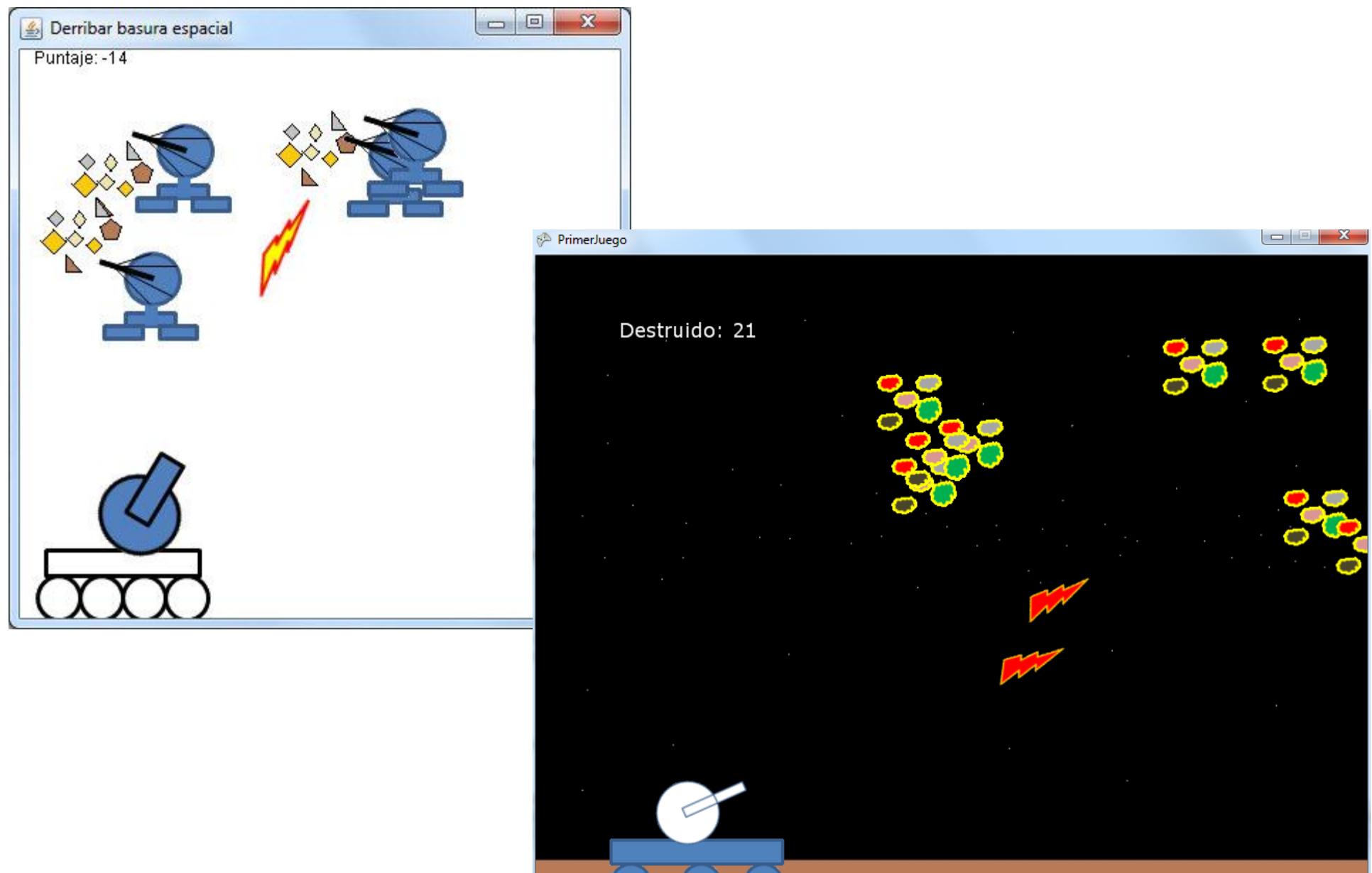




Universidad Libre  
Seccional Cali

# DESARROLLO DE JUEGOS EN 2D USANDO JAVA Y MICROSOFT XNA.



## Contenido

Dedicatoria .....	4
Licencia de este libro .....	5
Agradecimientos .....	6
Introducción .....	7
1. El entorno gráfico y las imágenes .....	8
1.1 Primitiva Gráfica.....	8
1.2 Cargando una imagen .....	8
1.3 Haciendo el código reutilizable .....	10
1.4 Mejorando la carga de imagen .....	10
1.5 Mejorando más aún la carga de imagen.....	11
1.6 Combinando controles con entorno gráfico.....	12
1.7 Posicionando los controles gráficos .....	13
2. Principios de control.....	16
2.1 Eventos .....	16
3. Colisiones .....	18
3.1 Definición .....	18
3.2 Validando una colisión entre dos objetos gráficos .....	18
3.3 Haciendo más sencillo el validar colisiones.....	22
3.4 Consideraciones sobre las colisiones .....	23
4. Animación .....	25
4.1 Uso de hilos para hacer animación.....	25
4.2 Deteniendo y reiniciando el hilo .....	26
4.3 Minimizando el titileo de la animación (técnica de double buffering).....	28
4.4 Mejorando la técnica para minimizar el titileo de la animación.....	29
4.5 Optimizando la técnica que minimiza el titileo de la animación .....	31
5. Captura de eventos de teclado y ratón .....	33
5.1 Implementando la captura de eventos.....	33
6. Uso de sonidos y música de fondo .....	36
6.1 Inicio .....	36
6.2 Ejemplo de animación y sonido .....	37
7. Organizando el código .....	41
8. Juego de ejemplo .....	45
9. El juego como Applet .....	54
9.1 Applets y servidores remotos .....	58
10. Otra forma de tener gráficos y animación en Java: El componente JPanel .....	60
11. Uso del componente JPanel en una aplicación de escritorio y en applets .....	62
12. Animación y control haciendo uso de JPanel .....	64
13. Juego reescrito usando el componente JPanel.....	67
14. XNA: Instalación de Microsoft XNA Game Studio 3.1 .....	75
15. XNA: Empezando a desarrollar en XNA .....	82
16. XNA: ¿Cómo es una aplicación? .....	85
17. XNA: Agregando un fondo de pantalla estático al juego .....	87
18. XNA: Manejo de imágenes con Paint .NET .....	97
19. XNA: Agregar el protagonista a nuestro juego .....	106
20. XNA: Buenas prácticas en el código fuente.....	116
21. XNA: Controlando al protagonista de nuestro juego.....	119
22. XNA: El protagonista lanza rayos .....	124
23. XNA: El "enemigo" a batir .....	133
24. XNA: Colisiones. Haciendo que el rayo golpee al enemigo .....	142
25. XNA: Mostrando el puntaje .....	143
26. XNA: Sonidos en el juego.....	151
27. XNA: Mejorando los tipos de letra para mostrar texto en el juego .....	170
28. XNA: Platformer Starter Kit 3.1 .....	179
29. XNA: Animando un personaje del juego .....	182
30. XNA: Cambiando la orientación de la animación del personaje del juego .....	187
31. XNA: Cambiando la orientación de la animación del personaje del juego con el teclado.....	190
32. XNA: Haciendo scroll (desplazamiento) del fondo de la pantalla.....	193
33. XNA: Mejorando el código para hacerlo más fácil de entender .....	204
34. XNA: Uso de un motor de Física.....	211
35. XNA: Usando un Motor de Física. Colisión entre dos objetos rectangulares.....	219
36. XNA: Usando un Motor de Física. Torque y control por teclado.....	232

37. XNA: Usando un Motor de Física. Colisión con múltiples objetos rectangulares.	236
38. XNA: Usando un Motor de Física. Colisión entre objeto rectangular y circular.	242
Bibliografía	247

## Dedicatoria

Dedicado a mi familia: José Alberto (mi padre), María del Rosario (mi madre), Diana Pilar (mi hermana) y Sally (mi gata).



## Licencia de este libro



## Marcas registradas

Universidad Libre – Seccional Cali. Enlace: <http://www.unilibrecali.edu.co/home/>

En este libro se hace uso de las siguientes tecnologías registradas:

Microsoft ® XNA ® Enlace: <http://msdn.microsoft.com/es-co/xna>

Microsoft ® Windows ® Enlace: <http://windows.microsoft.com/en-US/windows/home>

Microsoft ® Visual Studio 2010 Express Edition ® Enlace: <http://www.microsoft.com/visualstudio/en-us/products/2010-editions/express>

Microsoft ® Visual C# ® Enlace: <http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-csharp-express>

Microsoft ® XBOX ® Enlace: <http://marketplace.xbox.com/es-CO/>

Oracle ® Java ® Enlace: <http://www.oracle.com/technetwork/java/index.html>

## Agradecimientos

Este libro logró ser publicado gracias a las siguientes personas:

**Fabian Castillo Peña**

Director Programa  
Ingeniería de Sistemas  
Facultad de Ingeniería  
Universidad Libre

**María Mercedes Sinisterra Díaz**

Directora Centro de Investigaciones  
Programa de Ingeniería de Sistemas  
Universidad Libre

## Introducción

Buenas prácticas de programación y conocimientos se adquieren cuando se desarrolla un videojuego que pueden ser útiles en otros campos del software. Al programar un videojuego, debemos estar atentos a diversos aspectos como:

### Facilidad de uso

- El jugador debe comprender en qué consiste el juego;
- Cómo se juega;
- El acceso a los controles debe ser fluido y sencillo; y,
- Las opciones del juego deben ser claras.

### Desempeño

- La buena optimización de los algoritmos usados; y,
- El correcto uso de recursos como imágenes y sonidos que no deben ser muy complejos o enormes que hagan la aplicación correr lentamente.

### Funcionalidad

- Los diferentes escenarios;
- La galería de armas; y,
- Diferentes personajes entre aliados y enemigos.

### El manejo de recursos

- Imágenes;
- Sonidos; y,
- Videos

### Realismo

- El uso de motores de física para dar una sensación de realismo en el movimiento de los personajes en pantalla.

Este libro se enfoca exclusivamente en videojuegos en 2D en dos tecnologías: Oracle® Java® y Microsoft® XNA®.

El trabajo en 2D es más sencillo para explicar las bases de los videojuegos como animación y las colisiones, y demanda menos recursos de máquina por lo que puede probarse estos programas en computadores de poca potencia en CPU y tarjeta gráfica.

El uso de Java y C# nace porque en un ambiente universitario son los lenguajes de programación más utilizados. Cuando se desarrollan videojuegos con Microsoft XNA se utiliza como lenguaje de programación C#. Una ventaja para el estudiante es que esta tecnología puede ser adquirida en forma gratuita y legal. Tanto Java como Microsoft XNA pueden ser descargados de los sitios oficiales y ser usados sin requerir el pago de licencias. En cuanto a Java, el estudiante puede hacer uso de entornos de desarrollo gratuitos como Eclipse, Netbeans o IntelliJ IDEA Community Edition por nombrar unos ejemplos, en el lado de Microsoft XNA, el estudiante puede descargar el Microsoft Visual Studio 2008 Express Edition (la versión de C#) y luego instalar Microsoft XNA. Es recomendado hacer uso de Microsoft XNA 3.1 porque la versión 4.0 es muy exigente en cuanto a hardware.

El propósito de este libro es dar una introducción al desarrollo de videojuegos en 2D y que el lector tenga conocimiento de los diversos aspectos al momento de programarlos como es la animación (el uso de hilos), el uso de recursos (sonidos, imágenes) y la captura de eventos (leer las entradas por teclado, ratón y gamepad).

Para entender el código de este libro, el lector debe tener conocimientos en Java y C#, la programación orientada a objetos (especialmente la herencia), la implementación de estructuras de datos dinámica (pilas, colas, listas) con esos lenguajes de programación, y estar familiarizado con los formatos de imagen (.bmp, .jpg, .gif) y de sonido (.wav, .mp3).

## 1. El entorno gráfico y las imágenes

### 1.1 Primitiva Gráfica

Para mostrar gráficos en Java (Klawonn, 2008) (Brackeen, y otros, 2003) (Davison, 2005) utilizamos dos clases:

#### IniciaGrafico.java

```
import java.awt.*;

public class IniciaGrafico
{
    static Frame fraContenedor = new Frame("Inicio gráficos");

    public static void main(String[] args)
    {
        //Instancia el objeto que mostrará las gráficas
        Dibujar objDibuja = new Dibujar();

        //Dimensiona la ventana
        fraContenedor.setSize(350, 250);

        //El objeto gráfico queda incluido dentro de la ventana
        fraContenedor.add(objDibuja);

        //Muestra la ventana
        fraContenedor.setVisible(true);

        //El evento de cerrar ventana
        fraContenedor.addWindowListener
        (new java.awt.event.WindowAdapter()
        {public void windowClosing(java.awt.event.WindowEvent e)
        {System.exit(0);}});
    }
}
```

#### Dibujar.java

```
import java.awt.*;

public class Dibujar extends Canvas
{
    public void paint(Graphics objGrafico)
    {
        objGrafico.setColor(Color.blue); //Activa el color azul
        objGrafico.fillOval(20, 20, 40, 40); //Dibuja un óvalo
    }
}
```

La primera clase (IniciaGrafico.java) tendrá a su cargo el inicio de las ventanas, los botones y demás componentes visuales. La segunda clase (Dibujar.java) tendrá a su cargo los gráficos en sí y hereda los atributos y procedimientos de la clase *Canvas*.

Ya definidas las dos clases, el siguiente paso es escribir el código que generará las gráficas en el procedimiento *public void paint (Graphics objGrafico)*. Todas las instrucciones para graficar deben estar dentro de ese procedimiento. Nota: *objGrafico* es un nombre que damos al parámetro de tipo *Graphics*, se puede cambiar ese nombre, por supuesto, teniendo cuidado que todas las instrucciones de allí en adelante tengan el nuevo nombre. En el desarrollo de este libro se usará el nombre *objGrafico*.

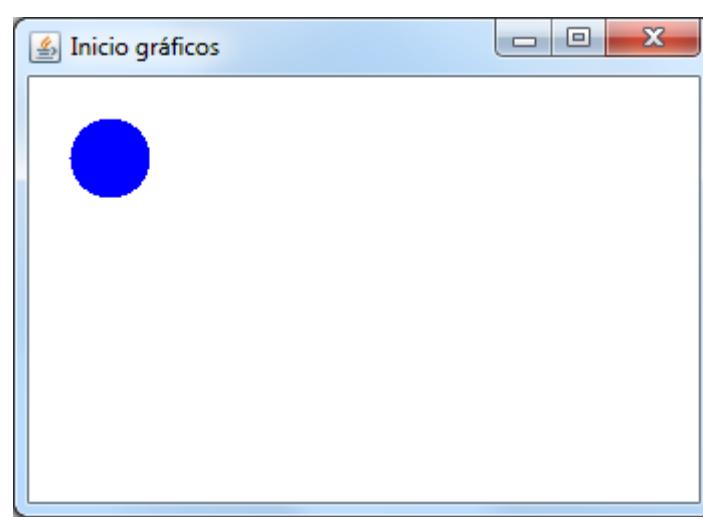
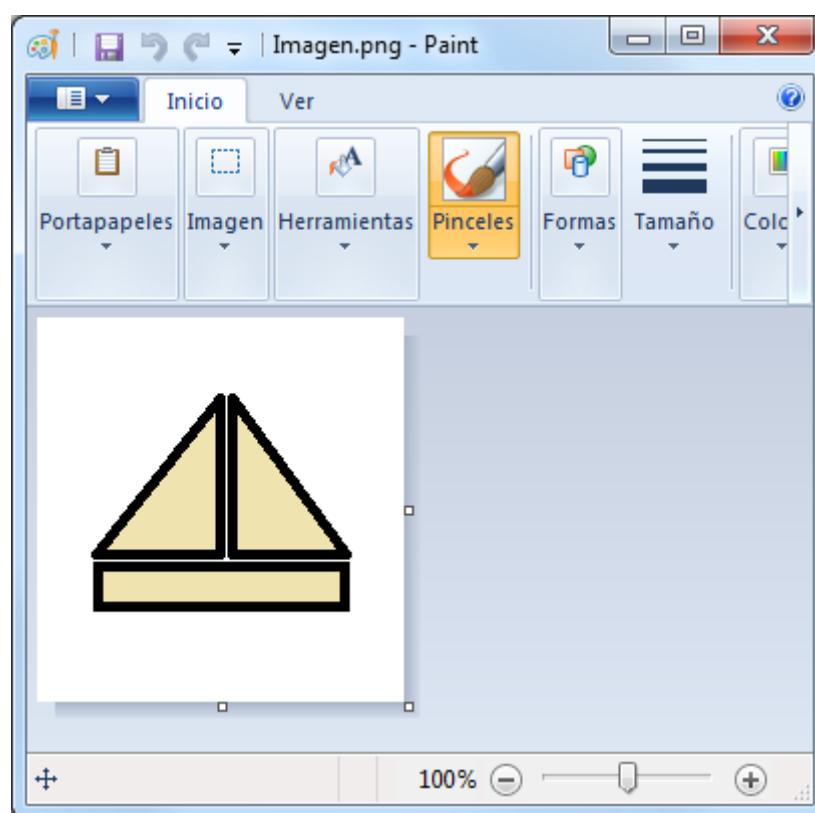


Figura 1: Gráfico generado en aplicación escrita en Java

El procedimiento *public void paint (Graphics objGrafico)* se ejecuta cuando alguna ventana pasa por encima del gráfico (usualmente cuando el usuario arrastra una ventana por la pantalla o simplemente cuando cambia de aplicación activa).

### 1.2 Cargando una imagen

En un videojuego, por lo general no se hacen uso de primitivas gráficas para diseñar escenarios, aliados o enemigos debido a que es demasiado simple lo obtenido por una instrucción y si se desea una representación más compleja requeriría de muchas instrucciones de dibujado lo que haría más lento el videojuego. La solución a esto es el uso de imágenes.



**Figura 2: Imagen hecha con el Paint de Windows**

Esta imagen se puede guardar en diversos formatos: .bmp, .gif, .jpg, .png.

En el ejemplo, se guarda la imagen en formato .png, para cargar y mostrar usamos el siguiente programa dividido en dos clases: *IniciaGrafico* y *Dibujar*

#### IniciaGrafico.java

```
import java.awt.*;

public class IniciaGrafico
{
    static Frame fraContenedor = new Frame("Carga una imagen");

    public static void main(String[] args)
    {
        //Instancia el objeto que mostrará las gráficas
        Dibujar objDibuja = new Dibujar();

        //Dimensiona la ventana
        fraContenedor.setSize(350, 250);

        //El objeto gráfico queda incluido dentro de la ventana
        fraContenedor.add(objDibuja);

        //Muestra la ventana
        fraContenedor.setVisible(true);

        //El evento de cerrar ventana
        fraContenedor.addWindowListener(
            new java.awt.event.WindowAdapter()
            {
                public void windowClosing(java.awt.event.WindowEvent e)
                {System.exit(0);}
            }
        );
    }
}
```

Y la clase que se encarga del entorno gráfico, cargar y mostrar la imagen (que se encuentra en el mismo directorio de las clases).

#### Dibujar.java

```
import java.awt.Image.*;
import java.net.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;

public class Dibujar extends Canvas
{
    //La imagen se guarda en un objeto BufferedImage
    BufferedImage Enemigo;

    //Carga en el evento paint
    public void paint(Graphics objGrafico)
    {
        try
        {
            //Ruta en la que se encuentra el archivo de imagen
            URL RutaImagen = getClass().getResource("Imagen.png");

            //Carga la imagen
            Enemigo = ImageIO.read(RutaImagen);
        }
        catch (Exception objError)
        {

```

```

System.out.println("Error en carga de archivo");
}

//Muestra la imagen en la ventana
objGrafico.drawImage(Enemigo, 0, 0, this);
}
}

```

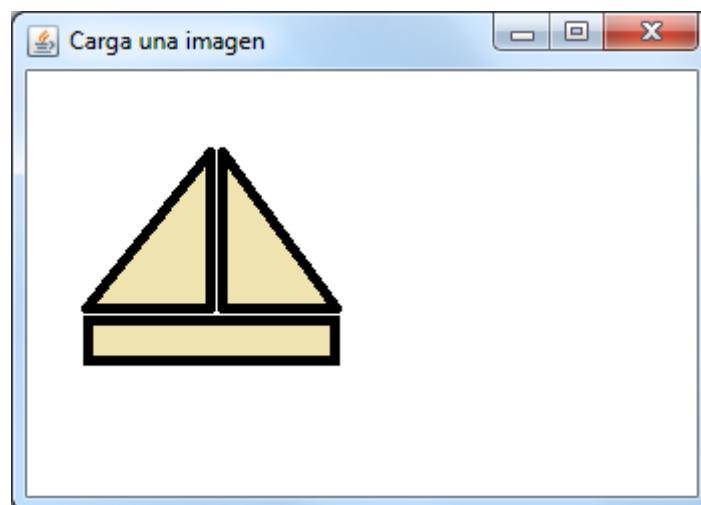


Figura 3: Aplicación en Java: Carga la imagen y la muestra.

### 1.3 Haciendo el código reutilizable

Un videojuego manejará muchas imágenes, luego el código que llama la imagen debe ser reutilizable. Se modifica la clase Dibujar para que tenga un método que se encargue de cargar imágenes.

Dibujar.java

```

import java.awt.Image.*;
import java.net.*;
import java.awt.*;
import java.awt.image	BufferedImage;
import javax.imageio.ImageIO;

public class Dibujar extends Canvas
{
    //Objeto de tipo imagen en el que se guardará en memoria
    //la imagen que se carga desde el medio de persistencia (por ejemplo, disco duro)
    BufferedImage Enemigo;

    //Método que cargará la imagen dada la dirección por parámetro
    public BufferedImage CargaImagen(String Direccion)
    {
        try
        {
            URL RutaImagen = getClass().getResource(Direccion);
            BufferedImage Imagen = ImageIO.read(RutaImagen);

            //Retorna el objeto imagen
            return Imagen;
        }
        catch (Exception objError)
        {
            System.out.println("Problemas en carga de archivo");
            return null;
        }
    }

    public void paint( Graphics objGrafico )
    {
        //Carga la imagen en memoria
        Enemigo = CargaImagen("Imagen.png");

        //Muestra esa imagen en ventana
        objGrafico.drawImage(Enemigo, 0, 0, this);
    }
}

```

### 1.4 Mejorando la carga de imagen

El código anterior tiene el problema que constantemente es llamado por el evento `paint()`. Por ejemplo, cuando una ventana cubra así sea parcialmente la aplicación, el evento `paint()` es ejecutado y este a su vez llamaría el evento de cargar imagen por lo que se harían muchas llamadas de carga de archivo (un proceso lento). Se valida entonces, que si la imagen ya está cargada en memoria no vuelva a llamar la carga de archivo. ¿Cómo se hace eso? Se chequea el objeto `Imagen`, si este es nulo entonces se llama al método `CargaImagen`.

Dibujar.java

```

import java.awt.Image.*;
import java.net.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;

public class Dibujar extends Canvas
{
    BufferedImage Enemigo;

```

```

public BufferedImage CargaImagen(String Direccion)
{
try
{
    URL RutaImagen = getClass().getResource(Direccion);
    BufferedImage Imagen = ImageIO.read(RutaImagen);
    return Imagen;
}
catch (Exception e)
{
    System.out.println("Problemas en carga de archivo");
    return null;
}
}

public void paint( Graphics objGrafico )
{
//Si no estaba cargada la imagen, llama la rutina de carga del medio de persistencia (por ejemplo, disco duro)
if (Enemigo==null)
    Enemigo = CargaImagen("Imagen.png");
objGrafico.drawImage(Enemigo, 0, 0,this);
}
}

```

## 1.5 Mejorando más aún la carga de imagen

El problema con el código anterior es que es necesario chequear constantemente si el objeto imagen está nulo o no. Si se va a usar ese objeto en diversas partes del código, puede ser dispendioso y aumenta las probabilidades de generar errores al no verificar. Una mejor opción es almacenar la imagen en una estructura de datos de tipo HashMap; si se requiere la imagen, se busca en la lista primero y si no se encuentra, entonces se hace el proceso de carga de archivo. ¿Por qué la estructura HashMap? Porque se busca por el nombre de la imagen en la lista.

Necesitamos entonces dos métodos: el primero se encarga de verificar si la imagen ya estaba cargada en la lista, si se da el caso, entonces lo que hace este método es retornar la imagen solicitada, en caso que la imagen no estuviese en lista, se llama al segundo método (que hace la carga de archivo) y una vez la imagen está cargada es puesta en memoria en la lista.

