++)
    {
        (*it_enemigos)->Actualizar();
        Comprobar_Colision_Enemigo_Disparo(*it_enemigos);

        if((*it_enemigos)->Eliminar)
        {
            delete *it_enemigos;
            *it_enemigos = NULL;
        }
    }
    Enemigos.remove_if(isNull);
    // Actualizar todos los disparos

```

—¿Ya está? ¿Ya puedo jugar? —preguntó emocionada.

—Sí, bueno, todavía le falta bastante para que sea divertido jugar... Pero sí, ya puedes destruir a los enemigos.

En cuanto se lo dije, lo ejecuté y estuvo un rato moviéndose por la pantalla y disparando a todo lo que se movía. Al poco, se cansó:

—Tienes razón —me dijo—, esto es bastante aburrido todavía... Solo hay que mantener pulsado el botón de disparar y ya nunca mueres.

—Sí, la verdad es que tal y como está ahora, no tiene ninguna gracia. Habría que hacer que los enemigos también dispares, que se muevan de más formas, limitar el disparo continuo, hacer que el jugador muera si choca con otro enemigo... Pero bueno, ya lo tienes encaminado; seguro que sabrás hacer muchas de esas cosas tú sola —ya llevábamos bastante tiempo con esto y yo no tenía ganas de quedarme mucho más tiempo.

—Ya, pero seguro que avanzo más si me ayudas. ¿No quieres quedarte otro rato más y añadimos alguna de esas cosas?

—Casi mejor no... No tengo ganas de estar aquí cuando vuelva ese... —no recordaba su nombre—. Carlos.

—¿Por? ¿Qué pasa con Carlos? —preguntó, extrañada por mi comentario.

—No nada —respondí sin querer meterme en problemas—. No me pasa nada con Carlos...

—Como que no —me interrumpió—. ¿Qué pasa con Carlos? Hoy te estoy notando muy raro. Quiero que me expliques qué te pasa —exigió.

Mi primera idea fue inventarme algo, o evitar la pregunta de alguna forma. Sin embargo, sabía que Olaya me conocía muy bien, y ya había notado que me pasaba algo... No me iba a dejar escapar sin saber lo que me pasaba, y no veía la forma de evitarlo. Al final elegí decir la verdad; parecía la salida más rápida...

—No me pasa nada con Carlos —respondí, sin poder mirarle a la cara—. ¡El problema es que no me puedo enfadar contigo!

—¿Qué? ¿Por qué te ibas a enfadar conmigo? —preguntó, fingiendo que no entendía nada. En realidad ya empezaba a entenderlo; era demasiado inteligente como para no darse cuenta.

—¡Oh, vamos! Creo que está bastante claro —dije, mirándola, ahora sí, a los ojos—. ¡Porque me gustas, por eso! ¡Me gustas mucho!

Conseguí aguantar la mirada el tiempo justo para decirlo. Al pensar en ello después, yo mismo me sorprendí de que fuera capaz de hacerlo. Sin embargo, en ese momento no pensé en ello y lo dije sin más; no tenía esperanzas de que sirviera para nada más que para irme cuanto antes de allí.

—Ahora no te puedes quedar callada —le dije al ver que no decía nada. Aparentemente, el hecho de que ella me gustara no era tan evidente para ella. Parecía estar muy sorprendida.

—No... Lo siento —dijo, lentamente, después de un momento—. Lo siento, Miguel, no tenía ni idea... Me caes muy bien, pero...

—¡No, déjalo! —le interrumpí—. Ya estoy harto de escuchar eso. Sí, ya sé que soy muy bueno contigo y que te caigo genial, pero al final te quedas con el otro. Con el que apenas sabe juntar dos frases con sentido y sin insultar a nadie. Con el que te insulta cada vez que se emborracha con sus amigos. Con el que no se preocupa por ti. ¡Con el otro!

—¡Oh, vamos! —respondió Olaya, casi llorando—. Tú no tienes ni idea de cómo es Carlos. No entiendo por qué dices eso, estás siendo un gilipollas.

Más adelante me di cuenta de que, efectivamente, estaba siendo un gilipollas. Pero claro, si fuéramos capaces de pensar tan rápido como para darnos cuenta de esas cosas en el momento, la vida perdería parte de su gracia...

—Pues no te vas a reír más de este gilipollas —le dije mientras me levantaba y recogía mi abrigo—. Y ya puedes ir buscándote a otro para que te de clases gratis... ¡Adiós!

—No, ¡espera! —me dijo mientras me seguía por el pasillo.

Salí de su casa y bajé corriendo las escaleras hasta el portal. Al salir vi que había empezado a llover bastante, pero no me importó. A los diez minutos estaba en el autobús de camino a Oviedo y a los tres cuartos de hora ya había llegado a casa.

Me encerré en mi habitación, me acurruqué en una esquina y me quedé quieto sin hacer nada...

Al día siguiente me di cuenta de la estupidez que había hecho.

Fin del capítulo

Nota del autor:

Hola, soy Miguel Santirso, el autor de esta guía y ahora me dirijo a ti, el lector que has seguido este capítulo hasta el final. Realmente agradezco que hayas elegido esta guía de entre todas las que hay en internet y espero que hayas disfrutado y aprendido con ella.

Como cualquier cosa que se haga, esta guía puede contener errores, omisiones o partes mejorables. Por eso, os invito a que contactéis conmigo a través de los diferentes medios que tenéis a vuestra disposición para comentarme cualquier cosa que me ayude a mejorar los siguientes capítulos. También podéis consultar dudas que tengáis acerca del desarrollo de videojuegos (ya sean dudas técnicas o no), pero esto prefiero que lo preguntéis a través de los foros en los que me suelo mover (www.stratos-ad.com básicamente) para que cualquiera pueda leer vuestras preguntas y os pueda responder más gente aparte de mí.

Mi correo electrónico personal es tirso.00@gmail.com, pero os pediría que lo utilicéis solo para asuntos importantes ya que para otras cosas prefiero que me contactéis de cualquier otra forma.

Apéndice A: Listado de imágenes

Ilustración 1: Ahora los disparos salen espaciados.....	4
Ilustración 2: Declaración de la nueva variable en Jugador.h.....	4
Ilustración 3: Inicializar la nueva variable en el constructor de Jugador y definir la separación ..	5
Ilustración 4: Cambios en la función Actualiza() de Jugador.cpp	5
Ilustración 5: Declaración inicial de la clase enemigo (Enemigo.h)	6
Ilustración 6: Estado del archivo Enemigo.cpp	7
Ilustración 7: Inclusión de "Enemigo.h" en el main.cpp para poder utilizar la nueva clase.	7
Ilustración 8: Declaración del puntero al enemigo (esto es provisional, para probarlo).....	8
Ilustración 9: Actualizar el enemigo de prueba (esto realmente no hace nada porque Actualiza está vacía)	8
Ilustración 10: Llamada a la función Dibuja del enemigo de prueba.....	8
Ilustración 11: Crear el nuevo enemigo. Usamos el gráfico de la nave para poder probarlo enseguida.	8
Ilustración 12: Eliminar el enemigo. Escribimos el "delete" justo después de escribir el "new" para no olvidarlo.....	9
Ilustración 13: El enemigo de prueba aparece en la parte superior de la pantalla	9
Ilustración 14: Gráfico provisional para el enemigo	10
Ilustración 15: Código para cargar y descargar el nuevo gráfico de los enemigos	11
Ilustración 16: Declaración de la lista donde almacenaremos los punteros a los enemigos.....	12
Ilustración 17: Nueva función que calcula cuando se debe crear un nuevo enemigo.....	12
Ilustración 18: Llamada a la nueva función Crea_Enemigos y código para actualizar todos los enemigos	12
Ilustración 19: Escribir el número de enemigos almacenados cuando se está en modo debug	13
Ilustración 20: Modificar el Enemigo.h para indicar la nueva forma del constructor.	14
Ilustración 21: Modificaciones en el constructor (Enemigo.cpp) para que reciba los nuevos parámetros y los guarde en las variables de la clase.....	14
Ilustración 22: Establecer la posición inicial en el momento de crear los enemigos.....	14
Ilustración 23: Código para hacer que los enemigos se muevan hacia abajo.....	15
Ilustración 24: Marcar el objeto para eliminar cuando se salga de la pantalla.	15
Ilustración 25: Línea para eliminar los enemigos destruidos de la lista de enemigos.....	16
Ilustración 26: Hacer que la función isNull admita un "puntero a cualquier cosa".	17
Ilustración 27: Los enemigos ya se desplazan hacia abajo y se destruyen al llegar al final de la pantalla.....	18
Ilustración 28: Declaraciones de los nuevos observadores de la clase Disparo	20
Ilustración 29: Implementación de los nuevos observadores de la clase Disparo	20
Ilustración 30: Declaracion de los observadores y de la función auxiliar en Enemigo.h	21
Ilustración 31: Código de las nuevas funciones en Enemigo.cpp	21
Ilustración 32: Función que comprueba si existe una colisión entre un enemigo y cualquier disparo.	22
Ilustración 33: Llamada a la función que comprueba las colisiones entre los enemigos y los disparos	23