Dibujar.java

```

import java.awt.Image.*;
import java.net.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.util.HashMap;

public class Dibujar extends Canvas
{
//Almacena las imágenes en una lista
HashMap ListaImagenes = new HashMap();

//Carga la imagen del medio de persistencia (por ejemplo, disco duro)
public BufferedImage CargaImagen(String Direccion)
{
try
{
    URL RutaImagen = getClass().getResource(Direccion);
    BufferedImage Imagen = ImageIO.read(RutaImagen);
    return Imagen;
}
catch (Exception e)
{
    System.out.println("Problemas en cargar archivo");
    return null;
}
}

//Si la imagen no está en memoria (verificando la lista)
//la carga del medio de persistencia (por ejemplo, disco duro)
public BufferedImage RetornaImagen(String NombreImagen)
{
//Busca la imagen en la lista
BufferedImage Imagen = (BufferedImage) ListaImagenes.get(NombreImagen);

//Si no encuentra la imagen en la lista, la carga del medio de persistencia (por ejemplo, disco duro)
if (Imagen == null)
{
    Imagen = CargaImagen(NombreImagen);
    ListaImagenes.put(NombreImagen, Imagen);
}
return Imagen;
}

public void paint( Graphics objGrafico )
{
    BufferedImage Enemigo = RetornaImagen("Imagen.png");
    objGrafico.drawImage(Enemigo, 0, 0,this);
}
}

```

## 1.6 Combinando controles con entorno gráfico

Un videojuego puede requerir entradas estándar del usuario, es decir, del entorno gráfico. Para combinar componentes gráficos como botones, cajas de texto (textbox), cajas de chequeo (checkbox), etc., necesitamos que la clase inicial implemente esos controles gráficos y a su vez que limite en un área rectangular aquella parte para gráficos/imágenes. La clase que dibuja no debe variar ninguna instrucción, solo varía la inicial. Usemos el siguiente programa dividido en dos clases: *Iniciar* y *Dibujar*

### Iniciar.java

```
import java.awt.*;

public class Iniciar
{
    static Frame Ventana = new Frame("Combina controles con gráficos");

    //Instancia el objeto que mostrará las gráficas
    static Dibujar objDibuja = new Dibujar();

    //Botones a usar
    static Button btnArriba = new Button();
    static Button btnAbajo = new Button();
    static Button btnIzquierda = new Button();
    static Button btnDerecha = new Button();

    public static void main(String sbParametros[])
    {
        //Dimensiona la ventana
        Ventana.setSize(500, 400);

        //Agrega los botones a la ventana
        btnArriba.setLabel("^");
        btnAbajo.setLabel("V");
        btnIzquierda.setLabel("<<");
        btnDerecha.setLabel(">>");

        Ventana.add(btnArriba, BorderLayout.NORTH);
        Ventana.add(btnAbajo, BorderLayout.SOUTH);
        Ventana.add(btnIzquierda, BorderLayout.WEST);
        Ventana.add(btnDerecha, BorderLayout.EAST);

        //El objeto gráfico queda incluido dentro de la ventana
        Ventana.add(objDibuja);

        //Muestra la ventana
        Ventana.setVisible(true);

        //El evento de cerrar ventana
        Ventana.addWindowListener(
            new java.awt.event.WindowAdapter()
            {public void windowClosing(java.awt.event.WindowEvent e)
            {System.exit(0);}});
    }
}
```

### Dibujar.java

```
import java.awt.Image.*;
import java.net.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.util.HashMap;

public class Dibujar extends Canvas
{
    //Almacena las imágenes en una lista
    HashMap ListaImagenes = new HashMap();

    //Carga la imagen del medio de persistencia (por ejemplo, disco duro)
    public BufferedImage CargaImagen(String Direccion)
    {
        try
        {
            URL RutaImagen = getClass().getResource(Direccion);
            BufferedImage Imagen = ImageIO.read(RutaImagen);
            return Imagen;
        }
        catch (Exception e)
        {
            System.out.println("Problemas en cargar archivo");
            return null;
        }
    }

    //Si la imagen no está en memoria (verificando la lista)
    //la carga del medio de persistencia (por ejemplo, disco duro)
    public BufferedImage RetornaImagen(String NombreImagen)
    {
        //Busca la imagen en la lista
        BufferedImage Imagen = (BufferedImage) ListaImagenes.get(NombreImagen);

        //Si no encuentra la imagen en la lista, la carga del medio de persistencia (por ejemplo, disco duro)
        if (Imagen == null)
```

```

{
    Imagen = CargaImagen(NombreImagen);
    ListaImagenes.put(NombreImagen, Imagen);
}
return Imagen;
}

public void paint( Graphics objGrafico )
{
    BufferedImage Enemigo = RetornaImagen("Imagen.png");
    objGrafico.drawImage(Enemigo, 0, 0, this);
}
}

```

Este es el resultado: una aplicación Java que combina controles gráficos y entorno gráfico.

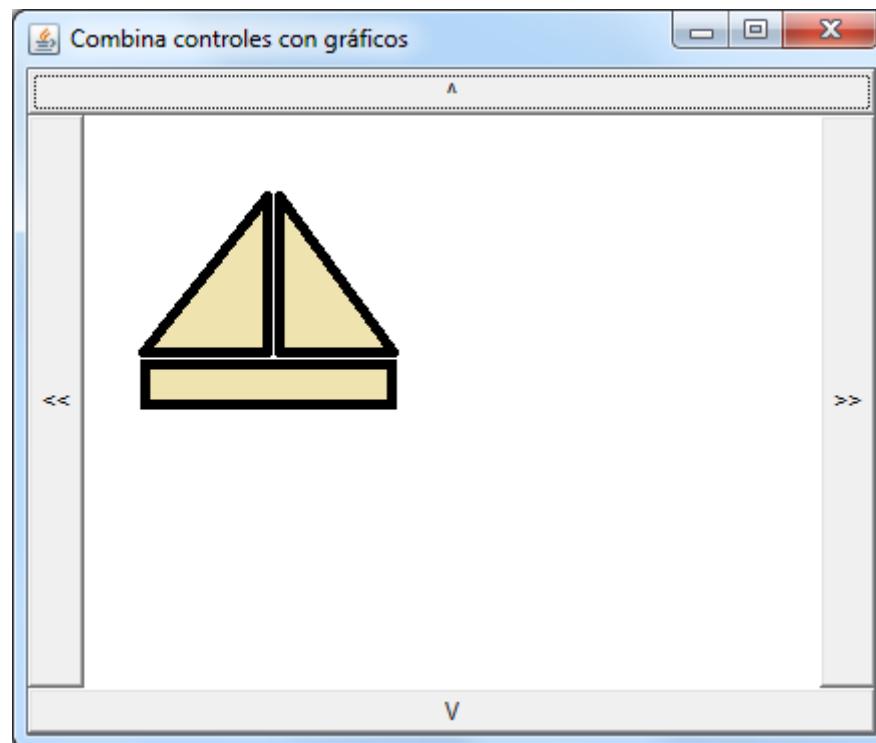


Figura 4: Controles con gráficos

### 1.7 Posicionando los controles gráficos

En el programa anterior, se observa que los botones están prefijados a las esquinas superior, inferior, izquierda y derecha de la ventana, la ventaja de este orden es que se puede redimensionar la ventana y los botones se adaptan al nuevo tamaño, la desventaja es que se pierde control sobre como luce la ventana. Si deseamos controlar mejor la apariencia, debemos entonces desactivar esta característica de "acomodarse" al tamaño. El "layout" de la ventana pasa a ser nulo y se puede controlar posición y tamaño de cada objeto gráfico, incluyendo la parte gráfica.

Este es el programa que requiere dos clases: *Iniciar* y *Dibujar*

```

Iniciar.java
import java.awt.*;

public class Iniciar
{
    static Frame Ventana = new Frame("Controles y gráficos separados");
    static Button btnArriba = new Button();
    static Button btnAbajo = new Button();
    static Button btnDerecha = new Button();
    static Button btnIzquierda = new Button();

    //Esta clase muestra la imagen
    static Dibujar objDibuja = new Dibujar();

    public static void main( String sbParametros[] )
    {
        //Tamaño de la ventana
        Ventana.setSize(450,450);

        //Usa coordenadas absolutas
        Ventana.setLayout(null);

        //Límite rectangular a los gráficos
        objDibuja.setBounds(100,120, 250, 250);

        //Adiciona la clase gráfica a la ventana en forma independiente
        Ventana.add(objDibuja);

        //Atributos botón arriba
        btnArriba.setLabel("^");
        btnArriba.setBounds(35,30,60,30);
        Ventana.add(btnArriba);

        //Atributos botón abajo
        btnAbajo.setLabel("V");
        btnAbajo.setBounds(35,90,60,30);
        Ventana.add(btnAbajo);

        //Atributos botón izquierda
        btnIzquierda.setLabel("<<");
        Ventana.add(btnIzquierda);
    }
}

```

```

btnIzquierda.setBounds(5,60,60,30);
Ventana.add(btnIzquierda);

//Atributos botón derecha
btnDerecha.setLabel(">>");
btnDerecha.setBounds(65,60,60,30);
Ventana.add(btnDerecha);

//Se muestra la ventana
Ventana.setVisible(true);

//El evento de cerrar ventana
Ventana.addWindowListener
(new java.awt.event.WindowAdapter()
{public void windowClosing(java.awt.event.WindowEvent e)
{System.exit(0);}});

}
}

```

**Dibujar.java**

```

import java.awt.Image.*;
import java.net.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.util.HashMap;

public class Dibujar extends Canvas
{
    //Almacena las imágenes en una lista
    HashMap ListaImagenes = new HashMap();

    //Carga la imagen de disco
    public BufferedImage CargaImagen(String Direccion)
    {
        try
        {
            URL RutaImagen = getClass().getResource(Direccion);
            BufferedImage Imagen = ImageIO.read(RutaImagen);
            return Imagen;
        }
        catch (Exception e)
        {
            System.out.println("Problemas en cargar archivo");
            return null;
        }
    }

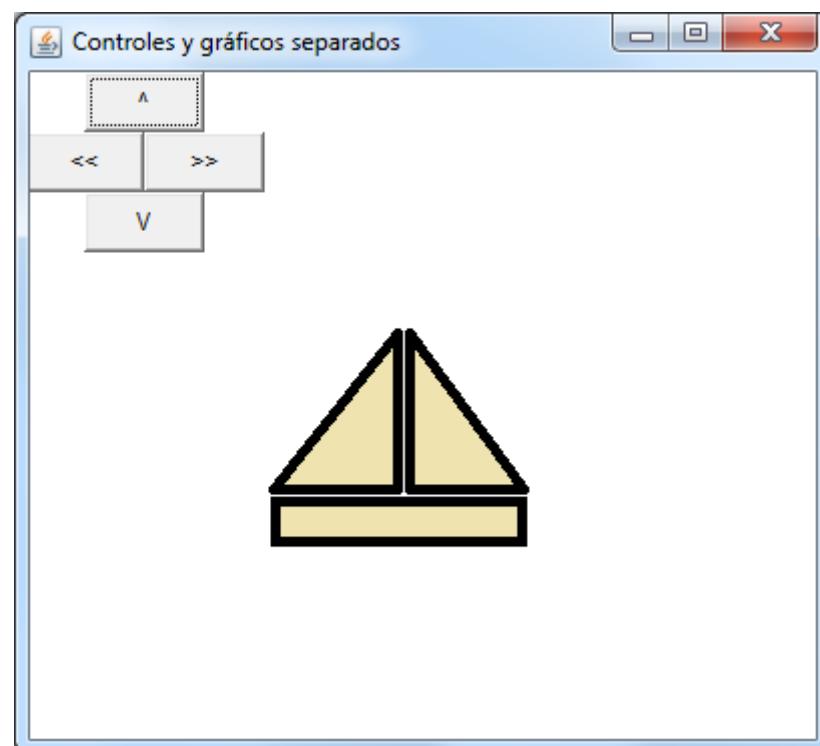
    //Si la imagen no está en memoria (verificando la lista)
    //la carga de disco
    public BufferedImage RetornaImagen(String NombreImagen)
    {
        //Busca la imagen en la lista
        BufferedImage Imagen = (BufferedImage) ListaImagenes.get(NombreImagen);

        //Si no encuentra la imagen en la lista, la carga de disco
        if (Imagen == null)
        {
            Imagen = CargaImagen(NombreImagen);
            ListaImagenes.put(NombreImagen, Imagen);
        }
        return Imagen;
    }

    public void paint( Graphics objGrafico )
    {
        BufferedImage Enemigo = RetornaImagen("Imagen.png");
        objGrafico.drawImage(Enemigo, 0, 0,this);
    }
}

```

El resultado obtenido es este:



**Figura 5: Sin layout. Controles y Gráficos.**

Por ahora, los botones no tienen ninguna acción cuando los presionamos, en el siguiente capítulo, se resuelve este problema.

## 2. Principios de control

En un videojuego es imprescindible que el jugador pueda controlar al protagonista. En este capítulo veremos cómo mover la imagen al oprimir algún botón.

### 2.1 Eventos

Al oprimir un botón generamos un evento, allí damos las instrucciones necesarias para mover al protagonista. Para lograr esto, debemos primero habilitar la captura de eventos de los botones. Luego dentro del evento de cada botón, se programa como mover el protagonista por la pantalla. La recomendación es llamar un método de la clase del protagonista, ese método se encarga de mover la figura y refrescar la pantalla.

Se modifica entonces la clase *Dibujar* para que tenga cuatro(4) nuevos métodos que mueven la imagen hacia arriba, abajo, izquierda y derecha. La imagen entonces se ubica dependiendo del valor de las coordenadas.

Dibujar.java

```
import java.awt.Image.*;
import java.net.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.util.HashMap;

public class Dibujar extends Canvas
{
    //Almacena las imágenes en una lista
    HashMap ListaImagenes = new HashMap();

    //Posición de la imagen
    int PosX, PosY;

    //Métodos para mover la imagen
    public void MueveArriba(){ PosY--; repaint(); }
    public void MueveAbajo(){ PosY++; repaint(); }
    public void MueveIzquierda(){ PosX--; repaint(); }
    public void MueveDerecha(){ PosX++; repaint(); }

    //Carga la imagen de disco
    public BufferedImage CargaImagen(String Direccion)
    {
        try
        {
            URL RutaImagen = getClass().getResource(Direccion);
            BufferedImage Imagen = ImageIO.read(RutaImagen);
            return Imagen;
        }
        catch (Exception e)
        {
            System.out.println("Problemas en cargar archivo");
            return null;
        }
    }

    //Si la imagen no está en memoria (verificando la lista)
    //la carga de disco
    public BufferedImage RetornaImagen(String NombreImagen)
    {
        //Busca la imagen en la lista
        BufferedImage Imagen = (BufferedImage) ListaImagenes.get(NombreImagen);

        //Si no encuentra la imagen en la lista, la carga de disco
        if (Imagen == null)
        {
            Imagen = CargaImagen(NombreImagen);
            ListaImagenes.put(NombreImagen, Imagen);
        }
        return Imagen;
    }

    public void paint( Graphics objGrafico )
    {
        BufferedImage Enemigo = RetornaImagen("Imagen.png");
        objGrafico.drawImage(Enemigo, PosX, PosY, this);
    }
}
```

En la clase Iniciar se añade el código de los eventos de cada botón

Iniciar.java

```
import java.awt.*;

public class Iniciar
{
    static Frame Ventana = new Frame("Controles y gráficos separados");
    static Button btnArriba = new Button();
    static Button btnAbajo = new Button();
    static Button btnDerecha = new Button();
    static Button btnIzquierda = new Button();
```

```

//Esta clase muestra la imagen
static Dibujar objDibuja = new Dibujar();

public static void main( String sbParametros[] )
{
    //Tamaño de la ventana
    Ventana.setSize(450,450);

    //Usa coordenadas absolutas
    Ventana.setLayout(null);

    //Límites de cada una de las clases gráficas
    objDibuja.setBounds(100,120, 250, 250);

    //Adiciona la clase gráfica a la ventana en forma independiente
    Ventana.add(objDibuja);

    //Atributos botón arriba
    btnArriba.setLabel("^");
    btnArriba.setBounds(35,30,60,30);
    Ventana.add(btnArriba);

    //Atributos botón abajo
    btnAbajo.setLabel("V");
    btnAbajo.setBounds(35,90,60,30);
    Ventana.add(btnAbajo);

    //Atributos botón izquierda
    btnIzquierda.setLabel("<<");
    btnIzquierda.setBounds(5,60,60,30);
    Ventana.add(btnIzquierda);

    //Atributos botón derecha
    btnDerecha.setLabel(">>");
    btnDerecha.setBounds(65,60,60,30);
    Ventana.add(btnDerecha);

    //Se muestra la ventana
    Ventana.setVisible(true);

    //El evento de cerrar ventana
    Ventana.addWindowListener(
        (new java.awt.event.WindowAdapter()
        {public void windowClosing(java.awt.event.WindowEvent e)
        {System.exit(0);}}));

    //Manejo de eventos. Cuando hacemos clic en botón abajo
    btnArriba.addActionListener(new java.awt.event.ActionListener()
    {public void actionPerformed(java.awt.event.ActionEvent e)
    {
        objDibuja.MueveArriba();
    }
});

    //Manejo de eventos. Cuando hacemos clic en botón arriba
    btnAbajo.addActionListener(new java.awt.event.ActionListener()
    {public void actionPerformed(java.awt.event.ActionEvent e)
    {
        objDibuja.MueveAbajo();
    }
});

    //Manejo de eventos. Cuando hacemos clic en botón izquierda
    btnIzquierda.addActionListener(new java.awt.event.ActionListener()
    {public void actionPerformed(java.awt.event.ActionEvent e)
    {
        objDibuja.MueveIzquierda();
    }
});

    //Manejo de eventos. Cuando hacemos clic en botón derecha
    btnDerecha.addActionListener(new java.awt.event.ActionListener()
    {public void actionPerformed(java.awt.event.ActionEvent e)
    {
        objDibuja.MueveDerecha();
    }
});
}
}

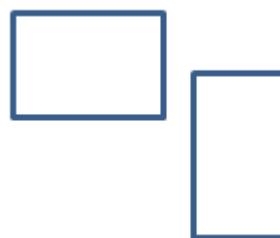
```

### 3. Colisiones

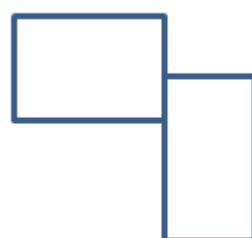
#### 3.1 Definición

En todo videojuego, se valida cuando un objeto gráfico entra en contacto con otro objeto gráfico, por ejemplo, cuando nuestro protagonista toca una pared o golpea un balón. El término técnico es colisión.

Observemos las siguientes imágenes que nos muestra cuando se produce una colisión.



**Figura 6: Dos rectángulos. No hay colisión.**



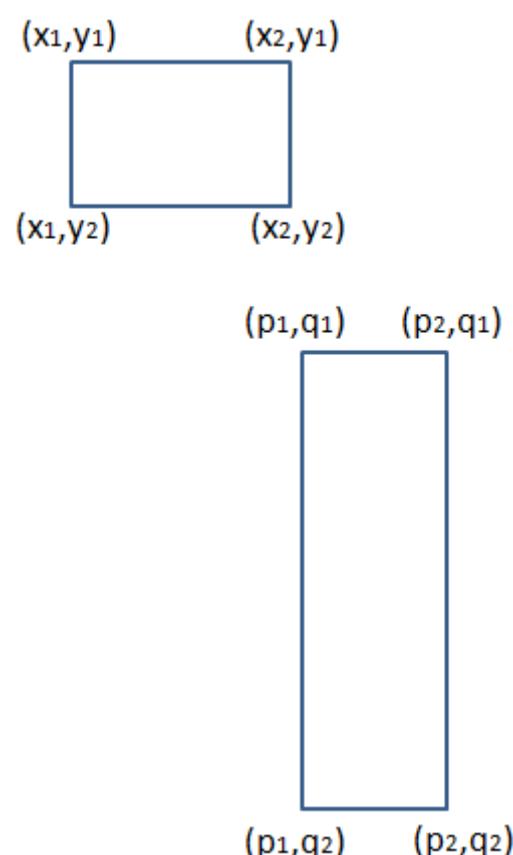
**Figura 7: Dos rectángulos. Hay colisión.**

Nuestra labor es ver en qué momento hubo una colisión.

Algunos lenguajes de programación ofrecen funciones o procedimientos para detectar colisiones, por lo que tenemos ahorrado ese código. Sin embargo, hay que entender en qué consiste la detección de colisiones porque si nos confiamos en esas funciones o procedimientos que trae el lenguaje de programación, corremos el riesgo que estos sean muy limitados para comportamientos más realistas o sofisticados.

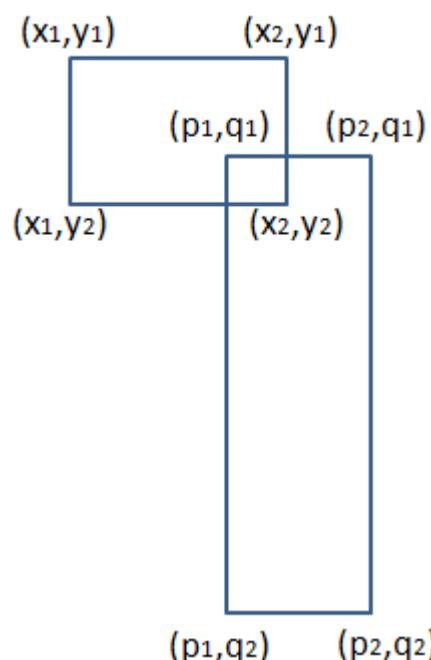
#### 3.2 Validando una colisión entre dos objetos gráficos

Una primera aproximación a las colisiones, es en 2D, entre objetos rectangulares. Observemos los dos rectángulos de la figura. En cada uno se detalla las coordenadas de cada vértice. ¿Cómo detectamos una colisión entre esos rectángulos?



**Figura 8: Coordenadas de vértices de dos rectángulos**

Empecemos con ver si el vértice  $(x_2, y_2)$  queda dentro de los límites del rectángulo de la parte inferior como se ve en la figura inferior.

**Figura 9: Colisión entre dos rectángulos**

Recordando que el valor del eje Y va creciendo en dirección hacia abajo en entornos gráficos.

La instrucción es:

Si ( $x_2 >= p_1$ ) y ( $x_2 <= p_2$ ) y ( $y_2 >= q_1$ ) y ( $y_2 <= q_2$ ) entonces "hay colisión"

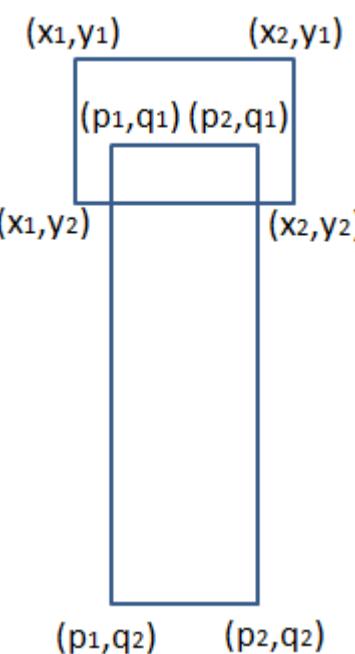
Lo mismo sucedería con los otros vértices del rectángulo. Estas serían las otras tres(3) instrucciones:

Si ( $x_1 >= p_1$ ) y ( $x_1 <= p_2$ ) y ( $y_1 >= q_1$ ) y ( $y_1 <= q_2$ ) entonces "hay colisión"

Si ( $x_2 >= p_1$ ) y ( $x_2 <= p_2$ ) y ( $y_1 >= q_1$ ) y ( $y_1 <= q_2$ ) entonces "hay colisión"

Si ( $x_1 >= p_1$ ) y ( $x_1 <= p_2$ ) y ( $y_2 >= q_1$ ) y ( $y_2 <= q_2$ ) entonces "hay colisión"

Con eso sabríamos si el rectángulo que se encuentra en la parte superior de la figura en algún momento colisiona con el rectángulo en la parte inferior. Pero hay un problema en el siguiente caso:

**Figura 10: Un caso de colisión**

En este caso, las cuatro instrucciones condicionales vistas anteriormente no detectan la colisión, por lo que se requiere validar si los cuatro vértices del rectángulo inferior colisionan con el rectángulo superior. Se necesitarían otras cuatro instrucciones que serían:

Si ( $p_1 >= x_1$ ) y ( $p_1 <= x_2$ ) y ( $q_1 >= y_1$ ) y ( $q_1 <= y_2$ ) entonces "hay colisión"

Si ( $p_2 >= x_1$ ) y ( $p_2 <= x_2$ ) y ( $q_1 >= y_1$ ) y ( $q_1 <= y_2$ ) entonces "hay colisión"

Si ( $p_1 >= x_1$ ) y ( $p_1 <= x_2$ ) y ( $q_2 >= y_1$ ) y ( $q_2 <= y_2$ ) entonces "hay colisión"

Si ( $p_2 >= x_1$ ) y ( $p_2 <= x_2$ ) y ( $q_2 >= y_1$ ) y ( $q_2 <= y_2$ ) entonces "hay colisión"

En otras palabras, si necesitamos saber si hay colisión entre los dos rectángulos necesitamos en total ocho instrucciones condicionales. Se podría decir que estas ocho instrucciones se pueden abbreviar en una sola usando la conjunción lógica "O" pero por motivos de legibilidad se mantendrán esas ocho instrucciones en el programa que se transcribe a continuación que requiere dos clases: Iniciar y Dibujar

Iniciar.java

```
import java.awt.*;

public class Iniciar
{
    static Frame Ventana = new Frame("Colisiones");
    static Button btnArriba = new Button();
    static Button btnAbajo = new Button();
    static Button btnDerecha = new Button();
    static Button btnIzquierda = new Button();
    static TextField txtTexto = new TextField("0");
}
```

```

//Esta clase muestra la imagen
static Dibujar objDibuja = new Dibujar();

public static void main( String sbParametros[] )
{
    //Tamaño de la ventana
    Ventana.setSize(450,450);

    //Usa coordenadas absolutas
    Ventana.setLayout(null);

    //Límites de cada una de las clases gráficas
    objDibuja.setBounds(100,130, 250, 250);

    //Adiciona la clase gráfica a la ventana en forma independiente
    Ventana.add(objDibuja);

    //Atributos botón arriba
    btnArriba.setLabel("^");
    btnArriba.setBounds(35,40,60,30);
    Ventana.add(btnArriba);

    //Atributos botón abajo
    btnAbajo.setLabel("V");
    btnAbajo.setBounds(35,100,60,30);
    Ventana.add(btnAbajo);

    //Atributos botón izquierda
    btnIzquierda.setLabel("<<");
    btnIzquierda.setBounds(5,70,60,30);
    Ventana.add(btnIzquierda);

    //Atributos botón derecha
    btnDerecha.setLabel(">>");
    btnDerecha.setBounds(65,70,60,30);
    Ventana.add(btnDerecha);

    //Atributos caja de texto
    txtTexto.setText("NO se detectó colisión");
    txtTexto.setBounds(100, 380, 250, 30);
    Ventana.add(txtTexto);

    //Se muestra la ventana
    Ventana.setVisible(true);

    //El evento de cerrar ventana
    Ventana.addWindowListener
    (new java.awt.event.WindowAdapter()
     {public void windowClosing(java.awt.event.WindowEvent e)
      {System.exit(0);}});

    //Manejo de eventos. Cuando hacemos clic en botón abajo
    btnArriba.addActionListener(new java.awt.event.ActionListener()
    {
        public void actionPerformed(java.awt.event.ActionEvent e)
        {
            objDibuja.MueveArriba();
            ValidaColision();
        }
    });

    //Manejo de eventos. Cuando hacemos clic en botón arriba
    btnAbajo.addActionListener(new java.awt.event.ActionListener()
    {
        public void actionPerformed(java.awt.event.ActionEvent e)
        {
            objDibuja.MueveAbajo();
            ValidaColision();
        }
    });

    //Manejo de eventos. Cuando hacemos clic en botón izquierda
    btnIzquierda.addActionListener(new java.awt.event.ActionListener()
    {
        public void actionPerformed(java.awt.event.ActionEvent e)
        {
            objDibuja.MueveIzquierda();
            ValidaColision();
        }
    });

    //Manejo de eventos. Cuando hacemos clic en botón derecha
    btnDerecha.addActionListener(new java.awt.event.ActionListener()
    {
        public void actionPerformed(java.awt.event.ActionEvent e)
        {
            objDibuja.MueveDerecha();
            ValidaColision();
        }
    });

    static void ValidaColision()
    {
        if (objDibuja.ChequeaColision())
            txtTexto.setText("Hubo colisión");
        else
    }
}

```

```

    txtTexto.setText("NO se detectó colisión");
}

}

```

## Dibujar.java

```

import java.awt.*;

public class Dibujar extends Canvas
{
    int X1, Y1; //Posición del rectángulo móvil
    int P1, Q1; //Posición del rectángulo estático

    //Base y altura del rectángulo 1
    int BaseRect1, AlturaRect1;

    //Base y altura del rectángulo 2
    int BaseRect2, AlturaRect2;

    //Métodos para mover el rectángulo
    public void MueveArriba(){ Y1-=2; repaint(); }
    public void MueveAbajo(){ Y1+=2; repaint(); }
    public void MueveIzquierda(){ X1-=2; repaint(); }
    public void MueveDerecha(){ X1+=2; repaint(); }

    public Dibujar()
    {
        X1=50; Y1=60;
        P1=90; Q1=70;
        BaseRect1 = 20; AlturaRect1 = 20;
        BaseRect2 = 50; AlturaRect2 = 80;
    }

    public void paint( Graphics objGrafico )
    {
        //Pinta el fondo de la parte gráfica
        objGrafico.setColor(Color.white);
        objGrafico.fillRect(0, 0, getWidth(), getHeight());

        //Objeto que mueve el usuario
        objGrafico.setColor(Color.black);
        objGrafico.drawRect(X1, Y1, BaseRect1, AlturaRect1);

        //Objeto estático
        objGrafico.setColor(Color.red);
        objGrafico.drawRect(P1, Q1, BaseRect2, AlturaRect2);
    }

    boolean ChequeaColision()
    {
        //Deduce el punto (X2, Y2) del rectángulo 1
        int X2, Y2;
        X2 = X1 + BaseRect1;
        Y2 = Y1 + AlturaRect1;

        //Deduce el punto (P2, Q2) del rectángulo 2
        int P2, Q2;
        P2 = P1 + BaseRect2;
        Q2 = Q1 + AlturaRect2;

        //Chequea si hay colisión
        if ( X1>=P1 && X1<=P2 && Y1>=Q1 && Y1<=Q2 ) return true;
        if ( X2>=P1 && X2<=P2 && Y1>=Q1 && Y1<=Q2 ) return true;
        if ( X1>=P1 && X1<=P2 && Y2>=Q1 && Y2<=Q2 ) return true;
        if ( X2>=P1 && X2<=P2 && Y2>=Q1 && Y2<=Q2 ) return true;

        if ( P1>=X1 && P1<=X2 && Q1>=Y1 && Q1<=Y2 ) return true;
        if ( P2>=X1 && P2<=X2 && Q1>=Y1 && Q1<=Y2 ) return true;
        if ( P1>=X1 && P1<=X2 && Q2>=Y1 && Q2<=Y2 ) return true;
        if ( P2>=X1 && P2<=X2 && Q2>=Y1 && Q2<=Y2 ) return true;

        //No se detectó colisión
        return false;
    }
}

```

El método `ChequeaColision()` de la clase `Dibujar` es la que valida si los dos rectángulos colisionan, retorna `true` si se da la colisión, `false` en caso contrario.

El resultado de ejecutar ese programa se muestra a continuación:

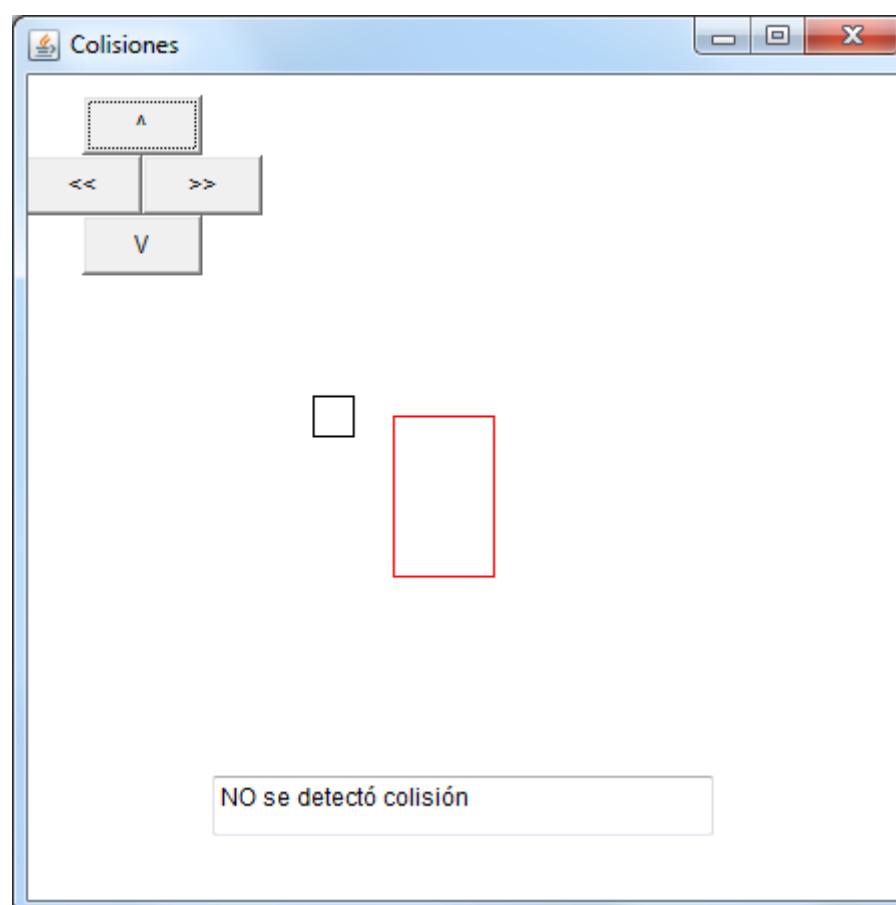


Figura 11: No hay colisión entre los rectángulos

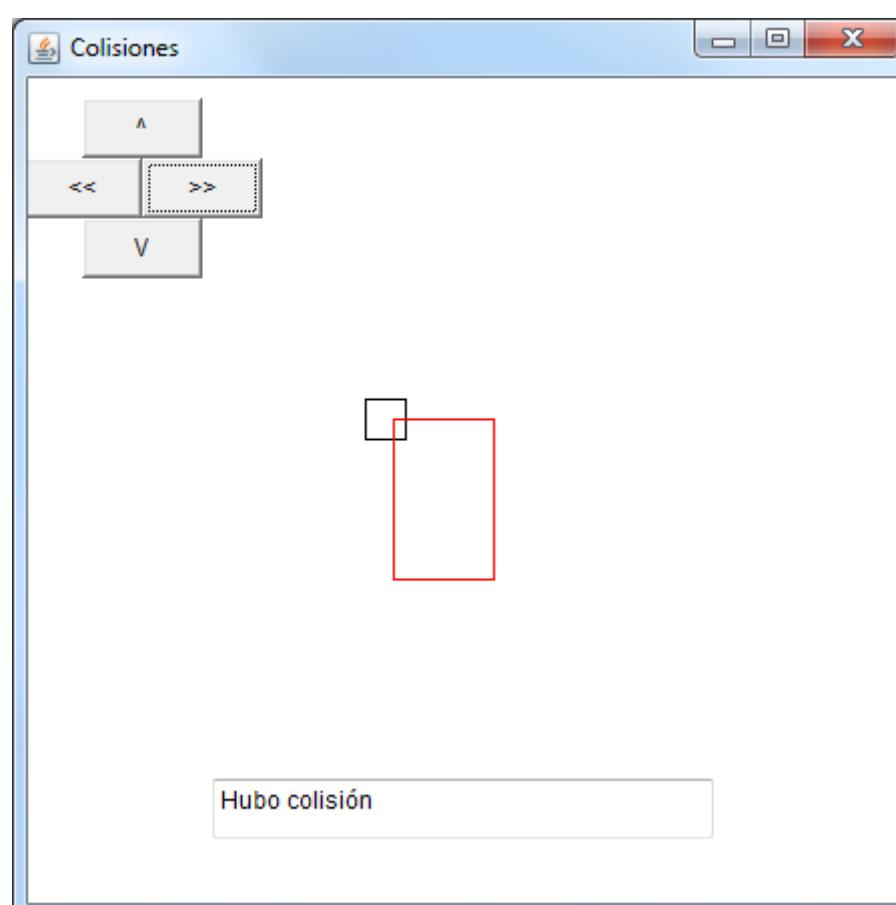


Figura 12: Hay colisión entre los rectángulos

### 3.3 Haciendo más sencillo el validar colisiones

Hay una forma más sencilla de validar la colisión entre dos rectángulos y es haciendo uso de la API de Java, la instrucción es *intersects*. Se modifica entonces la clase *Dibujar* en el método *ChequeaColision*.

Dibujar.java

```
import java.awt.*;

public class Dibujar extends Canvas
{
    int X1, Y1; //Posición del rectángulo móvil
    int P1, Q1; //Posición del rectángulo estático

    //Base y altura del rectángulo 1
    int BaseRect1, AlturaRect1;

    //Base y altura del rectángulo 2
    int BaseRect2, AlturaRect2;

    //Métodos para mover el rectángulo
    public void MueveArriba(){ Y1-=2; repaint(); }
    public void MueveAbajo(){ Y1+=2; repaint(); }
    public void MueveIzquierda(){ X1-=2; repaint(); }
    public void MueveDerecha(){ X1+=2; repaint(); }

    public Dibujar()
    {
        X1=50; Y1=60;
        P1=90; Q1=70;
        BaseRect1 = 20; AlturaRect1 = 20;
    }
}
```

```

BaseRect2 = 50; AlturaRect2 = 80;
}

public void paint( Graphics objGrafico )
{
    //Pinta el fondo de la parte gráfica
    objGrafico.setColor(Color.white);
    objGrafico.fillRect(0, 0, getWidth(), getHeight());

    //Objeto que mueve el usuario
    objGrafico.setColor(Color.black);
    objGrafico.drawRect(X1, Y1, BaseRect1, AlturaRect1);

    //Objeto estático
    objGrafico.setColor(Color.red);
    objGrafico.drawRect(P1, Q1, BaseRect2, AlturaRect2);
}

boolean ChequeaColision()
{
    //Deduce los rectángulos para detectar si hay colisión
    //entre ellos.
    Rectangle Rect1 = new Rectangle(X1-1,Y1-1,BaseRect1+2,AlturaRect1+2);
    Rectangle Rect2 = new Rectangle(P1-1,Q1-1,BaseRect2+2,AlturaRect2+2);

    //Verifica si hay colisión
    if (Rect1.intersects(Rect2)) return true;

    //No se detectó colisión
    return false;
}
}

```

Notaremos al ejecutar el programa que cuando los rectángulos solo se tocan no hay colisión para la función intersects, solo hay colisión cuando hay una intersección entre rectángulos. Si deseáramos que con sólo tocarse los rectángulos se reportara una colisión, podríamos simplemente hacer que el rectángulo de detección fuese dos pixeles más ancho y largo, las instrucciones a cambiar en la clase Dibujar, método ChequeaColision serían:

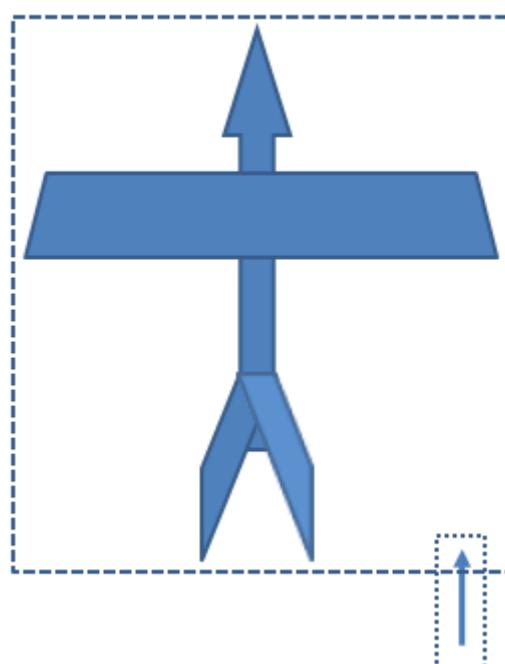
```

Rectangle Rect1 = new Rectangle(X1-1,Y1-1,BaseRect1+2,AlturaRect1+2);
Rectangle Rect2 = new Rectangle(P1-1,Q1-1,BaseRect2+2,AlturaRect2+2);

```

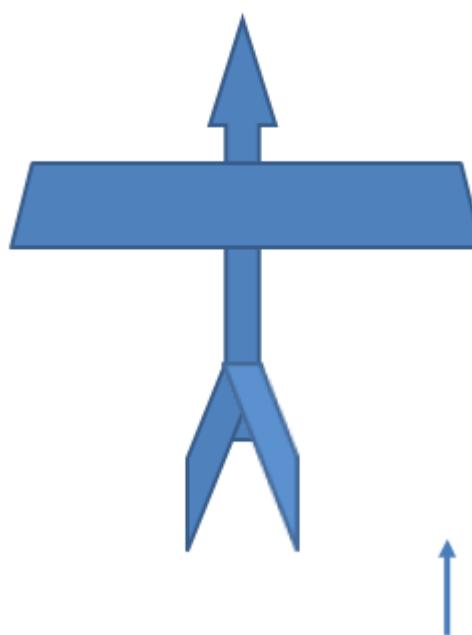
### 3.4 Consideraciones sobre las colisiones

Observemos la siguiente figura en el que nuestro protagonista es un avión, se ha dibujado un rectángulo alrededor de este con líneas discontinuas que representa el rectángulo que valida la colisión de nuestro avión. Hay un misil que lo persigue y hemos puesto también un rectángulo alrededor de este que representa el rectángulo que valida la colisión. Como podemos apreciar hay una colisión entre los dos rectángulos y el programa mostrará que el avión ha sido destruido.



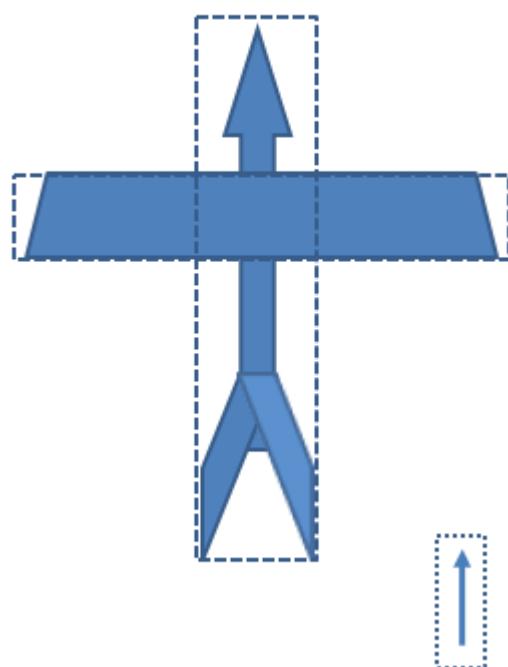
**Figura 13: Mostrando rectángulos de colisión**

El problema es que el jugador no ve esos rectángulos que validan la colisión, lo que ve es esto:



**Figura 14: Lo que el jugador realmente ve**

Y de repente el avión es destruido. El jugador pensará que hay un gran error en el juego y perderá interés. Luego para hacer más realista el juego (actualmente se pide más realismo a los juegos) debemos usar más de un rectángulo de validación de colisión como se observa en la figura siguiente. En ese caso se verá más realista la colisión.



**Figura 15: Mas rectángulos de colisión dan más realismo**

El problema aquí es que entre más rectángulos de detección de colisión tengan los protagonistas del juego, mayor será el número de instrucciones necesarias y por lo tanto mayor el tiempo requerido por la CPU para evaluar todas esas instrucciones con el inconveniente que el juego se vuelva más lento.

## 4. Animación

### 4.1 Uso de hilos para hacer animación

En una aplicación escrita en Java, está el hilo principal de AWT; este hilo se encarga de la ventana y los controles gráficos, luego si necesitamos hacer animación por aparte (por ejemplo, al oprimir un botón queremos ver que un triángulo se desplace) es necesario generar un segundo hilo (independiente del hilo AWT) para programarle las rutinas gráficas de animación.

Para dar el efecto de animación, debemos dibujar la pantalla, luego borrarla y redibujarla de nuevo con los objetos ligeramente modificados (en posición, tamaño o forma), el proceso se repite continuamente. Para lograr esto, definimos un hilo, luego damos la orden de iniciar ejecución del hilo, este ejecuta el método `run()` hasta finalizarlo, una vez termina de ejecutar `run()` el hilo muere.

En el siguiente programa compuesto por dos clases: `InicioAnimacion` y `Animar` se describe el ejemplo de uso de hilos. Observemos que el hilo es iniciado desde el `main()`

`InicioAnimacion.java`

```
import java.awt.*;
public class InicioAnimacion
{
    //La ventana
    static Frame fraContenedor = new Frame("Principios de animación");

    //Objeto que muestra el gráfico animado
    static Animar objAnimar = new Animar();

    public static void main(String[] args)
    {
        //Tamaño de la ventana
        fraContenedor.setSize(600, 600);

        //Adiciona el objeto gráfico a la ventana
        fraContenedor.add(objAnimar);

        //Inicia el hilo
        objAnimar.Iniciar();

        //Muestra la ventana
        fraContenedor.setVisible(true);

        //El evento de cerrar ventana
        fraContenedor.addWindowListener(
            new java.awt.event.WindowAdapter()
            {public void windowClosing(java.awt.event.WindowEvent e)
            {System.exit(0);}});
    }
}
```

A continuación la clase que aplica la animación

`Animar.java`

```
import java.awt.*;
import java.util.*;

//Para usar hilos debe heredar de Runnable
public class Animar extends Canvas implements Runnable
{
    //Hilo que controla la animación
    Thread objHilo;

    //Posición del círculo relleno
    int PosX=180, PosY=10;

    //Desplazamiento del círculo
    int IncX=1, IncY=1;

    //Diámetro del círculo
    int Diametro = 40;

    //Método ejecutado por el hilo
    public void run()
    {
        for(int Cont=1; Cont<=10000; Cont++)
        {
            repaint(); //Repinta la pantalla

            try{Thread.sleep(5);}catch (InterruptedException E){}
        }
    }

    public void paint (Graphics objGrafico)
    {
        //Mueve el círculo
        PosX+=IncX;
        PosY+=IncY;

        //Dibuja el círculo
        objGrafico.setColor(Color.blue);
    }
}
```

```

objGrafico.fillOval(PosX, PosY, Diametro, Diametro);

//Chequea si rebota contra una pared.
if (PosX<=0 || PosX+Diametro >= getSize().width) IncX*=-1;
if (PosY<=0 || PosY+Diametro >= getSize().height) IncY*=-1;
}

public void Iniciar()
{
  if (objHilo == null)
  {
    objHilo = new Thread(this);
    objHilo.start();
  }
}
}

```

La clase *Animar* hereda de *Runnable* con la instrucción “implements Runnable”, esto es necesario para usar hilos. El hilo es definido al inicio con la instrucción:

```
Thread objHilo;
```

El hilo es puesto en ejecución por el programa inicial (*InicioAnimacion*) cuando ejecuta la instrucción *objAnimar.Iniciar()*; el método *Iniciar()* está implementado en *Animar* y lo que hace es instanciar e inicializar.

El hilo ejecutará el método *run()*, una vez que ese método termine, el hilo muere.

## 4.2 Deteniendo y reiniciando el hilo

En el ejemplo anterior, la animación se detenía cuando se hacían mil iteraciones, el cambio que haremos es el siguiente: un botón detiene ese círculo (finalizando el hilo) y otro botón se encarga de reiniciar el hilo (volviendo a ejecutar *run()* )

*InicioAnimacion.java*

```

import java.awt.*;

public class InicioAnimacion
{
  //La ventana
  static Frame Ventana = new Frame("Principios de animación");

  //Botones que controlan el inicio y fin del hilo
  static Button btnIniciar = new Button();
  static Button btnDetener = new Button();

  //Objeto que muestra el gráfico animado
  static Animar objAnimar = new Animar();

  public static void main(String sbParametros[])
  {
    //Tamaño de la ventana
    Ventana.setSize(600,600);

    //Adiciona los botones
    btnIniciar.setText("Iniciar o Continuar");
    btnDetener.setText("Detener");
    Ventana.add(btnIniciar, BorderLayout.NORTH);
    Ventana.add(btnDetener, BorderLayout.SOUTH);

    //Adiciona el objeto gráfico a la ventana
    Ventana.add(objAnimar);

    //Muestra la ventana
    Ventana.setVisible(true);

    //El evento de cerrar ventana
    Ventana.addWindowListener(
      new java.awt.event.WindowAdapter()
      {public void windowClosing(java.awt.event.WindowEvent e)
       {System.exit(0);}});

    //Manejo de eventos. Cuando hacemos clic en botón Iniciar
    btnIniciar.addActionListener(new java.awt.event.ActionListener(){
      public void actionPerformed(java.awt.event.ActionEvent e)
      {
        objAnimar.Iniciar();
      }
    });

    //Manejo de eventos. Cuando hacemos clic en botón Detener
    btnDetener.addActionListener(new java.awt.event.ActionListener(){
      public void actionPerformed(java.awt.event.ActionEvent e)
      {
        objAnimar.Detener();
      }
    });
  }
}

```

```

import java.awt.*;
import java.util.*;

//Para usar hilos debe heredar de Runnable
public class Animar extends Canvas implements Runnable
{
    //Hilo que controla la animación
    Thread objHilo;

    //Controla que run() continue
    boolean Ejecuta = false;

    //Posición del círculo relleno
    int PosX=180, PosY=10;

    //Desplazamiento del círculo
    int IncX=1, IncY=1;

    //Diámetro del círculo
    int Diametro = 40;

    //Método ejecutado por el hilo
    public void run()
    {
        while(Ejecuta)
        {
            repaint(); //Repinta la pantalla

            try{Thread.sleep(5);}catch (InterruptedException E){}
        }
    }

    public void paint (Graphics objGrafico)
    {
        //Mueve el círculo
        PosX+=IncX;
        PosY+=IncY;

        //Dibuja el círculo
        objGrafico.setColor(Color.blue);
        objGrafico.fillOval(PosX, PosY, Diametro, Diametro);

        //Chequea si rebota contra una pared.
        if (PosX<=0 || PosX+Diametro >= getSize().width) IncX*=-1;
        if (PosY<=0 || PosY+Diametro >= getSize().height) IncY*=-1;
    }

    public void Iniciar()
    {
        if (Ejecuta==false)
        {
            objHilo = new Thread(this);
            Ejecuta = true;
            objHilo.start();
        }
    }

    public void Detener()
    {
        Ejecuta = false;
    }
}

```

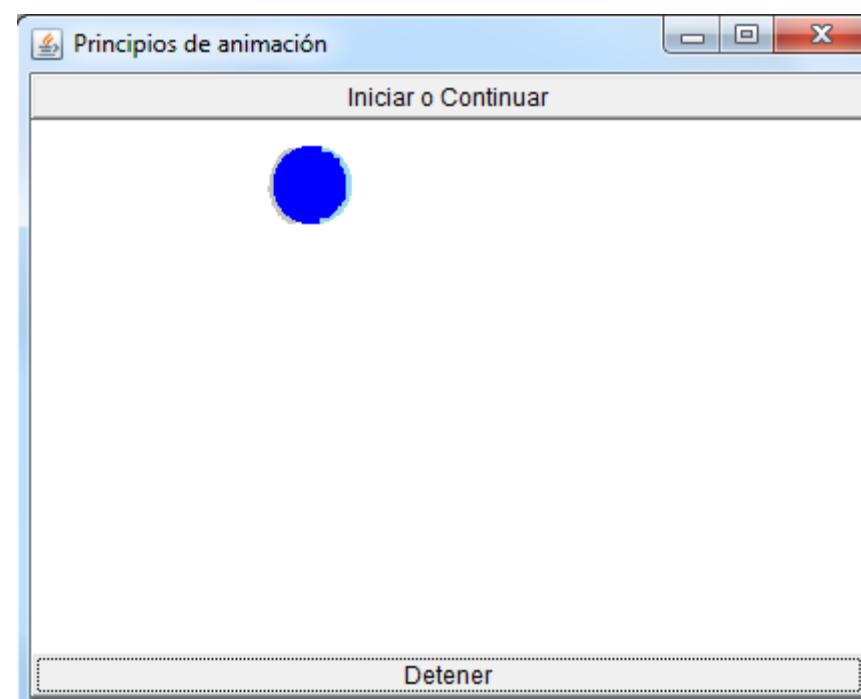


Figura 16: Controlando la animación

#### 4.3 Minimizando el titileo de la animación (técnica de double buffering)

En el ejemplo anterior se empieza a notar un efecto visual realmente molesto, y es que el círculo comienza a titilar mientras se desplaza. La razón de este comportamiento es que en cada `repaint()`, la parte gráfica se borra y luego se vuelve a dibujar el círculo. Ese proceso es detectado por nuestros ojos y es interpretado como si la figura titilase.

Para quitar ese efecto existe una técnica llamada double buffering, que consiste en hacer todo el procesamiento gráfico en una sección no visible y una vez se termina de procesar, el resultado es enviado en forma instantánea a la sección visible.

A continuación el programa anterior agregando las líneas de double buffering en la clase Animar

Animar.java

```
import java.awt.*;
import java.util.*;

public class Animar extends Canvas implements Runnable
{
    //Hilo que controla la animación
    Thread objHilo;

    //Controla que run() continue
    boolean Ejecuta = false;

    //Posición del círculo relleno
    int PosX=180, PosY=10;

    //Desplazamiento del círculo
    int IncX=1, IncY=1;

    //Diámetro del círculo
    int Diametro = 40;

    //Variables para el double buffering
    private Image imgPantalla;
    private Graphics graPantalla;

    //Método ejecutado por el hilo
    public void run()
    {
        while(Ejecuta)
        {
            repaint(); //Repinta la pantalla
            try{Thread.sleep(5);}catch (InterruptedException E){}
        }
    }

    public void paint (Graphics objGrafico)
    {
        //Activa el double buffering
        if (imgPantalla == null)
        {
            imgPantalla = createImage(getSize().width, getSize().height);
            graPantalla = imgPantalla.getGraphics();
        }

        //Fondo blanco para borrar pantalla
        graPantalla.setColor(Color.white);
        graPantalla.fillRect(0, 0, getSize().width, getSize().height);

        //Mueve el círculo
        PosX+=IncX;
        PosY+=IncY;

        //Dibuja el círculo
        graPantalla.setColor(Color.blue);
        graPantalla.fillOval(PosX, PosY, Diametro, Diametro);

        //Chequea si rebota contra una pared.
        if (PosX<=0 || PosX+Diametro >= getSize().width) IncX*=-1;
        if (PosY<=0 || PosY+Diametro >= getSize().height) IncY*=-1;

        //Muestra lo hecho en graPantalla al usuario
        objGrafico.drawImage(imgPantalla, 0, 0, this);
    }

    //Evita que se borre toda la ventana cada vez que hace repaint()
    public void update(Graphics objGrafico)
    {
        paint( objGrafico );
    }

    public void Iniciar()
    {
        if (Ejecuta==false)
        {
            objHilo = new Thread(this);
            Ejecuta = true;
            objHilo.start();
        }
    }

    public void Detener()
    
```

```
{
    Ejecuta = false;
}
}
```

En primer lugar se agregan dos atributos a la clase

```
//Variables para el double buffering
private Image imgPantalla;
private Graphics graPantalla;
```

Luego en el método *paint()* se modifica para activar el double buffering

```
//Activa el double buffering
if (imgPantalla == null)
{
    imgPantalla = createImage(getSize().width, getSize().height);
    graPantalla = imgPantalla.getGraphics();
}
```

De allí en adelante, todas las operaciones gráficas se hacen sobre *graPantalla* (la región no visible). Una de las primeras operaciones es borrar la parte gráfica.

```
//Fondo blanco para borrar pantalla
graPantalla.setColor(Color.white);
graPantalla.fillRect(0, 0, getSize().width, getSize().height);
```

Observemos que las operaciones gráficas ya son sobre *graPantalla*, por ejemplo:

```
//Dibuja el círculo
graPantalla.setColor(Color.blue);
graPantalla.fillOval(PosX, PosY, Diametro, Diametro);
```

Y la última línea del método *paint()* es la que copia el contenido de la región no visible a la región visible:

```
//Muestra lo hecho en graPantalla al usuario
objGrafico.drawImage(imgPantalla, 0, 0, this);
```

Debemos adicionar un método que es *update()* para que cuando se haga el *repaint()*, no se borre la pantalla visible y así se evita el titileo de las figuras, el borrado ya lo hicimos en *graPantalla*

```
//Evita que se borre toda la ventana cada vez que hace repaint()
public void update(Graphics objGrafico)
{
    paint( objGrafico );
}
```

El resultado salta a la vista, el círculo relleno se desplaza con suavidad, sin titilar.

#### 4.4 Mejorando la técnica para minimizar el titileo de la animación

En el ejemplo anterior tuvimos que modificar varias instrucciones de *paint()* para poder hacer uso del doblé buffering. ¿Hay alguna manera de hacerlo más sencillo? La respuesta es si y sólo es modificar un método, el *update()*. No se requiere hacer ningún cambio adicional.

Este es el código:

```
Animar.java
import java.awt.*;
import java.util.*;

//Para usar hilos debe heredar de Runnable
public class Animar extends Canvas implements Runnable
{
    //Hilo que controla la animación
    Thread objHilo;

    //Controla que run() continue
    boolean Ejecuta = false;

    //Posición del círculo relleno
    int PosX=180, PosY=10;

    //Desplazamiento del círculo
    int IncX=1, IncY=1;

    //Diámetro del círculo
    int Diametro = 40;

    //Método ejecutado por el hilo
    public void run()
    {
        while(Ejecuta)
        {
            repaint(); //Repinta la pantalla
            try{Thread.sleep(5);}catch (InterruptedException E){}
        }
    }
}
```

```

public void paint (Graphics objGrafico)
{
    //Mueve el círculo
    PosX+=IncX;
    PosY+=IncY;

    //Dibuja el círculo
    objGrafico.setColor(Color.blue);
    objGrafico.fillOval(PosX, PosY, Diametro, Diametro);

    //Chequea si rebota contra una pared.
    if (PosX<=0 || PosX+Diametro >= getSize().width) IncX*=-1;
    if (PosY<=0 || PosY+Diametro >= getSize().height) IncY*=-1;
}

//Se implementa aquí el double buffering
public void update(Graphics objGrafico)
{
    //Área de graficado no visible
    Graphics NoVisible;

    //Imagen
    Image Imagen = null;

    //Tamaño de la ventana
    Dimension Tamano = size();

    //Crea el buffer de Imagen y lo asocia
    //al área de graficado no visible
    Imagen = createImage(Tamano.width, Tamano.height);
    NoVisible = Imagen.getGraphics();

    //Borra el fondo
    NoVisible.setColor(getBackground());
    NoVisible.fillRect(0, 0, Tamano.width, Tamano.height);
    NoVisible.setForeground();

    //Llama a paint pero con el objeto gráfico no visible
    paint(NoVisible);

    //Pinta la imagen al área visible
    objGrafico.drawImage(Imagen, 0, 0, this);
}
}

public void Iniciar()
{
    if (Ejecuta==false)
    {
        objHilo = new Thread(this);
        Ejecuta = true;
        objHilo.start();
    }
}

public void Detener()
{
    Ejecuta = false;
}
}

```

Como podemos observar, el método `paint()` no necesita ningún cambio, seguimos con las instrucciones gráficas normales, tampoco es necesario adicionar atributos a la clase. El programa solo requiere este nuevo método:

```

//Se implementa aquí el double buffering
public void update(Graphics objGrafico)
{
    //Área de graficado no visible
    Graphics NoVisible;

    //Imagen
    Image Imagen = null;

    //Tamaño de la ventana
    Dimension Tamano = size();

    //Crea el buffer de Imagen y lo asocia
    //al área de graficado no visible
    Imagen = createImage(Tamano.width, Tamano.height);
    NoVisible = Imagen.getGraphics();

    //Borra el fondo
    NoVisible.setColor(getBackground());
    NoVisible.fillRect(0, 0, Tamano.width, Tamano.height);
    NoVisible.setForeground();

    //Llama a paint pero con el objeto gráfico no visible
    paint(NoVisible);

    //Pinta la imagen al área visible
    objGrafico.drawImage(Imagen, 0, 0, this);
}

```

{}

## 4.5 Optimizando la técnica que minimiza el titiloe de la animación

El double buffering de los ejemplos anteriores requiere dibujar una imagen en un área no visible y después mover esa imagen al área visible. Ese procedimiento de mover es muy costoso en tiempo de CPU y entre más grande sea la ventana, más costoso será. La solución a esto es mover solo la parte que requiere ser redibujada. La instrucción es getClipRect().

El cambio se hace en el método update() de Animar.java

Animar.java

```

import java.awt.*;
import java.util.*;

//Para usar hilos debe heredar de Runnable
public class Animar extends Canvas implements Runnable
{
    //Hilo que controla la animación
    Thread objHilo;

    //Controla que run() continue
    boolean Ejecuta = false;

    //Posición del círculo relleno
    int PosX=180, PosY=10;

    //Desplazamiento del círculo
    int IncX=1, IncY=1;

    //Diámetro del círculo
    int Diametro = 40;

    //Método ejecutado por el hilo
    public void run()
    {
        while(Ejecuta)
        {
            repaint(); //Repinta la pantalla

            try{Thread.sleep(5);}catch (InterruptedException E){}
        }
    }

    public void paint (Graphics objGrafico)
    {
        //Mueve el círculo
        PosX+=IncX;
        PosY+=IncY;

        //Dibuja el círculo
        objGrafico.setColor(Color.blue);
        objGrafico.fillOval(PosX, PosY, Diametro, Diametro);

        //Chequea si rebota contra una pared.
        if (PosX<=0 || PosX+Diametro >= getSize().width) IncX*=-1;
        if (PosY<=0 || PosY+Diametro >= getSize().height) IncY*=-1;
    }

    //Se implementa aquí el double buffering
    public void update(Graphics objGrafico)
    {
        //Área de graficado no visible
        Graphics NoVisible;

        //Imagen
        Image Imagen = null;

        //Rectángulo donde se actualiza la parte gráfica
        Rectangle Rectangulo = objGrafico.getClipRect();

        //Crea el buffer de Imagen y lo asocia
        //al área de graficado no visible
        Imagen = createImage(Rectangulo.width, Rectangulo.height);
        NoVisible = Imagen.getGraphics();

        //Borra el fondo
        NoVisible.setColor(getBackground());
        NoVisible.fillRect(0, 0, Rectangulo.width, Rectangulo.height);
        NoVisible.setColor(getForeground());

        //Llama a paint pero con el objeto gráfico no visible
        NoVisible.translate(-Rectangulo.x, -Rectangulo.y);
        paint(NoVisible);

        //Pinta la imagen al área visible
        objGrafico.drawImage(Imagen, 0, 0, this);
    }

    public void Iniciar()
}

```

```
{  
    if (Ejecuta==false)  
    {  
        objHilo = new Thread(this);  
        Ejecuta = true;  
        objHilo.start();  
    }  
}  
  
public void Detener()  
{  
    Ejecuta = false;  
}  
}
```

## 5. Captura de eventos de teclado y ratón

Necesitamos poder controlar el movimiento de nuestro protagonista en el juego. A diferencia de otro tipo de software (por ejemplo, un sistema de información automatizado) que espera la entrada del usuario para poder tomar luego una acción, en un videojuego la acción es continua con o sin entrada del usuario.

### 5.1 Implementando la captura de eventos

El primer paso para hacer la captura de eventos del ratón y teclado es heredar de las clases MouseMotionListener, MouseListener y KeyListener con la instrucción:

```
public class Animar extends Canvas implements Runnable, MouseMotionListener, MouseListener, KeyListener
```

Luego se debe activar que el programa esté atento al teclado y ratón. Para lograr eso debemos en el constructor escribir las instrucciones de activación, este es el código:

```
//Constructor. Activa la escucha de eventos
public Animar()
{
    addMouseMotionListener(this);
    addMouseListener(this);
    addKeyListener(this);
}
```

Finalmente es implementar los métodos de cada tipo de evento y escribir las instrucciones que se requieran dentro de cada método.

```
public void mouseMoved(MouseEvent objMouse){}
public void mouseDragged(MouseEvent objMouse){}

public void mousePressed (MouseEvent objMouse){}
public void mouseClicked (MouseEvent objMouse){}
public void mouseEntered (MouseEvent objMouse){}
public void mouseExited (MouseEvent objMouse){}
public void mouseReleased (MouseEvent objMouse){}

public void keyTyped(KeyEvent e){}
public void keyPressed(KeyEvent e){}
public void keyReleased(KeyEvent e){}
```

En el ejemplo a continuación, el círculo relleno persigue al cursor del ratón a donde se mueva este.

#### Animar.java

```
import java.awt.*;
import java.awt.event.*;

//Para usar hilos debe heredar de Runnable
//Para capturar eventos de ratón debe heredar de
//MouseMotionListener, MouseListener
//Para capturar eventos de teclado debe heredar de KeyListener
public class Animar extends Canvas implements Runnable, MouseMotionListener, MouseListener, KeyListener
{
    //Hilo que controla la animación
    Thread objHilo;

    //Controla que run() continue
    boolean Ejecuta = false;

    //Posición del círculo relleno
    int PosX=180, PosY=10;

    //Desplazamiento del círculo
    int IncX=1, IncY=1;

    //Diámetro del círculo
    int Diametro = 40;

    //Posición del mouse
    int PosXMouse, PosYMouse;

    //Constructor. Activa la escucha de eventos
    public Animar()
    {
        addMouseMotionListener(this);
        addMouseListener(this);
        addKeyListener(this);
    }

    //Método ejecutado por el hilo
    public void run()
    {
        while(Ejecuta)
        {
            repaint(); //Repinta la pantalla
            try{Thread.sleep(2);}catch (InterruptedException E){}
        }
    }

    public void paint (Graphics objGrafico)
    {
        //Mueve el círculo
        PosX+=IncX;
        PosY+=IncY;
    }
}
```

```

PosY+=IncY;

//Dibuja el círculo
objGrafico.setColor(Color.blue);
objGrafico.fillOval(PosX, PosY, Diametro, Diametro);

//Persigue al Mouse
if (PosXMouse < PosX) IncX = -1; else IncX = 1;
if (PosYMouse < PosY) IncY = -1; else IncY = 1;
}

//Se implementa aquí el double buffering
public void update(Graphics objGrafico)
{
    //Área de graficado no visible
    Graphics NoVisible;

    //Imagen
    Image Imagen = null;

    //Rectángulo donde se actualiza la parte gráfica
    Rectangle Rectangulo = objGrafico.getClipRect();

    //Crea el buffer de Imagen y lo asocia
    //al área de graficado no visible
    Imagen = createImage(Rectangulo.width, Rectangulo.height);
    NoVisible = Imagen.getGraphics();

    //Borra el fondo
    NoVisible.setColor(getBackground());
    NoVisible.fillRect(0, 0, Rectangulo.width, Rectangulo.height);
    NoVisible.setForeground();

    //Llama a paint pero con el objeto gráfico no visible
    NoVisible.translate(-Rectangulo.x, -Rectangulo.y);
    paint(NoVisible);

    //Pinta la imagen al área visible
    objGrafico.drawImage(Imagen, 0, 0, this);
}

public void Iniciar()
{
    if (Ejecuta==false)
    {
        objHilo = new Thread(this);
        Ejecuta = true;
        objHilo.start();
    }
}

public void Detener()
{
    Ejecuta = false;
}

//De MouseMotionListener
public void mouseMoved(MouseEvent objMouse)
{
    PosXMouse = (int) objMouse.getPoint().getX();
    PosYMouse = (int) objMouse.getPoint().getY();
}

public void mouseDragged(MouseEvent objMouse){}

//De MouseListener
public void mousePressed(MouseEvent objMouse){}
public void mouseClicked(MouseEvent objMouse){}
public void mouseEntered(MouseEvent objMouse){}
public void mouseExited(MouseEvent objMouse){}
public void mouseReleased(MouseEvent objMouse){}

//De KeyListener
public void keyTyped(KeyEvent e){}
public void keyPressed(KeyEvent e){}
public void keyReleased(KeyEvent e){}
}

```

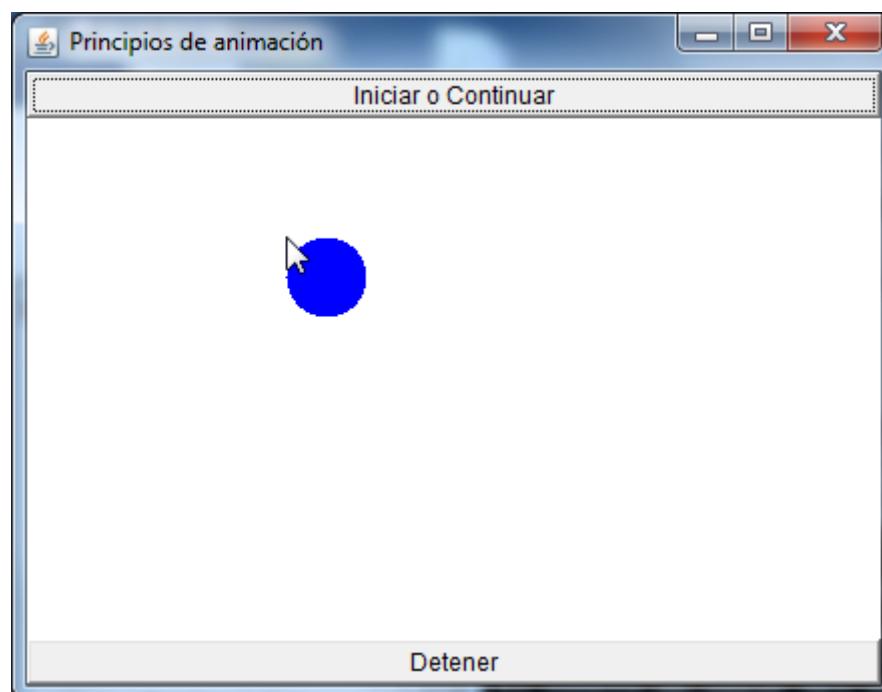


Figura 17: Círculo persiguiendo el cursor

## 6. Uso de sonidos y música de fondo

En un videojuego encontramos sonidos cuando se da un evento (por ejemplo cuando el protagonista acierta en un blanco, cuando se dispara, cuando un enemigo muere, etc.) y la música de fondo o ambiental.

### 6.1 Inicio

Tomando como modelo el código visto en el capítulo “1.5 Mejorando más aún la carga de imagen” se modifica para que cargue y ejecute sonidos. La clase principal activa el hilo para que cada cierto tiempo este ejecute el sonido.

#### IniciaSonido.java

```
import java.awt.*;

public class IniciaSonido
{
    static Frame fraContenedor = new Frame("Carga un sonido");

    public static void main(String sbParametros[])
    {
        //Instancia el objeto que ejecutará los sonidos
        UsoSonido objSonido = new UsoSonido();

        //Dimensiona la ventana
        fraContenedor.setSize(350,250);

        //El objeto queda incluido dentro de la ventana
        fraContenedor.add(objSonido);

        //Muestra la ventana
        fraContenedor.setVisible(true);

        //Inicia el hilo
        objSonido.Iniciar();

        //El evento de cerrar ventana
        fraContenedor.addWindowListener
        (new java.awt.event.WindowAdapter()
        {public void windowClosing(java.awt.event.WindowEvent e)
        {System.exit(0);}});
    }
}
```

#### UsoSonido.java

```
import java.applet.AudioClip;
import java.applet.Applet;
import java.util.*;
import java.awt.*;
import java.net.*;

public class UsoSonido extends Canvas implements Runnable
{
    //Hilo que controla la animación
    Thread objHilo;

    //Almacena los sonidos en una lista
    HashMap ListaSonidos = new HashMap();

    //Generador de números aleatorios
    Random Azar = new Random();

    //Carga el sonido de disco
    public AudioClip CargaSonido(String Direccion)
    {
        try
        {
            URL RutaSonido = getClass().getResource(Direccion);
            AudioClip Sonido = Applet.newAudioClip(RutaSonido);
            return Sonido;
        }
        catch (Exception e)
        {
            System.out.println("Problemas en cargar archivo");
            return null;
        }
    }

    //Si el sonido no está en memoria (verificando la lista)
    //lo carga de disco
    public AudioClip RetornaSonido(String NombreSonido)
    {
        //Busca el sonido en la lista
        AudioClip Sonido = (AudioClip) ListaSonidos.get(NombreSonido);

        //Si no encuentra el sonido en la lista, lo carga de disco
        if (Sonido == null)
        {
            Sonido = CargaSonido(NombreSonido);
            ListaSonidos.put(NombreSonido, Sonido);
        }
    }
}
```

```

    return Sonido;
}

//Método ejecutado por el hilo
public void run()
{
    for(int Cont=1; Cont<=10; Cont++)
    {
        repaint(); //Repinta la pantalla

        //Carga el sonido (si es necesario)
        AudioClip SonidoEjecuta = RetornaSonido("Prueba.wav");
        SonidoEjecuta.play();

        try{Thread.sleep(5000);}catch (InterruptedException E){}
    }
}

public void paint( Graphics objGrafico )
{
    //Un color al azar en formato RGB
    int Rojo = Math.abs(Azar.nextInt())%255;
    int Verde = Math.abs(Azar.nextInt())%255;
    int Azul = Math.abs(Azar.nextInt())%255;

    //Activa ese color al azar
    objGrafico.setColor(new Color(Rojo, Verde, Azul));

    //Fondo en ese color
    objGrafico.fillRect(0, 0, getSize().width, getSize().height);
}

//Inicia el hilo
public void Iniciar()
{
    if (objHilo == null)
    {
        objHilo = new Thread(this);
        objHilo.start();
    }
}
}

```

Los sonidos son gestionados por objetos AudioClip, la carga del sonido del disco se hace con:

```
AudioClip Sonido = Applet.newAudioClip(RutaSonido);
```

El sonido es ejecutado con los métodos play() que lo hace una vez y loop() que lo mantiene ejecutando constantemente, útil para música de fondo.

## 6.2 Ejemplo de animación y sonido

En el siguiente ejemplo, tendremos un código completo (requiere dos clases) en el cual se muestra un círculo relleno rebotando en las paredes de la ventana y se produce un sonido cada vez que se da el rebote.

InicioAnimacion.java

```

import java.awt.*;

public class InicioAnimacion
{
    //La ventana
    static Frame Ventana = new Frame("Principios de animación");

    //Botones que controlan el inicio y fin del hilo
    static Button btnIniciar = new Button();
    static Button btnDetener = new Button();

    //Objeto que muestra el gráfico animado
    static Animar objAnimar = new Animar();

    public static void main(String sbParametros[])
    {
        //Tamaño de la ventana
        Ventana.setSize(600,600);

        //Adiciona los botones
        btnIniciar.setLabel("Iniciar o Continuar");
        btnDetener.setLabel("Detener");
        Ventana.add(btnIniciar, BorderLayout.NORTH);
        Ventana.add(btnDetener, BorderLayout.SOUTH);

        //Adiciona el objeto gráfico a la ventana
        Ventana.add(objAnimar);

        //Muestra la ventana
        Ventana.setVisible(true);

        //El evento de cerrar ventana
        Ventana.addWindowListener
        (new java.awt.event.WindowAdapter()
        {public void windowClosing(java.awt.event.WindowEvent e)
        {System.exit(0);}})
    }
}

```

```
//Manejo de eventos. Cuando hacemos clic en botón Iniciar
btnIniciar.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e)
    {
        objAnimar.Iniciar();
    }
});

//Manejo de eventos. Cuando hacemos clic en botón Detener
btnDetener.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e)
    {
        objAnimar.Detener();
    }
});
```

Y la segunda clase que se encarga de cargar el sonido, hacer la animación y emitir el sonido cada vez que hay un rebote.

#### Animar.java

```
import java.net.*;
import java.awt.*;
import java.util.*;
import java.applet.AudioClip;
import java.applet.Applet;
import java.util.HashMap;

//Para usar hilos debe heredar de Runnable
public class Animar extends Canvas implements Runnable
{
    //Hilo que controla la animación
    Thread objHilo;

    //Controla que run() continue
    boolean Ejecuta = false;

    //Posición del círculo relleno
    int PosX=180, PosY=10;

    //Desplazamiento del círculo
    int IncX=1, IncY=1;

    //Diámetro del círculo
    int Diametro = 40;

    //Almacena los sonidos en una lista
    HashMap ListaSonidos = new HashMap();

    //Sonido del rebote
    AudioClip SonidoEjecuta;

    //Carga el sonido de disco
    public AudioClip CargaSonido(String Direccion)
    {
        try
        {
            URL RutaSonido = getClass().getResource(Direccion);
            AudioClip Sonido = Applet.newAudioClip(RutaSonido);
            return Sonido;
        }
        catch (Exception e)
        {
            System.out.println("Problemas en cargar archivo");
            return null;
        }
    }

    //Si el sonido no está en memoria (verificando la lista)
    //lo carga de disco
    public AudioClip RetornaSonido(String NombreSonido)
    {
        //Busca el sonido en la lista
        AudioClip Sonido = (AudioClip) ListaSonidos.get(NombreSonido);

        //Si no encuentra el sonido en la lista, lo carga de disco
        if (Sonido == null)
        {
            Sonido = CargaSonido(NombreSonido);
            ListaSonidos.put(NombreSonido, Sonido);
        }
        return Sonido;
    }

    //Método ejecutado por el hilo
    public void run()
    {
        SonidoEjecuta = RetornaSonido("rebote.wav");
        while(Ejecuta)
```

```

{
    repaint(); //Repinta la pantalla
    try{Thread.sleep(2);}catch (InterruptedException E){}
}

public void paint (Graphics objGrafico)
{
    //Mueve el círculo
    PosX+=IncX;
    PosY+=IncY;

    //Dibuja el círculo
    objGrafico.setColor(Color.blue);
    objGrafico.fillOval(PosX, PosY, Diametro, Diametro);

    //Chequea si rebota contra una pared.
    if (PosX<=0 || PosX+Diametro >= getSize().width)
    {
        IncX*=-1;
        SonidoEjecuta.play();
    }

    if (PosY<=0 || PosY+Diametro >= getSize().height)
    {
        IncY*=-1;
        SonidoEjecuta.play();
    }
}

//Se implementa aquí el double buffering
public void update(Graphics objGrafico)
{
    //Area de graficado no visible
    Graphics NoVisible;

    //Imagen
    Image Imagen = null;

    //Tamaño de la ventana
    Dimension Tamano = size();

    //Crea el buffer de Imagen y lo asocia
    //al área de graficado no visible
    Imagen = createImage(Tamano.width, Tamano.height);
    NoVisible = Imagen.getGraphics();

    //Borra el fondo
    NoVisible.setColor(getBackground());
    NoVisible.fillRect(0, 0, Tamano.width, Tamano.height);
    NoVisible.setForeground();

    //Llama a paint pero con el objeto gráfico no visible
    paint(NoVisible);

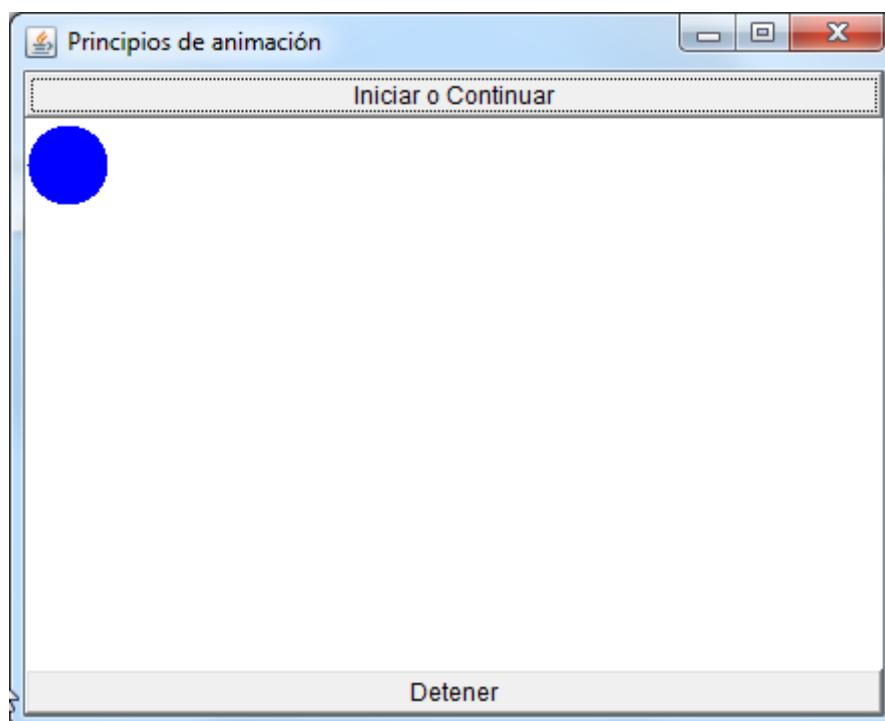
    //Pinta la imagen al área visible
    objGrafico.drawImage(Imagen, 0, 0, this);
}
}

//Inicia el hilo
public void Iniciar()
{
    if (Ejecuta==false)
    {
        objHilo = new Thread(this);
        Ejecuta = true;
        objHilo.start();
    }
}

//Detiene el hilo
public void Detener()
{
    Ejecuta = false;
}
}

```

Por supuesto, debemos tener en el mismo directorio el archivo de sonido a ejecutar.



**Figura 18: Un círculo relleno rebota en las paredes produciendo un sonido en ese momento**

## 7. Organizando el código

Como podemos observar, estamos aumentando cada vez más las líneas de código por lo que va a volverse más complicado y confuso agregar nuevas funcionalidades a nuestro juego. Por lo tanto, debemos reorganizar el código en clases lógicamente separadas. Una clase se dedica a iniciar la aplicación (InicioAnimacion), otra clase se encarga del hilo de ejecución del juego (Animar), una tercera clase se encarga de cargar los recursos (GestionRecursos) y una última clase es la del protagonista (Protagonista).

La clase del protagonista es interesante porque encapsula la carga de imágenes, sonidos, la lógica y el dibujado de este.

Este es el código completo.

### InicioAnimacion.java

```
/* Esta clase se encarga de lanzar la aplicación,
dibuja los botones y activa sus eventos */

import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class InicioAnimacion {
    //La ventana
    static Frame Ventana = new Frame("Principios de animación");

    //Botones que controlan el inicio y fin del hilo
    static Button btnIniciar = new Button();
    static Button btnDetener = new Button();

    //Objeto que muestra el gráfico animado
    static Animar objAnimar = new Animar();

    public static void main(String sbParametros[])
    {
        //Tamaño de la ventana
        Ventana.setSize(600,600);

        //Adiciona los botones
        btnIniciar.setText("Iniciar o Continuar");
        btnDetener.setText("Detener");
        Ventana.add(btnIniciar, BorderLayout.NORTH);
        Ventana.add(btnDetener, BorderLayout.SOUTH);

        //Adiciona el objeto gráfico a la ventana
        Ventana.add(objAnimar);

        //Muestra la ventana
        Ventana.setVisible(true);

        //El evento de cerrar ventana
        Ventana.addWindowListener(
            new java.awt.event.WindowAdapter()
            {public void windowClosing(java.awt.event.WindowEvent e)
            {System.exit(0);}});
    }

    //Manejo de eventos. Cuando hacemos clic en botón Iniciar
    btnIniciar.addActionListener(new java.awt.event.ActionListener(){
        public void actionPerformed(java.awt.event.ActionEvent e)
        {
            objAnimar.Iniciar();
        }
    });

    //Manejo de eventos. Cuando hacemos clic en botón Detener
    btnDetener.addActionListener(new java.awt.event.ActionListener(){
        public void actionPerformed(java.awt.event.ActionEvent e)
        {
            objAnimar.Detener();
        }
    });
}
}
```

### Animar.java

```
/* Esa clase se encarga del hilo de ejecución del videojuego */
import java.awt.*;
import java.util.*;

//Para usar hilos debe heredar de Runnable
public class Animar extends Canvas implements Runnable
{
    //Hilo que controla la animación
    Thread objHilo;

    //Controla el hilo
    boolean Ejecuta = false;

    //Instancia al protagonista
    Protagonista objProtagonista = new Protagonista();

    //Método ejecutado por el hilo
    public void run()
    {
        while(Ejecuta)
        {
            repaint();
            try{Thread.sleep(1000/60);}
            catch(InterruptedException ex){ex.printStackTrace();}
        }
    }
}
```

```

{
    while(Ejecuta)
    {
        repaint();
        try{Thread.sleep(2);}catch (InterruptedException E){}
    }
}

//Pinta al protagonista
public void paint (Graphics objGrafico)
{
    objProtagonista.Pintar(objGrafico, getSize().width, getSize().height);
}

//Se implementa aquí el double buffering
public void update(Graphics objGrafico)
{
    //Area de graficado no visible
    Graphics NoVisible;

    //Imagen
    Image Imagen = null;

    //Tamaño de la ventana
    Dimension Tamano = size();

    //Crea el buffer de Imagen y lo asocia
    //al área de graficado no visible
    Imagen = createImage(Tamano.width, Tamano.height);
    NoVisible = Imagen.getGraphics();

    //Borra el fondo
    NoVisible.setColor(getBackground());
    NoVisible.fillRect(0, 0, Tamano.width, Tamano.height);
    NoVisible.setColor(getForeground());

    //Llama a paint pero con el objeto gráfico no visible
    paint(NoVisible);

    //Pinta la imagen al área visible
    objGrafico.drawImage(Imagen, 0, 0, this);
}

//Inicia el hilo
public void Iniciar()
{
    if (Ejecuta==false)
    {
        objHilo = new Thread(this);
        Ejecuta = true;
        objHilo.start();
    }
}

//Detiene el hilo
public void Detener()
{
    Ejecuta = false;
}
}

```

### GestionRecursos.java

```

/* Esta clase se encarga de los recursos de imagen y sonido
que tendrá el juego. Sus funciones son cargar el recurso
de disco (haciendo operaciones de archivo) y ponerlo en una
lista HashMap para que sea usado rápidamente */

import java.util.HashMap;
import java.applet.AudioClip;
import java.awt.image	BufferedImage;
import javax.imageio.ImageIO;
import java.net.*;
import java.applet.*;

public class GestionRecursos
{
    //Almacena los sonidos en una lista
    HashMap ListaSonidos = new HashMap();

    //Almacena las imágenes en una lista
    HashMap ListaImagenes = new HashMap();

    //Carga el sonido de disco
    public AudioClip CargaSonido(String Direccion)
    {
        try
        {
            URL RutaSonido = getClass().getResource(Direccion);
            AudioClip Sonido = Applet.newAudioClip(RutaSonido);
            return Sonido;
        }
    }
}

```

```

}
catch (Exception e)
{
    System.out.println("Problemas en cargar archivo");
    return null;
}

//Si el sonido no está en memoria (verificando la lista)
//lo carga de disco
public AudioClip RetornaSonido(String NombreSonido)
{
    //Busca el sonido en la lista
    AudioClip Sonido = (AudioClip) ListaSonidos.get(NombreSonido);

    //Si no encuentra el sonido en la lista, lo carga de disco
    if (Sonido == null)
    {
        Sonido = CargaSonido(NombreSonido);
        ListaSonidos.put(NombreSonido, Sonido);
    }
    return Sonido;
}

//Carga la imagen de disco
public BufferedImage CargaImagen(String Direccion)
{
    try
    {
        URL RutaImagen = getClass().getResource(Direccion);
        BufferedImage Imagen = ImageIO.read(RutaImagen);
        return Imagen;
    }
    catch (Exception e)
    {
        System.out.println("Problemas en cargar archivo");
        return null;
    }
}

//Si la imagen no está en memoria (verificando la lista)
//la carga de disco
public BufferedImage RetornaImagen(String NombreImagen)
{
    //Busca la imagen en la lista
    BufferedImage Imagen = (BufferedImage) ListaImagenes.get(NombreImagen);

    //Si no encuentra la imagen en la lista, la carga de disco
    if (Imagen == null)
    {
        Imagen = CargaImagen(NombreImagen);
        ListaImagenes.put(NombreImagen, Imagen);
    }
    return Imagen;
}
}

```

**Protagonista.java**

```

/* Esta clase maneja al protagonista, carga los
recursos de imagen y sonido, la lógica y de dibujarlo */

import java.awt.*;
import java.applet.AudioClip;
import java.awt.image.BufferedImage;

public class Protagonista extends Canvas
{
    //Posición del protagonista al inicio
    int PosX=180, PosY=10;

    //Desplazamiento del protagonista
    int IncX=1, IncY=1;

    //Sonido del rebote
    GestionRecursos Recurso = new GestionRecursos();
    AudioClip SonidoEjecuta;

    //Imagen del protagonista
    BufferedImage Imagen;

    //Carga la imagen y sonido del protagonista
    public Protagonista()
    {
        SonidoEjecuta = Recurso.RetornaSonido("rebote.wav");
        Imagen = Recurso.RetornaImagen("Imagen.png");
    }

    public void Pintar(Graphics objGrafico, int Ancho, int Alto)
    {
        //Mueve el protagonista
    }
}

```

```
PosX+=IncX;
PosY+=IncY;

//Chequea si rebota contra una pared.
if (PosX<=0 || PosX+Imagen.getWidth() >= Ancho)
{
    IncX*=-1;
    SonidoEjecuta.play();
}

if (PosY<=0 || PosY+Imagen.getHeight() >= Alto)
{
    IncY*=-1;
    SonidoEjecuta.play();
}

//Muestra la imagen del protagonista
objGrafico.drawImage(Imagen, PosX, PosY, this);

}
```

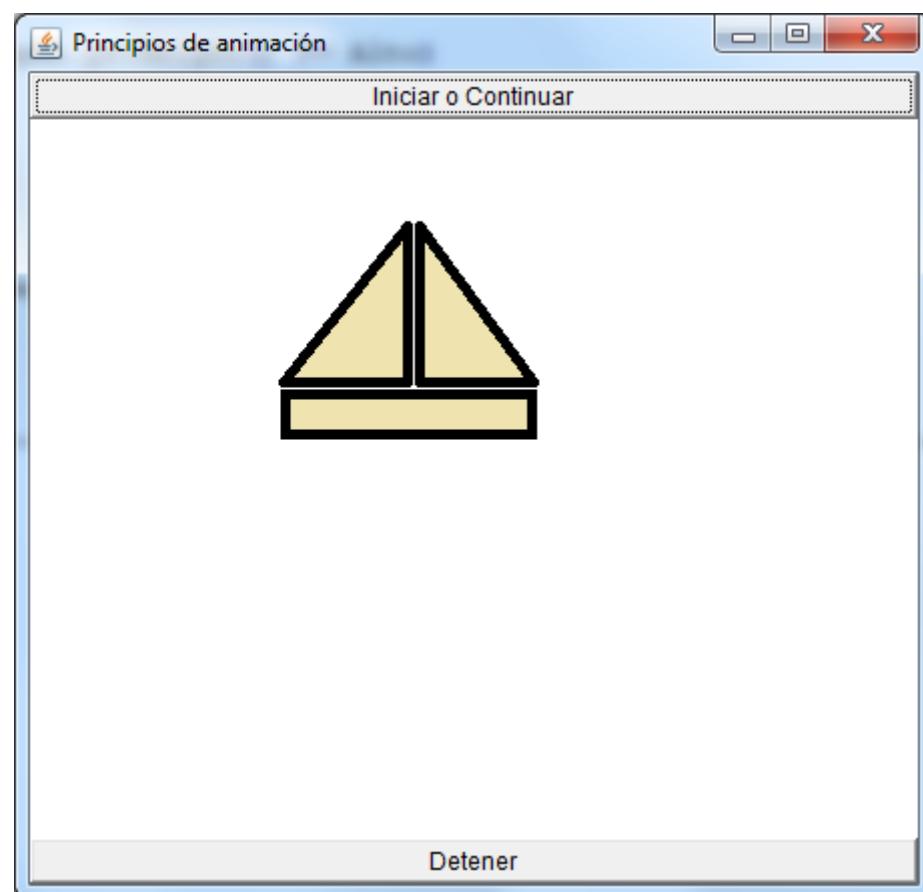


Figura 19: Animación y buen código

## 8. Juego de ejemplo

Ya tenemos todos los ingredientes para hacer un primer juego. Lo primero que debemos pensar es en que consiste el juego. Para este ejemplo, se trata de un cañón que dispara rayos para destruir chatarra espacial y en el que hay que ser cuidadoso para no destruir satélites activos.

El primer paso es tener las texturas o imágenes para cada protagonista del juego y estos son:

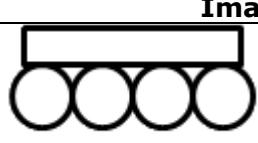
1. La base del cañón.
2. El cañón que debe girar para disparar a determinada posición.
3. Los rayos.
4. La chatarra espacial.
5. Satélites activos.

Otro recurso son los sonidos, que deben ser:

1. Cuando el rayo sale disparado del cañón.
2. Cuando el rayo impacta chatarra espacial o un satélite activo.

Hay que tener especial cuidado con los recursos de imágenes y sonidos, deben ser livianos porque serán mostrados en un videojuego, es decir, no deben ocupar mucho espacio en memoria, los sonidos deben ser cortos porque las explosiones serán continuas.

Las imágenes son las siguientes:

Imagen	Corresponde
	Base del cañón
	Cañón el cual gira
	Rayo disparado por el cañón
	Satélite activo
	Basura espacial

Lo que deseamos es que el juego luzca así:

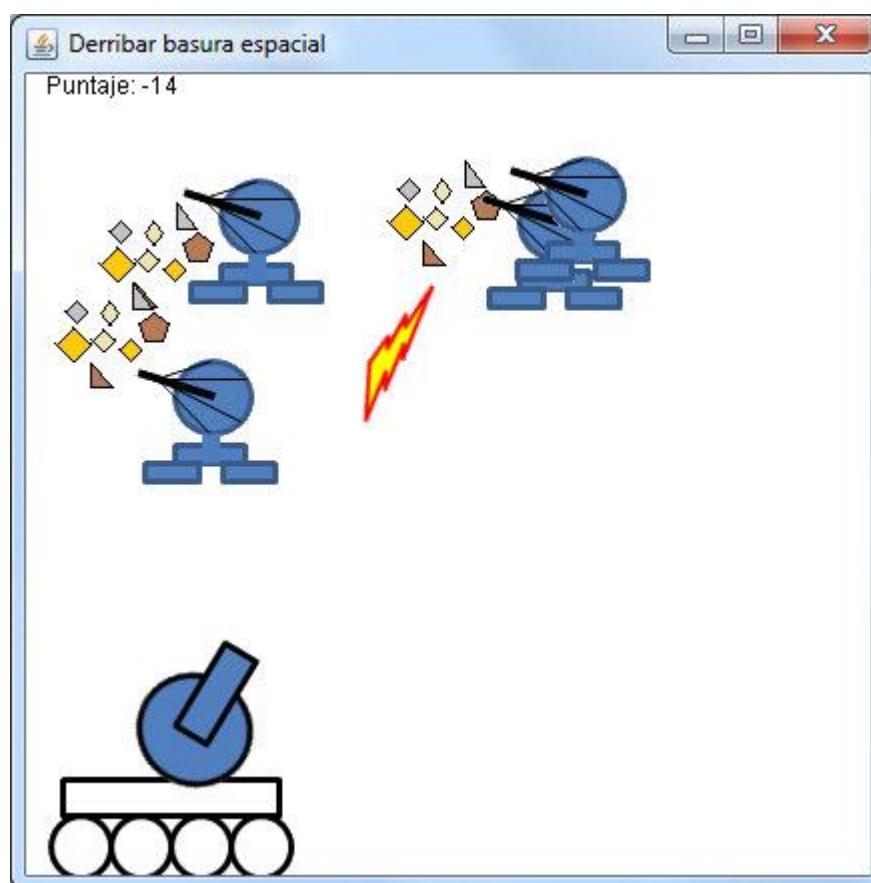


Figura 20: Imagen del juego

El comportamiento de los diferentes elementos del juego es:

Elemento	Comportamiento
Base del cañón	Estático, no se mueve.
Cañon	Gira sobre si mismo de 0 a 90 grados haciendo uso del ratón o el teclado, dispara con espaciadora el rayo.
Rayo	Sale disparado del cañón cuando el jugador presiona la barra espaciadora y se dibuja dependiendo del ángulo de salida que tenga el cañón. El rayo se desplaza a velocidad constante en la dirección de salida por toda la pantalla hasta que salga de esta o impacte contra chatarra o algún satélite activo. En el caso de impacto el rayo desaparece. Sólo puede

	haber un número determinado de rayos.
Satélite activo y chatarra	Se desplaza de izquierda a derecha horizontalmente a velocidad constante. Hay un número determinado de satélites y chatarra al tiempo mostrados en pantalla. Tanto la velocidad con que se desplaza y la altura en que aparece son generados al azar al inicio.
Puntaje	Aumenta en 1 cuando el rayo impacta en basura espacial y disminuye en 1 cuando el rayo impacta en satélites activos.

Necesitamos entonces una clase que inicie la aplicación, esa aplicación activa el modo gráfico y el hilo de animación.

#### InicioAnimacion.java

```
/* Inicia el videojuego */
import java.awt.*;

public class InicioAnimacion
{
    //La ventana
    static Frame Ventana = new Frame("Derribar basura espacial");

    //Objeto que muestra el gráfico animado
    static Animar objAnimar = new Animar();

    public static void main(String sbParametros[])
    {
        //Tamaño de la ventana
        Ventana.setSize(700, 450);

        //Adiciona el objeto gráfico a la ventana
        Ventana.add(objAnimar);

        //Muestra la ventana
        Ventana.setVisible(true);

        //Inicia el juego
        objAnimar.Iniciar();

        //El evento de cerrar ventana
        Ventana.addWindowListener(
            new java.awt.event.WindowAdapter()
            {public void windowClosing(java.awt.event.WindowEvent e)
            {System.exit(0);}});
    }
}
```

Viene luego la clase utilitaria que se encarga de cargar las imágenes y los sonidos.

#### GestionRecursos.java

```
/* Esta clase se encarga de los recursos de imagen y sonido
que tendrá el juego. Sus funciones son cargar el recurso
de disco (haciendo operaciones de archivo) y ponerlo en una
lista HashMap para que sea usado rápidamente */

import java.util.HashMap;
import java.applet.AudioClip;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.net.*;
import java.applet.*;

public class GestionRecursos
{
    //Almacena los sonidos en una lista
    HashMap ListaSonidos = new HashMap();

    //Almacena las imágenes en una lista
    HashMap ListaImagenes = new HashMap();

    //Carga el sonido de disco
    public AudioClip CargaSonido(String Direccion)
    {
        try
        {
            URL RutaSonido = getClass().getResource(Direccion);
            AudioClip Sonido = Applet.newAudioClip(RutaSonido);
            return Sonido;
        }
        catch (Exception e)
        {
            System.out.println("Problemas en cargar archivo");
            return null;
        }
    }

    //Si el sonido no está en memoria (verificando la lista)
    //lo carga de disco
    public AudioClip RetornaSonido(String NombreSonido)
```

```

{
    //Busca el sonido en la lista
    AudioClip Sonido = (AudioClip) ListaSonidos.get(NombreSonido);

    //Si no encuentra el sonido en la lista, lo carga de disco
    if (Sonido == null)
    {
        Sonido = CargaSonido(NombreSonido);
        ListaSonidos.put(NombreSonido, Sonido);
    }
    return Sonido;
}

//Carga la imagen de disco
public BufferedImage CargaImagen(String Direccion)
{
    try
    {
        URL RutaImagen = getClass().getResource(Direccion);
        BufferedImage Imagen = ImageIO.read(RutaImagen);
        return Imagen;
    }
    catch (Exception e)
    {
        System.out.println("Problemas en cargar archivo");
        return null;
    }
}

//Si la imagen no está en memoria (verificando la lista)
//la carga de disco
public BufferedImage RetornaImagen(String NombreImagen)
{
    //Busca la imagen en la lista
    BufferedImage Imagen = (BufferedImage) ListaImagenes.get(NombreImagen);

    //Si no encuentra la imagen en la lista, la carga de disco
    if (Imagen == null)
    {
        Imagen = CargaImagen(NombreImagen);
        ListaImagenes.put(NombreImagen, Imagen);
    }
    return Imagen;
}
}

```

Cada elemento del videojuego es controlado por su propia clase para dar así una mayor comprensión del juego. Empecemos con la clase que se encarga de mostrar la base del lanzador.

#### BaseLanzador.java

```

/* Esta clase maneja la base del lanzador */
import java.awt.*;
import java.awt.image.BufferedImage;

public class BaseLanzador extends Canvas
{
    //Posición de la base del lanzador
    int PosX=10, PosY=350;

    //Imagen del Lanzador
    GestionRecursos Recurso = new GestionRecursos();
    BufferedImage Imagen;

    //Carga la imagen del lanzador
    public BaseLanzador()
    {
        Imagen = Recurso.RetornaImagen("BaseLanzador.png");
    }

    public void Pintar(Graphics objGrafico)
    {
        //Muestra la imagen de la base del lanzador
        objGrafico.drawImage(Imagen, PosX, PosY, this);
    }
}

```

Observemos que esa clase requiere de la clase utilitaria GestionRecursos para cargar la imagen de la base. El mostrar esta imagen es hecho por el método Pintar.

Viene luego la clase que maneja el cañón. El cañón debe girar sobre si mismo dependiendo de cómo el jugador mueve el ratón. Luego la imagen del cañón debe girar dependiendo de un ángulo dado. Técnicamente estamos transformando la imagen y eso requiere de la clase `AffineTransform` y `AffineTransformOp` de Java.

#### Lanzador.java

```

/* Esta clase se encarga del lanzador */
import java.awt.*;
import java.awt.image.BufferedImage;
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;

```

```

import java.awt.image.AffineTransformOp;

public class Lanzador extends Canvas
{
    //Posición del lanzador al inicio
    int PosX=50, PosY=290;

    //Ángulo del lanzador en grados
    int Angulo = 0;

    //Imagen del lanzador
    GestionRecursos Recurso = new GestionRecursos();
    BufferedImage Imagen;

    //Retorna valores
    public int getPosX() { return PosX; }
    public int getPosY() { return PosY; }
    public int getAngulo() { return Angulo; }

    //Carga la imagen del lanzador
    public Lanzador()
    {
        Imagen = Recurso.RetornaImagen("Lanzador.png");
    }

    public void Pintar(Graphics objGrafico, int Angulo)
    {
        this.Angulo = Angulo;

        //Las siguientes líneas sirven para transformar la imagen (girarla)
        AffineTransform GiraImagen = new AffineTransform();
        GiraImagen.rotate(this.Angulo*Math.PI/180, Imagen.getWidth()/2, Imagen.getHeight()/2);
        AffineTransformOp Transformacion = new AffineTransformOp(GiraImagen, AffineTransformOp.TYPE_BILINEAR);

        //Se requiere Java 2D
        Graphics2D Graficos2D = (Graphics2D) objGrafico;
        Graficos2D.drawImage(Imagen, Transformacion, PosX, PosY);
    }
}

```

Los siguientes elementos del juego tienen estados: el estado activo si el objeto es visible e interactua con los otros elementos activos, y el estado inactivo en el que el objeto simplemente no se muestra.

En el caso del rayo, cuando se dispara este objeto debe girar la imagen dependiendo de cómo esté el cañón (usa de nuevo las clases `AffineTransform` y `AffineTransformOp`), la imagen se desplaza por el tablero a una velocidad constante. Si el rayo se sale de la pantalla, su estado pasa a inactivo.

#### Rayo.java

```

/* Manejo de los Rayos */
import java.awt.*;
import java.applet.AudioClip;
import java.awt.image.BufferedImage;
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;
import java.awt.image.AffineTransformOp;

public class Rayo extends Canvas
{
    //Estado del Rayo
    boolean Estado;

    //Posición del Rayo
    float PosX, PosY;

    //Avance del Rayo
    float Angulo, IncX, IncY;

    //Sonido del despegue
    GestionRecursos Recurso = new GestionRecursos();
    AudioClip SonidoEjecuta;

    //Imagen del Rayo
    BufferedImage Imagen;

    //Giro de la imagen
    AffineTransformOp Transformacion;

    //Retorna los valores de los atributos
    public boolean getEstado(){ return Estado; }
    public int getPosX(){ return (int) PosX; }
    public int getPosY(){ return (int) PosY; }
    public int getAncho(){ return Imagen.getWidth(); }
    public int getAlto(){ return Imagen.getHeight(); }

    //Da valores a los atributos
    public void setEstado(boolean Estado) { this.Estado = Estado; }

    //Carga la imagen y sonido del Rayo. Inicializa valores.
    public Rayo()
    {
        SonidoEjecuta = Recurso.RetornaSonido("Disparo.wav");
        Imagen = Recurso.RetornaImagen("Rayo.png");
    }
}

```

```

PosX = 0;
PosY = 0;
Angulo = 0;
IncX = 0;
IncY = 0;
Estado = false;
}

//Dispara el rayo, lo activa y pone los valores iniciales
public void Disparar(float PosX, float PosY, float Angulo, float Velocidad)
{
    Estado = true;
    this.PosX = PosX;
    this.PosY = PosY;
    this.Angulo = Angulo;
    this.IncX = (float) (Velocidad*Math.cos(Angulo*Math.PI/180));
    this.IncY = (float) (Velocidad*Math.sin(Angulo*Math.PI/180));

    //Las siguientes líneas sirven para transformar la imagen (girarla)
    AffineTransform GiraImagen = new AffineTransform();
    GiraImagen.rotate(this.Angulo*Math.PI/180, Imagen.getWidth()/2, Imagen.getHeight()/2);
    Transformacion = new AffineTransformOp(GiraImagen, AffineTransformOp.TYPE_BILINEAR);

    //Ejecuta sonido de disparo
    SonidoEjecuta.play();
}

//Actualiza la posición del rayo si está activo y luego lo dibuja
public void Pintar(Graphics objGrafico, int Ancho, int Alto)
{
    if (Estado)
    {
        //Mueve el rayo
        PosX+=IncX;
        PosY+=IncY;

        //Se desactiva si se sale de la pantalla
        if (PosX > Ancho || PosY > Alto || PosX < 0 || PosY < 0)
            Estado = false;

        //Se requiere Java 2D
        Graphics2D Graficos2D = (Graphics2D) objGrafico;
        Graficos2D.drawImage(Imagen, Transformacion, (int) PosX, (int) PosY);
    }
}
}
}

```

Viene la chatarra espacial y los satélites, estos objetos se mueven de izquierda a derecha, tanto la altura como la velocidad con que inician es aleatoria. Cómo la chatarra y los satélites tendrán el mismo comportamiento y sólo los diferencia la imagen a mostrar entonces solo se requiere de una clase que es la siguiente:

#### Blanco.java

```

/* Esta clase maneja la chatarra y los satélites */
import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.*;

public class Blanco extends Canvas
{
    //Activo o inactivo el blanco
    boolean Estado;

    //Posición del blanco
    int PosX, PosY;

    //Desplazamiento del blanco
    int Velocidad;

    //Recurso de imagen
    GestionRecursos Recurso = new GestionRecursos();

    //Imagen de la chatarra
    BufferedImage Imagen;

    //Retorna los valores de los atributos
    public boolean getEstado(){ return Estado; }
    public int getPosX(){ return PosX; }
    public int getPosY(){ return PosY; }
    public int getAncho(){ return Imagen.getWidth(); }
    public int getAlto(){ return Imagen.getHeight(); }

    //Da valores a los atributos
    public void setEstado(boolean Estado) { this.Estado = Estado; }

    //Carga la imagen del blanco
    public Blanco(int TipoBlanco)
    {
        if (TipoBlanco==1) Imagen = Recurso.RetornaImagen("BasuraEspacial.png");
        if (TipoBlanco==2) Imagen = Recurso.RetornaImagen("Satelite.png");
        PosX = 0;
    }
}

```

```

    PosY = 0;
    Velocidad = 0;
    Estado = false;
}

public void Pintar(Graphics objGrafico, Random Azar, int Ancho, int Alto, int AlturaMax, int AlturaMin, int VelocidadMax, int
VelocidadMin)
{
    if (Estado==true)
    {
        //Mueve la chatarra
        PosX+=Velocidad;

        //Chequea si se sale de la ventana
        if (PosX<0 || PosX+Imagen.getWidth() > Ancho)
            Estado = false;

        //Muestra la imagen del blanco
        objGrafico.drawImage(Imagen, PosX, PosY, this);
    }
    else
    { //Activa la chatarra/satélite con altura y velocidad al azar
        Estado = true;
        this.PosX = 0;
        PosY = (int) (Azar.nextFloat()*AlturaMax-AlturaMin)+AlturaMin;
        Velocidad = (int) (Azar.nextFloat()*(VelocidadMax-VelocidadMin)+VelocidadMin);
    }
}
}
}

```

Finalmente viene la clase que controla todo el juego, sus funcionalidades son:

1. Gestionar el hilo de la animación.
2. Instancia la base y el cañón.
3. Instancia los rayos, la chatarra y los satélites como arreglos unidimensionales de objetos porque son varios los que aparecen en pantalla.
4. Detecta cuando el jugador mueve el ratón para girar el cañón.
5. Detecta cuando el jugador presiona la barra espaciadora para disparar los rayos.
6. Se encarga de chequear cuando colisiona el rayo con la basura y con los satélites. Si hay colisión desaparece el rayo y la basura, o el rayo y el satélite. Reproduce sonido de la colisión.
7. Muestra el puntaje. Aumenta puntaje si el rayo colisiona con basura espacial y disminuye si el rayo colisiona con satélites.
8. Se encarga de activar el double buffering.
9. Llama el método de pintar de cada objeto en pantalla.

#### Animar.java

```

/* Esa clase se encarga del hilo de ejecución del videojuego */
import java.awt.*;
import java.util.*;
import java.awt.event.*;
import java.applet.AudioClip;

//Para usar hilos debe heredar de Runnable
public class Animar extends Canvas implements Runnable, MouseMotionListener, MouseListener, KeyListener
{
    //Hilo que controla la animación
    Thread objHilo;

    //Objeto que inicia los números aleatorios
    Random Azar = new Random();

    //Controla el hilo
    boolean Ejecuta = false;

    //Constantes del juego
    boolean INACTIVO = false;

    //Instancia la base y lanzador
    BaseLanzador objBaseLanzador = new BaseLanzador();
    Lanzador objLanzador = new Lanzador();
    int Angulo = 270;

    //Instancia los rayos como lista
    int TOTALRAYOS = 10;
    Rayo ListaRayo[] = new Rayo[TOTALRAYOS];

    //Instancia la chatarra como lista
    int TOTALCHATARRA = 3;
    Blanco ListaChatarra[] = new Blanco[TOTALCHATARRA];

    //Instancia los satélites como lista
    int TOTALSATELITE = 4;
    Blanco ListaSatelite[] = new Blanco[TOTALSATELITE];

    //Puntaje del juego
    int Puntaje = 0;

    //Valor X del mouse anterior
    double MousePosXAnterior = 0;

    //Recursos de imagen y sonido
    GestionRecursos Recurso = new GestionRecursos();
    AudioClip SonidoEjecuta;
}

```

```

//Activa la captura de eventos
public Animar()
{
    addMouseMotionListener(this);
    addMouseListener(this);
    addKeyListener(this);

    //Inicia la lista de rayos, chatarra, satélites
    for (int Cont=0; Cont<TOTALRAYOS; Cont++)
        ListaRayo[Cont]= new Rayo();

    for (int Cont=0; Cont<TOTALCHATARRA; Cont++)
        ListaChatarra[Cont]= new Blanco(1);

    for (int Cont=0; Cont<TOTALSATELITE; Cont++)
        ListaSatelite[Cont]= new Blanco(2);

    //Carga el sonido de la explosión de la colisión
    SonidoEjecuta = Recurso.RetornaSonido("Explosion.wav");
}

//De MouseMotionListener
public void mouseMoved(MouseEvent objMouse)
{
    //Dependiendo del movimiento del ratón mueve el cañón
    Angulo+=objMouse.getPoint().getX()-MousePosXAnterior;
    if (Angulo > 360) Angulo = 360;
    if (Angulo < 270) Angulo = 270;
    MousePosXAnterior = objMouse.getPoint().getX();
}

public void mouseDragged(MouseEvent objMouse){}

//De MouseListener
public void mousePressed (MouseEvent objMouse){}
public void mouseClicked (MouseEvent objMouse){}
public void mouseEntered (MouseEvent objMouse){}
public void mouseExited (MouseEvent objMouse){}
public void mouseReleased (MouseEvent objMouse){}

//De KeyListener
public void keyTyped(KeyEvent e){}
public void keyReleased(KeyEvent e){}

public void keyPressed(KeyEvent e)
{
    //Gira el cañón
    if (e.getKeyCode() == KeyEvent.VK_RIGHT) Angulo+=3;
    if (e.getKeyCode() == KeyEvent.VK_LEFT) Angulo-=3;

    //Limita cuanto se puede girar el cañón
    if (Angulo > 360) Angulo = 360;
    if (Angulo < 270) Angulo = 270;

    //Si se dispara un rayo
    if (e.getKeyCode() == KeyEvent.VK_SPACE)
        DispararRayo();
}

public void DispararRayo()
{
    //Busca algún rayo inactivo
    for(int Cont=0; Cont<TOTALRAYOS; Cont++)
        if (ListaRayo[Cont].getEstado() == INACTIVO)
        {
            ListaRayo[Cont].Disparar((float) objLanzador.getPosX(), (float) objLanzador.getPosY(), (float) objLanzador.getAngulo(),
3.0f);
            break;
        }
}

//Método ejecutado por el hilo
public void run()
{
    while(Ejecuta)
    {
        //Chequea si algún rayo colisiona con alguna chatarra o satélite
        Puntaje += ChequeaColisiones();

        repaint();

        try{Thread.sleep(10); }catch (InterruptedException E){}
    }
}

//Chequea si algún rayo colisiona con chatarra o satélite
public int ChequeaColisiones()
{
    //Va de rayo en rayo
    for (int Cont=0; Cont<TOTALRAYOS; Cont++)
    {
        //Si el rayo está inactivo pasa al siguiente
        if (ListaRayo[Cont].getEstado() == INACTIVO) continue;
}

```

```

//Deduce un rectángulo para ese rayo
Rectangle Rect1 = new Rectangle(ListaRayo[Cont].getPosX(), ListaRayo[Cont].getPosY(), ListaRayo[Cont].getAncho(),
ListaRayo[Cont].getAlto());

//Va de chatarra en chatarra
for (int Chatarra=0; Chatarra<TOTALCHATARRA; Chatarra++)
{
    //Si la chatarra está inactiva pasa a la siguiente
    if (ListaChatarra[Chatarra].getEstado() == INACTIVO) continue;

    //Deduce un rectángulo para esa chatarra
    Rectangle Rect2 = new Rectangle(ListaChatarra[Chatarra].getPosX(), ListaChatarra[Chatarra].getPosY(),
ListaChatarra[Chatarra].getAncho(), ListaChatarra[Chatarra].getAlto());

    //Si hay intersección entre esos dos rectángulos, entonces hay colisión
    if (Rect1.intersects(Rect2))
    {
        //Inactiva el rayo
        ListaRayo[Cont].setEstado(INACTIVO);

        //Inactiva la chatarra
        ListaChatarra[Chatarra].setEstado(INACTIVO);

        //Ejecuta sonido de explosión
        SonidoEjecuta.play();

        //Retorna 1 para el puntaje
        return 1;
    }
}

//Va de satélite en satélite
for (int Satelite=0; Satelite<TOTALSATELITE; Satelite++)
{
    //Si el satélite está inactivo pasa al siguiente
    if (ListaSatelite[Satelite].getEstado() == INACTIVO) continue;

    //Deduce un rectángulo para ese satélite
    Rectangle Rect2 = new Rectangle(ListaSatelite[Satelite].getPosX(), ListaSatelite[Satelite].getPosY(),
ListaSatelite[Satelite].getAncho(), ListaSatelite[Satelite].getAlto());

    //Si hay intersección entre esos dos rectángulos, entonces hay colisión
    if (Rect1.intersects(Rect2))
    {
        //Inactiva el rayo
        ListaRayo[Cont].setEstado(INACTIVO);

        //Inactiva el satélite
        ListaSatelite[Satelite].setEstado(INACTIVO);

        //Ejecuta sonido de explosión
        SonidoEjecuta.play();

        //Retorna -1 para el puntaje
        return -1;
    }
}

//No hubo colisión, luego no hay puntaje
return 0;
}

//Actualiza y pinta los protagonistas del videojuego
public void paint (Graphics objGrafico)
{
    objBaseLanzador.Pintar(objGrafico);
    objLanzador.Pintar(objGrafico, Angulo);

    //Actualiza los rayos, chatarra, satélites y los dibuja
    for (int Cont=0; Cont<TOTALRAYOS; Cont++)
        ListaRayo[Cont].Pintar(objGrafico, getSize().width, getSize().height);

    for (int Cont=0; Cont<TOTALCHATARRA; Cont++)
        ListaChatarra[Cont].Pintar(objGrafico, Azar, getSize().width, getSize().height, (int) (getSize().height*0.01), (int)
(getSize().height*0.5), 2, 5);

    for (int Cont=0; Cont<TOTALSATELITE; Cont++)
        ListaSatelite[Cont].Pintar(objGrafico, Azar, getSize().width, getSize().height, (int) (getSize().height*0.1), (int)
(getSize().height*0.5), 1, 4);

    //Muestra el puntaje
    objGrafico.drawString("Puntaje: " + Puntaje, 10, 10);
}

//Se implementa aquí el double buffering
public void update(Graphics objGrafico)
{
    //Area de graficado no visible
    Graphics NoVisible;

    //Crea el buffer de Imagen y lo asocia
}

```

```
//al área de graficado no visible
Image Imagen = createImage(getSize().width, getSize().height);
NoVisible = Imagen.getGraphics();

//Borra el fondo
NoVisible.setColor(getBackground());
NoVisible.fillRect(0, 0, getSize().width, getSize().height);
NoVisible.setColor(getForeground());

//Llama a paint pero con el objeto gráfico no visible
paint(NoVisible);

//Pinta la imagen al área visible
objGrafico.drawImage(Imagen, 0, 0, this);
}

//Inicia el hilo
public void Iniciar()
{
if (Ejecuta==false)
{
    objHilo = new Thread(this);
    Ejecuta = true;
    objHilo.start();
}
}

//Detiene el hilo
public void Detener()
{
    Ejecuta = false;
}
}
```

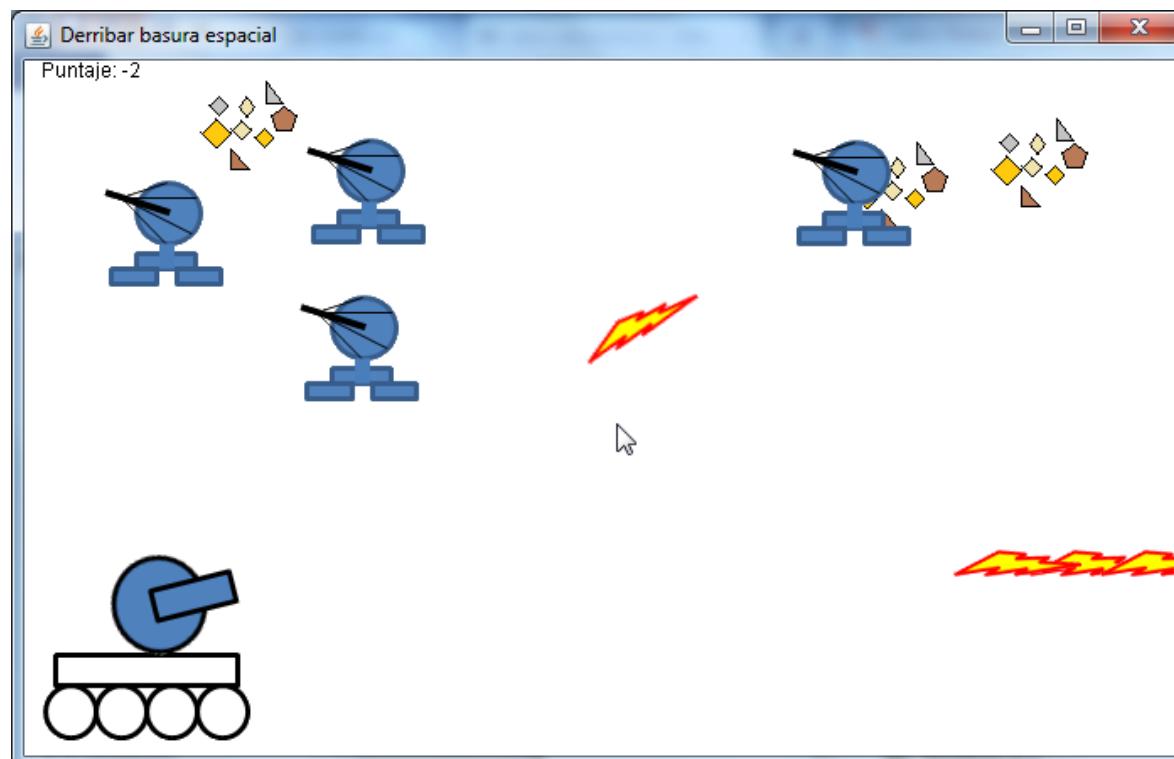


Figura 21: Ejemplo de ejecución del videojuego

## 9. El juego como Applet

Hasta ahora el juego solo puede ser ejecutado como una aplicación de escritorio. Si queremos que el juego pueda ejecutarse desde un navegador, entonces debemos programarlo como un applet. Afortunadamente no es necesario reescribir todo el juego, solo necesitamos variar la clase que arranca el videojuego: InicioAnimacion, el resto de clases permanece sin cambio.

El cambio que vamos a hacer a InicioAnimacion es para que pueda ejecutarse el juego de dos formas: como aplicación de escritorio y como applet. El código es el siguiente:

InicioAnimacion.java

```
/* Inicia el videojuego */
import java.awt.*;
import java.awt.event.WindowListener;
import java.awt.event.WindowEvent;
import java.applet.*;

public class InicioAnimacion extends Applet
{
    //La ventana
    Frame Ventana = new Frame("Derribar basura espacial");

    //Objeto que muestra el gráfico animado
    Animar objAnimar = new Animar();

    //Botón para activar el applet
    Button btnBoton;

    //Ejecuta la aplicación
    public static void main(String sbParametros[])
    {
        InicioAnimacion objJuego = new InicioAnimacion();
        objJuego.Inicia();
    }

    //Ejecuta como Applet
    public void init()
    {
        btnBoton = new Button("Iniciar el videojuego");
        add(btnBoton);
    }

    public boolean action (Event objEvento, Object objObjeto)
    {
        if (objEvento.target == btnBoton)
            Inicia();
        return true;
    }

    public void Inicia()
    {
        //Tamaño de la ventana
        Ventana.setSize(700, 450);

        //Adiciona el objeto gráfico a la ventana
        Ventana.add(objAnimar);

        //Adiciona un WindowsListener
        Ventana.addWindowListener(new WindowListener() {
            public void windowActivated(WindowEvent e){}
            public void windowClosed(WindowEvent e){}
            public void windowDeactivated(WindowEvent e){}
            public void windowDeiconified(WindowEvent e){}
            public void windowIconified(WindowEvent e){}
            public void windowOpened(WindowEvent e){}
            public void windowClosing(WindowEvent e)
            {
                System.out.println("Cerrando ventana...");
                objAnimar.Ejecuta = false;
                e.getWindow().dispose();
            }
        });

        //Muestra la ventana
        Ventana.setVisible(true);

        //Inicia el juego
        objAnimar.Iniciar();
    }
}
```

Los cambios con respecto a la clase original fueron:

1. Esta clase hereda de la clase Applet.
2. En main() la clase se instancia a si misma.
3. Se agrega el método init() que es el que primero ejecuta los navegadores.
4. En el método Inicia() se agrega un WindowsListener y allí se implementa el cierre de la aplicación que funciona tanto para aplicación de escritorio como applet.
5. Se agregar el método windowClosing que lo primero que debe hacer es romper el ciclo que ejecuta el hilo (con la instrucción objAnimar.Ejecuta = false). Luego se llama a dispose(), para liberar los recursos GUI.

Se requiere además generar un archivo HTML que invoque el applet. Un sencillo ejemplo de cómo debe ser ese archivo HTML es:

Rafael Alberto Moreno Parra. <http://darwin.50webs.com>

## Videojuego.html

```

<html>
<head>
<title>Videojuego (Applet Java)</title>
</head>
<body>
<applet code="InicioAnimacion.class" width="261" height="37"></applet>
</body>
</html>

```

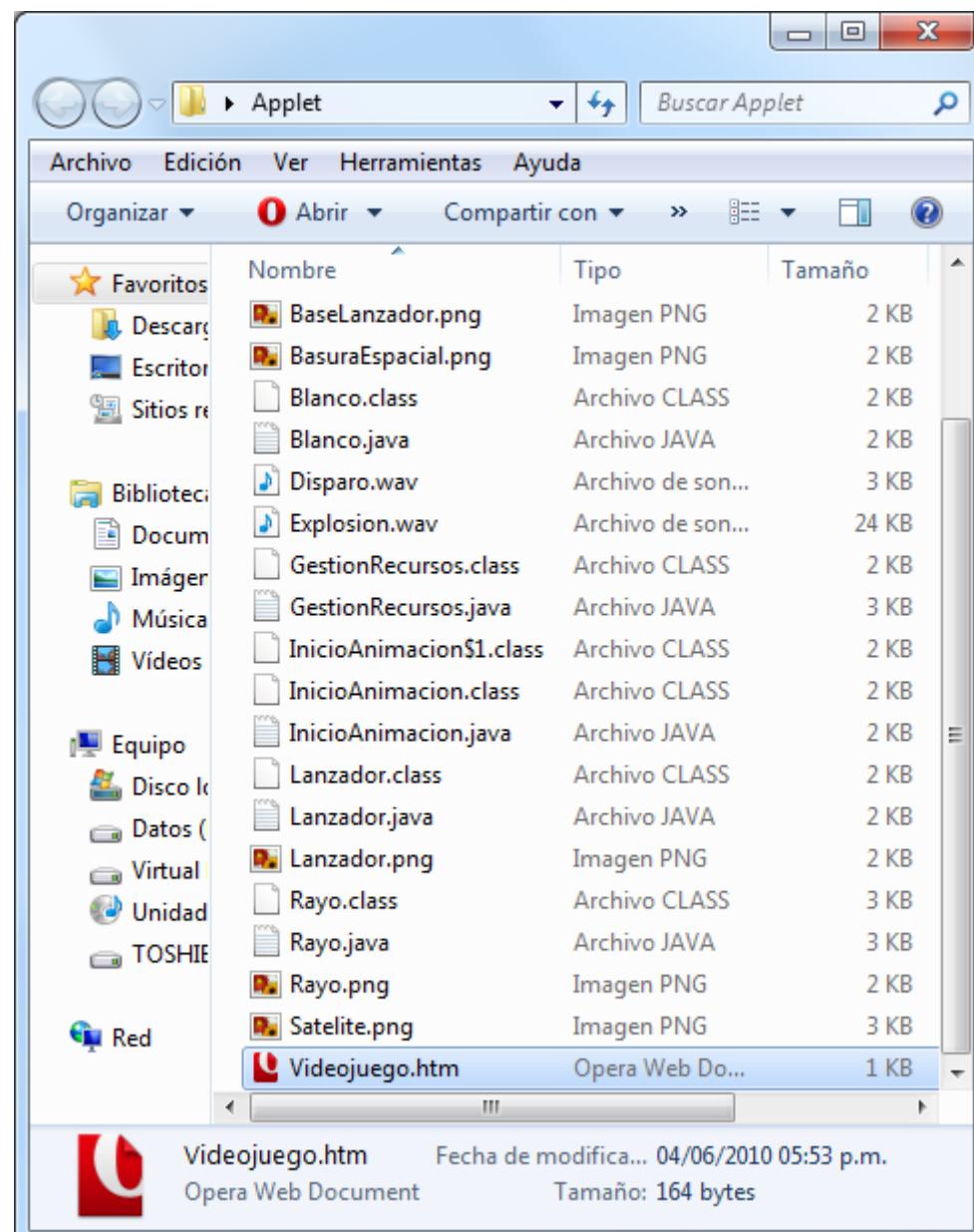


Figura 22: Juego tanto para escritorio y applet

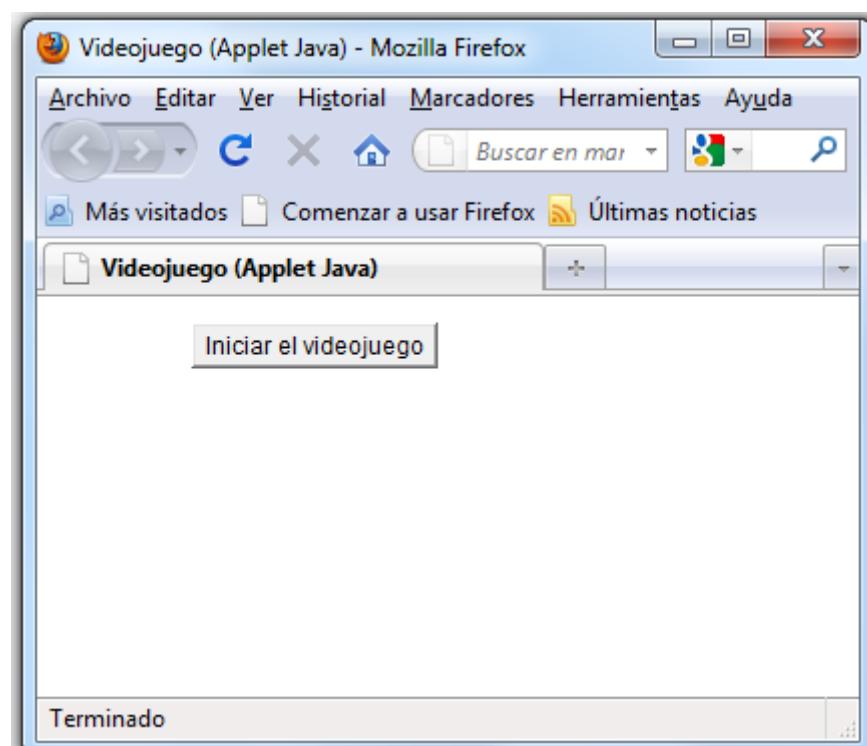


Figura 23: Applet inicia en Mozilla Firefox

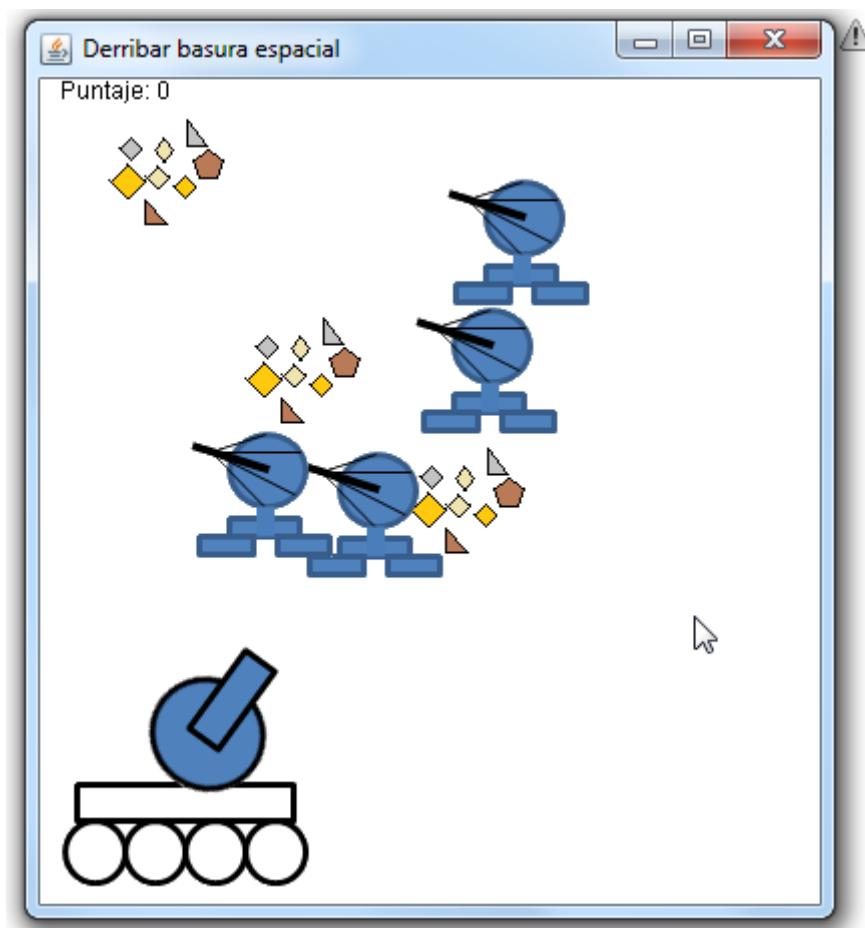


Figura 24: Juego ejecutando como Applet

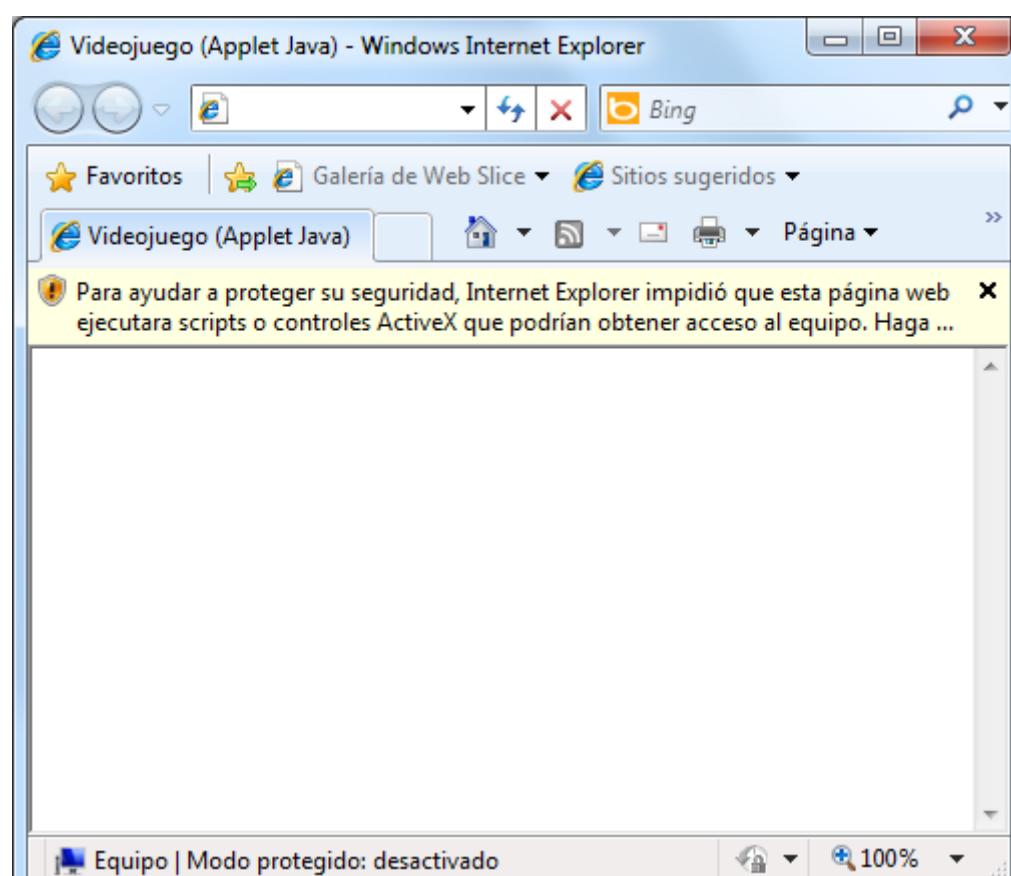
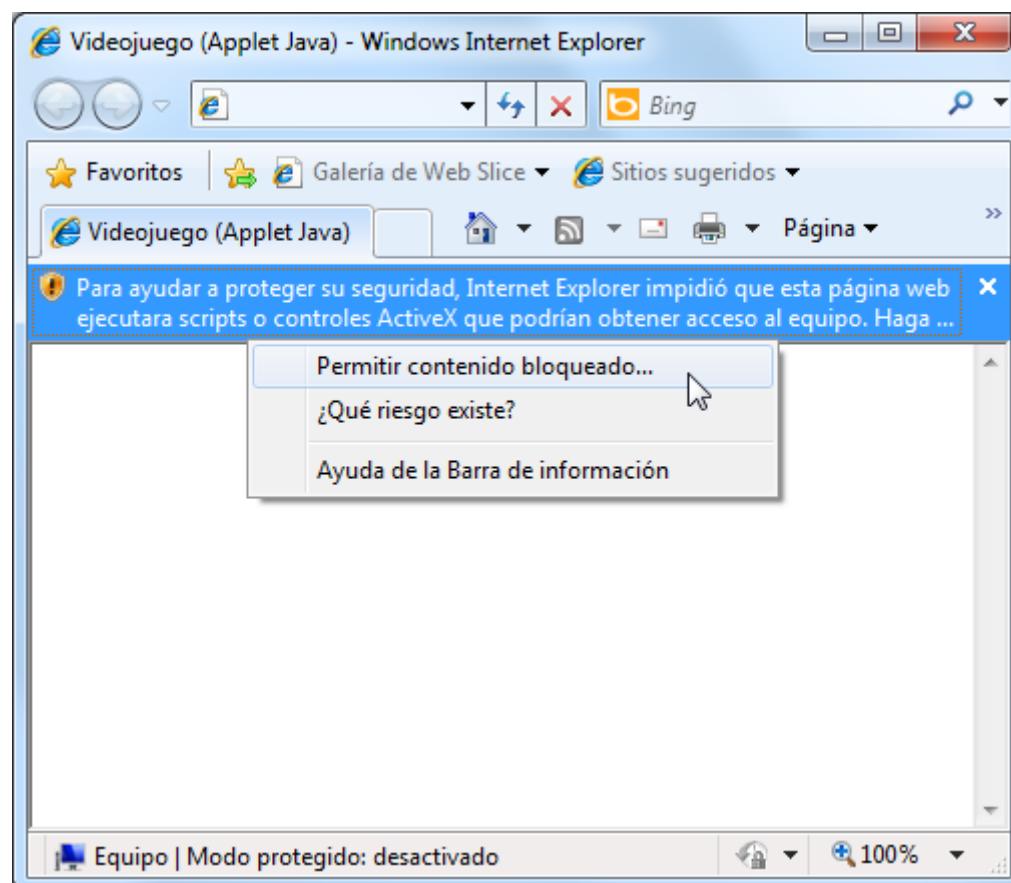
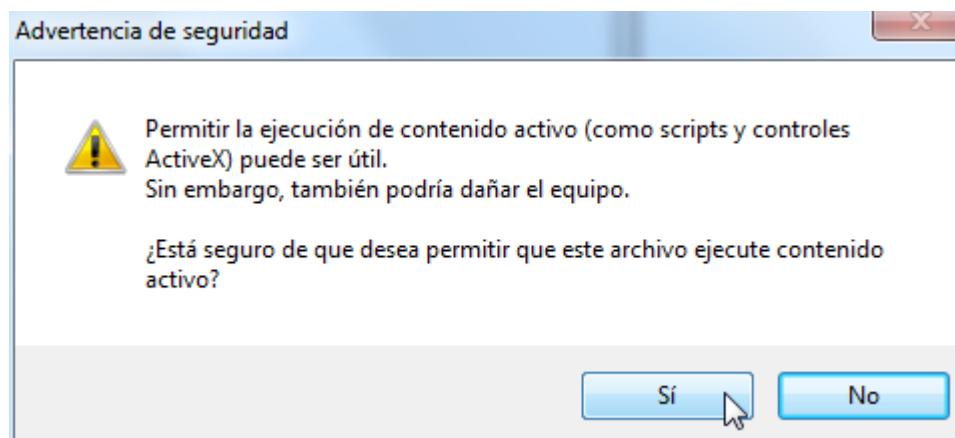


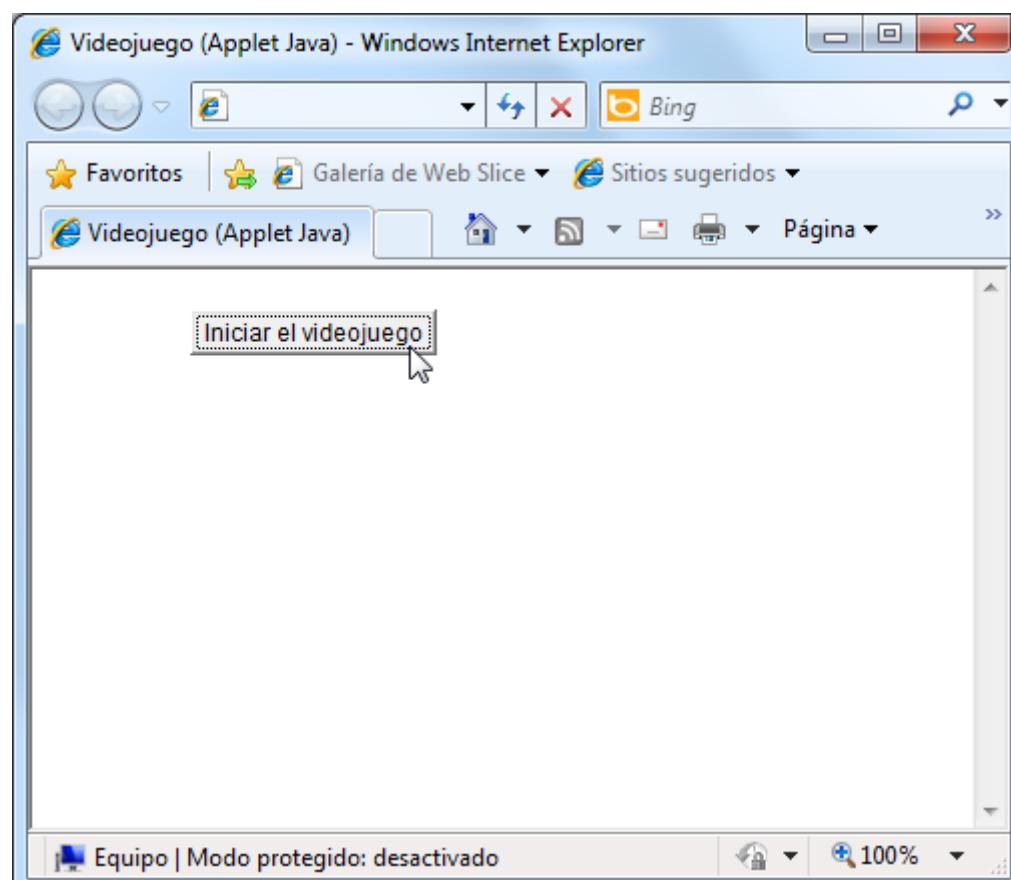
Figura 25: Internet Explorer requiere aprobación del usuario para ejecutar applets



**Figura 26: Dando permiso para ejecutar el videojuego**



**Figura 27: Reconfirmando que se ejecuta un applet Java**



**Figura 28: Inicia el applet en Internet Explorer**

Limitaciones de los applets: Un applet no puede acceder al disco duro del PC del usuario, luego guardar puntuajes o preferencias requerirá hacer operaciones en el servidor y no en el cliente. Otra limitante es que no controlamos qué versión de JRE (Java Runtime Environment), ni qué navegador tiene el usuario, luego se recomienda usar funciones o métodos antiguos en Java y no preocuparnos mucho por las advertencias del compilador que determinada API es "deprecated". El resultado es que a pesar de las advertencias del compilador, nuestro juego tiene mayor probabilidad de ser ejecutado en más equipos.

## 9.1 Applets y servidores remotos

El interés de que un juego en Java ejecute en un navegador es que cualquier usuario en internet pueda jugarlo sin contratiempos, sin importar el navegador o el sistema operativo del usuario.

Como hemos visto, el juego que tenemos hasta ahora requiere de varias clases, imágenes y sonidos. Todo eso debe ser transmitido del servidor al usuario por la red. El ejemplo anterior ejecutaba el applet pero con las clases y recursos (sonidos, imágenes) en el mismo disco duro. ¿Cómo hacemos entonces para que nuestro juego se transmita por completo desde un servidor remoto? Debemos entonces hacer dos pasos:

1. Generar un .jar con todas las clases y recursos.
2. Modificar el HTML para que haga referencia a ese .jar

¿Cómo generar un .jar?

En primer lugar debemos generar un archivo llamado MANIFEST.MF (en mayúsculas) que debe tener las siguientes líneas (debe dejar un salto de línea al final).

```
Manifest-Version: 1.0
Main-Class: InicioAnimacion
```

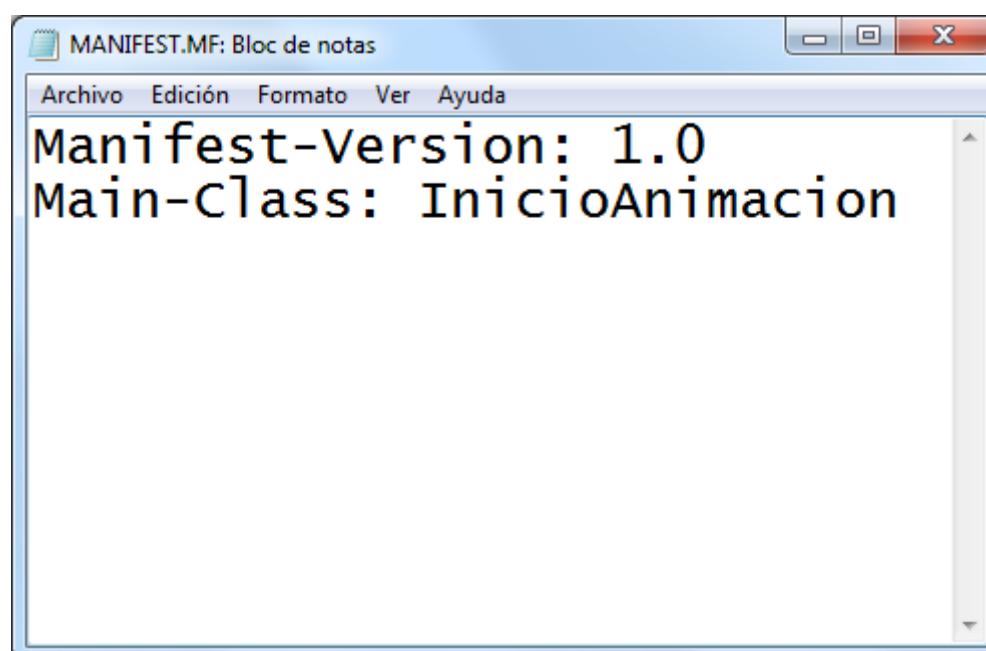


Figura 29: Archivo MANIFEST.MF

La **Main-Class** es la clase por donde inicia la aplicación.

Ese archivo debe ir en el directorio donde están las clases y archivos de recursos, luego se debe digitar en consola esta instrucción:

```
jar cmf MANIFEST.MF Juego.jar *.class *.png *.wav
```

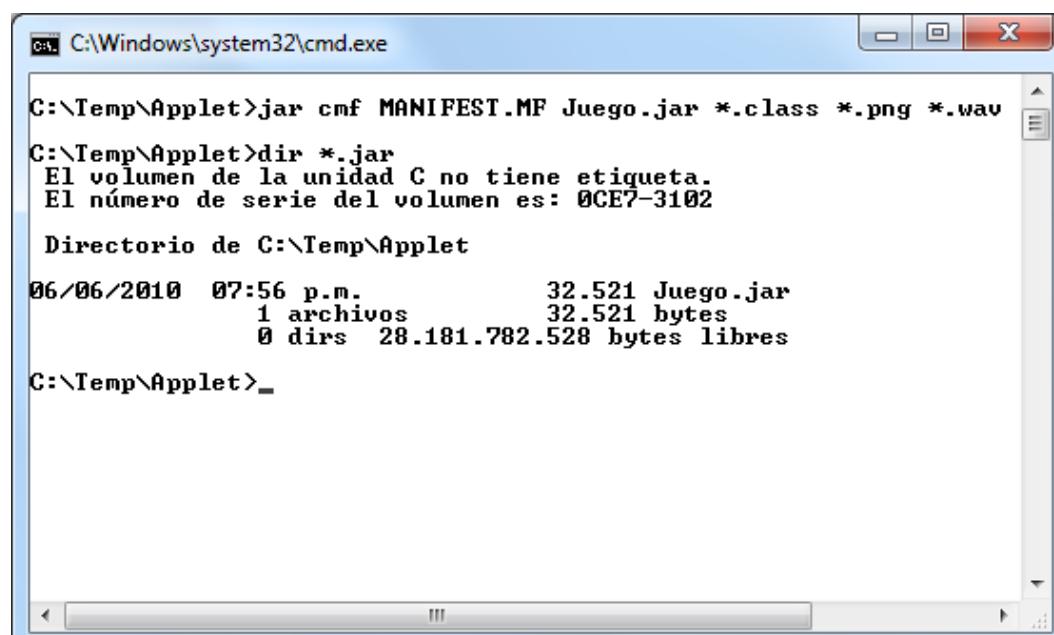


Figura 30: Generando el .jar

El .jar queda como un archivo independiente, que al dar doble clic en este, se ejecuta el juego.

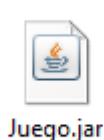


Figura 31: El .jar es un archivo ejecutable

Luego se debe generar un archivo HTML que llame al juego y que envíe el .jar al usuario. Este sería el código:

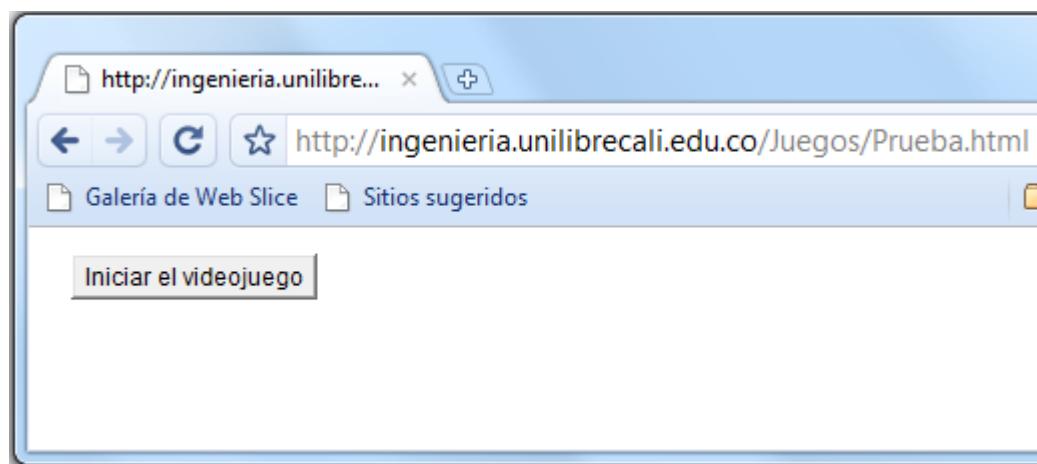
```
<html>
<body>
<applet code="InicioAnimacion.class" archive="Juego.jar">
</body>
</html>
```

Ambos archivos (Juego.jar y Prueba.html) estarían en el servidor web



**Figura 32: Archivos en el servidor Web**

Desde cualquier navegador se llama a la URL de la aplicación apuntando al archivo HTML.



**Figura 33: Se llama al videojuego**

Una de las ventajas de un juego escrito en Java y tenerlo como applet, es que no importa la plataforma que el jugador esté usando o el navegador, podrá ejecutar el juego, siempre y cuando tenga instalado el JRE (Java Runtime Environment).

## 10. Otra forma de tener gráficos y animación en Java: El componente JPanel

Swing es un conjunto de clases diseñadas para gráficos (incluye los controles de interfaz de usuario como botones, cajas de texto, etiquetas, etc.), 100% puro Java y más sofisticado que su predecesor AWT. JPanel es un contenedor liviano o ligero para gráficos que hace parte de Swing. Una de las ventajas del uso de JPanel es que ya no es necesario preocuparse por implementar el double buffering porque este ya está activado por defecto.

Para poder hacer uso de JPanel debemos primero hacer la ventana con JFrame.

En este primer ejemplo, se muestra como activar una ventana con el componente JFrame.

**Inicio.java**

```
//Necesario para usar Swing
import javax.swing.JFrame;

public class Inicio
{
    public static void main(String[] args)
    {
        //Usamos Swing para mostrar la ventana
        JFrame objFrame = new JFrame("Muestra ventana");

        //Ubicación y tamaño de la ventana
        objFrame.setBounds(0, 0, 300, 300);

        //Cierra la aplicación al cerrar la ventana
        objFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Muestra la ventana
        objFrame.setVisible(true);
    }
}
```

El resultado es una ventana la cual reacciona al evento de cerrar.

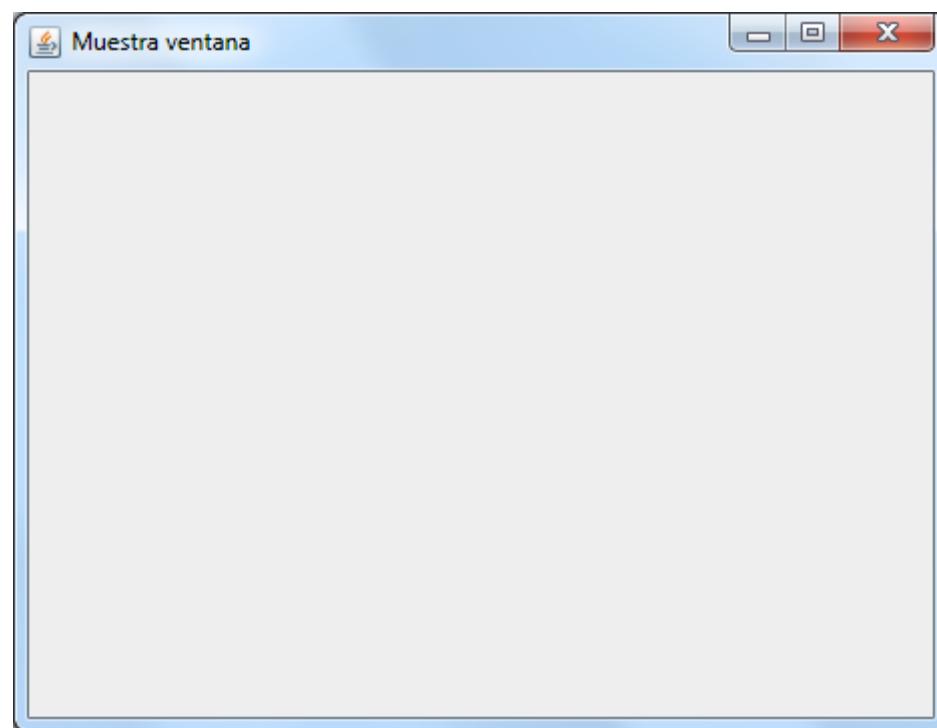


Figura 34: Una ventana con Swing

Es tiempo de dibujar algo, para eso usamos nuevamente dos clases: Inicio y Dibujar. La clase Dibujar hereda de JPanel.

Un JPanel es un contenedor. Dentro de ese contenedor se pueden poner otros componentes Swing/JFC como botones, cajas de texto, cajas de chequeo (checkbox), etc. Podemos poner cualquier número de JPanel, inclusive unos dentro de otros pero el JPanel requiere de un objeto ventana o applet o JDialog o JInternalFrame para ser visible. Y es allí donde reside su fortaleza, solo programamos para el JPanel y luego este lo mandamos a un applet o a una aplicación por lo que es muy reutilizable.

#### Inicio.java

```
import javax.swing.JFrame;

public class Inicio
{
    public static void main(String[] args)
    {
        //Usamos Swing para mostrar la ventana
        JFrame objFrame = new JFrame("Inicio Gráficos");

        //El componente que grafica
        Dibujar objDibuja = new Dibujar();
        objFrame.add(objDibuja);

        //Ubicación y tamaño de la ventana
        objFrame.setBounds(0, 0, 300, 300);

        //Cierra la aplicación al cerrar la ventana
        objFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

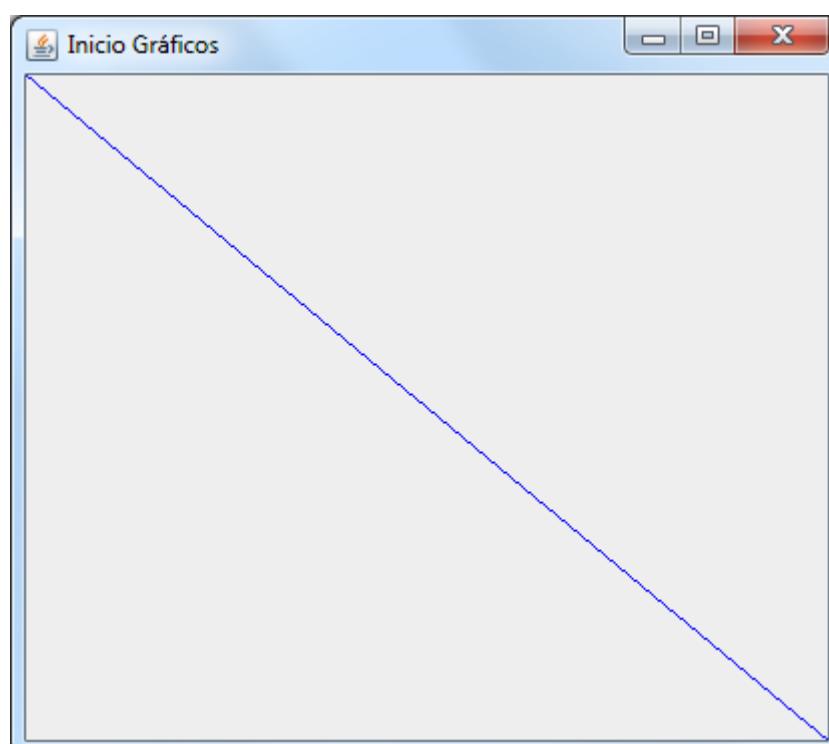
        //Muestra la ventana
        objFrame.setVisible(true);
    }
}
```

#### Dibujar.java

```
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

//Clase que dibuja, hereda de JPanel
class Dibujar extends JPanel
{
    //El método que dibuja
    public void paintComponent(Graphics objGrafico)
    {
        //Dibuja una línea
        objGrafico.setColor(Color.blue);
        objGrafico.drawLine(0, 0, getSize().width, getSize().height);
    }
}
```

Este es el resultado



**Figura 35: Uso de JPanel para mostrar gráficos**

## 11. Uso del componente JPanel en una aplicación de escritorio y en applets

Para que nuestro juego corra como aplicación y como applet, la clase inicial debe heredar de JApplet.

### Inicio.java

```
import javax.swing.JApplet;
import javax.swing.JFrame;

//Applet y aplicación
public class Inicio extends JApplet
{
    public void init()
    {
        //Instancia el JPanel
        Dibujar objDibuja = new Dibujar();

        //Muestra el JPanel dentro del applet
        this.getContentPane().add(objDibuja);
    }

    public static void main (String sbParametros[])
    {
        //Instancia el JPanel
        Dibujar objDibuja = new Dibujar();

        //Muestra el JPanel dentro de un JFrame
        JFrame objVentana = new JFrame();
        objVentana.getContentPane().add(objDibuja);

        //Ubicación y tamaño del JFrame
        objVentana.setBounds(0, 0, 500, 500);

        //Muestra el JFrame
        objVentana.setVisible(true);

        //Cierra la aplicación al cerrar el JFrame
        objVentana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

### Dibujar.java

```
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

// Dibuja en un JPanel
public class Dibujar extends JPanel
{
    public void paintComponent(Graphics objGrafico)
    {
        objGrafico.setColor (Color.BLACK);
        objGrafico.drawLine(0, 0, getSize().width, getSize().height);
    }
}
```

Como aplicación este ejemplo ejecuta así:

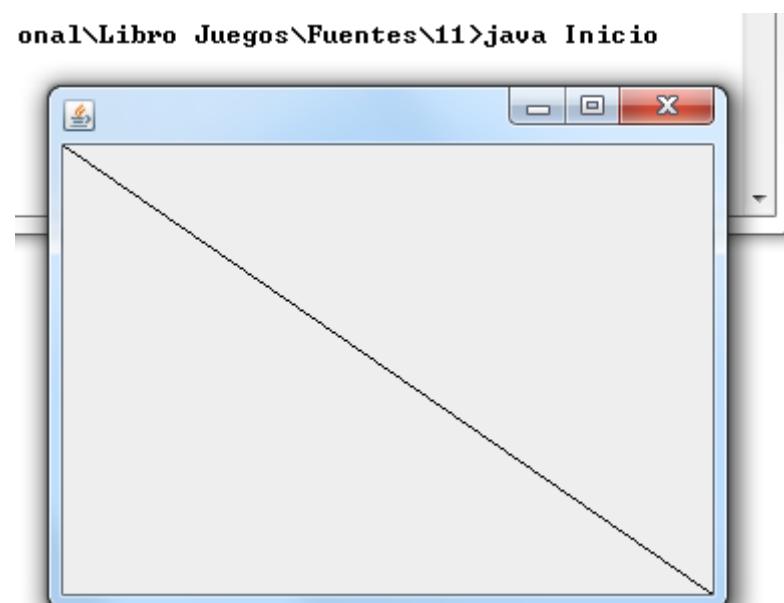


Figura 36: Ejecutando como aplicación

Para correr el applet, es necesario un archivo HTML

Prueba.html

```
<html>
<head>
<title>Prueba de JApplet</title>
</head>
<body>
<applet code="Inicio.class" width=100 height=100></applet>
</body>
</html>
```

Se da doble clic al archivo HTML para dar inicio a un navegador, el applet ejecuta de esta forma:

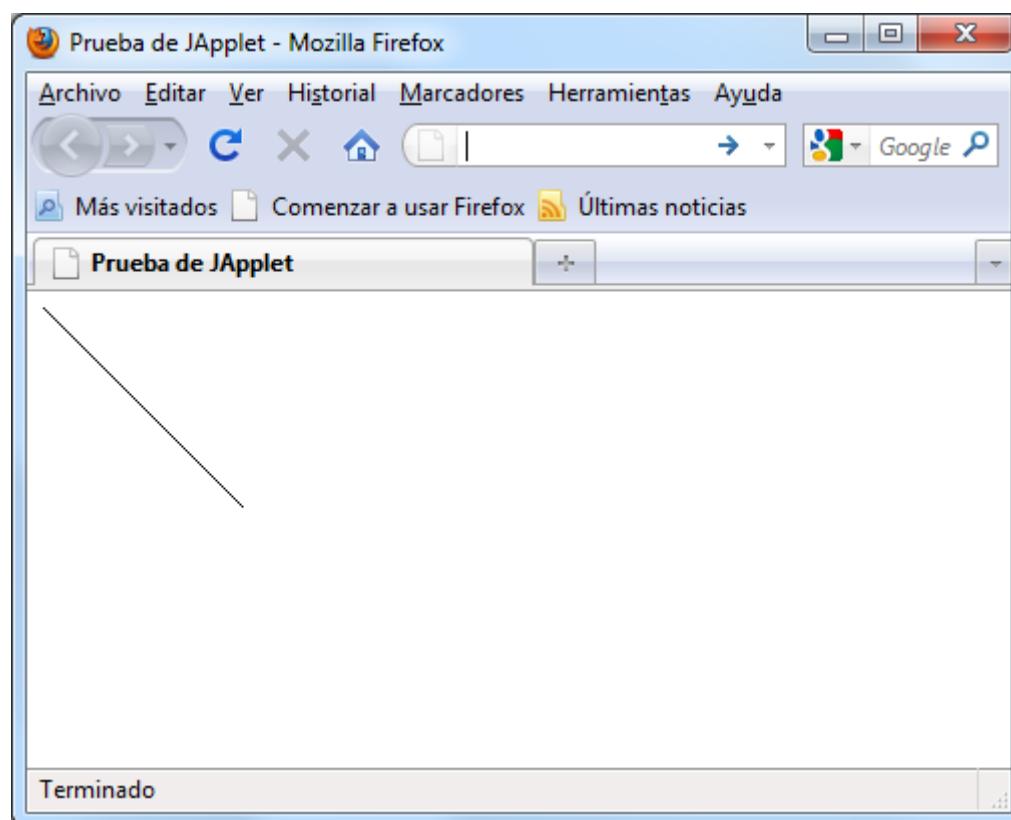


Figura 37: Ejecutando como aplicación

## 12. Animación y control haciendo uso de JPanel

En este ejemplo, tendremos una animación interactiva: un círculo rebotando por la ventana. La interacción es que el usuario puede controlar la animación (iniciándola y deteniéndola) usando botones de tipo JButton.

Este programa ejecuta como aplicación (usando JFrame) y como Applet (usando JApplet).

### Inicio.java

```
/* Ejemplo de animación interactiva en Java con componentes Swing que ejecuta como aplicación de escritorio y como applet */
/*
//Necesario para usar Swing
import java.awt.BorderLayout;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JFrame;

public class Inicio extends JApplet
{
    //Botones que controlan el inicio y fin del hilo
    JButton btnIniciar = new JButton();
    JButton btnDetener = new JButton();
    Dibujar objDibuja = new Dibujar();

    //Ejecutado si es applet
    public void init()
    {
        ControlesGraficos();
        ControlEventos();
    }

    //Ejecutado si es applet (cuando cierran la página debe detener el hilo)
    public void destroy()
    {
        objDibuja.Detener();
    }

    //Si ejecuta como aplicación
    public static void main(String[] args)
    {
        Inicio Aplicacion = new Inicio();
        Aplicacion.EjecutaEscritorio();
    }

    public void EjecutaEscritorio()
    {
        JFrame Ventana = new JFrame("Animación interactiva");

        //Tamaño de la ventana
        Ventana.setSize(750, 570);

        //Instancia el applet
        Inicio applet = new Inicio();

        //Inicializa el applet
        applet.init();

        //Adiciona el applet al JFrame
        Ventana.add(applet, BorderLayout.CENTER);
        Ventana.setVisible(true);

        //Evento de cerrar ventana (detiene el hilo primero)
        Ventana.addWindowListener(
        {
            new java.awt.event.WindowAdapter()
            {
                public void windowClosing(java.awt.event.WindowEvent e)
                {
                    objDibuja.Detener();
                    System.exit(0);
                }
            }
        });
    }

    public void ControlesGraficos()
    {
        //Adiciona los botones
        btnIniciar.setText("Iniciar o Continuar");
        btnDetener.setText("Detener");
        add(btnIniciar, BorderLayout.NORTH);
        add(btnDetener, BorderLayout.SOUTH);

        //El componente que grafica
        add(objDibuja);
        objDibuja.Iniciar();

        //Ubicación y tamaño de la ventana
        setBounds(0, 0, 400, 400);

        //Muestra la ventana
    }
}
```

```

setVisible(true);

}

public void ControlEventos()
{
//Manejo de eventos. Cuando hacemos clic en botón Iniciar
btnIniciar.addActionListener(
(new java.awt.event.ActionListener(){
    public void actionPerformed(java.awt.event.ActionEvent e)
    {
        objDibuja.Iniciar();
    }
}));

//Manejo de eventos. Cuando hacemos clic en botón Detener
btnDetener.addActionListener(
(new java.awt.event.ActionListener(){
    public void actionPerformed(java.awt.event.ActionEvent e)
    {
        objDibuja.Detener();
    }
}));
}
}

```

**Dibujar.java**

```

import java.awt.*;
import javax.swing.JPanel;

//Para usar hilos debe heredar de Runnable
public class Dibujar extends JPanel implements Runnable
{
//Hilo que controla la animación
Thread objHilo;

//Controla que run() continue
boolean Ejecuta = false;

//Posición del círculo relleno
int PosX=180, PosY=10;

//Desplazamiento del círculo
int IncX=1, IncY=1;

//Diámetro del círculo
int Diametro = 80;

//Método ejecutado por el hilo
public void run()
{
    while(Ejecuta)
    {
        repaint(); //Repinta la pantalla

        try{Thread.sleep(5);}catch (InterruptedException E){}
    }
}

//Dibuja el círculo
public void paintComponent (Graphics objGrafico)
{
//Limpia la pantalla primero
objGrafico.setColor(Color.white);
objGrafico.fillRect(0, 0, getSize().width, getSize().height);

//Mueve el círculo
PosX+=IncX;
PosY+=IncY;

//Dibuja el círculo
objGrafico.setColor(Color.blue);
objGrafico.fillOval(PosX, PosY, Diametro, Diametro);

//Chequea si rebota contra una pared.
if (PosX<=0 || PosX+Diametro >= getSize().width) IncX*=-1;
if (PosY<=0 || PosY+Diametro >= getSize().height) IncY*=-1;
}

public void Iniciar()
{
if (Ejecuta==false)
{
    objHilo = new Thread(this);
    Ejecuta = true;
    objHilo.start();
}
}

public void Detener()
{
}

```

```
Ejecuta = false;
}
}
```

## Animacion.htm

```
<html><body>
<table border=0 width="100%" id=table1>
<tr><td>
<p><a href="http://darwin.50webs.com" class=blue>Web del autor</a>
</td></tr></table>
<p>Animación interactiva usando JPanel y JApplet</p>
<p><applet code="Inicio" width="330" height="230" archive="AnimacionInteractiva.jar"></applet>
</p>
<p>Círculo relleno rebotando en la pantalla</p>
</body></html>
```

Ejemplo de ejecución:

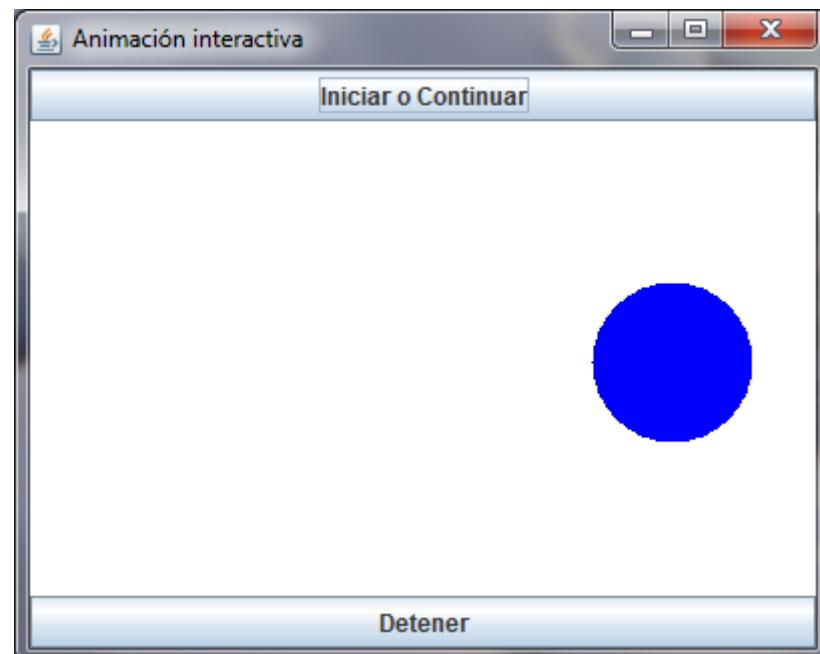


Figura 38: Ejecutando como aplicación

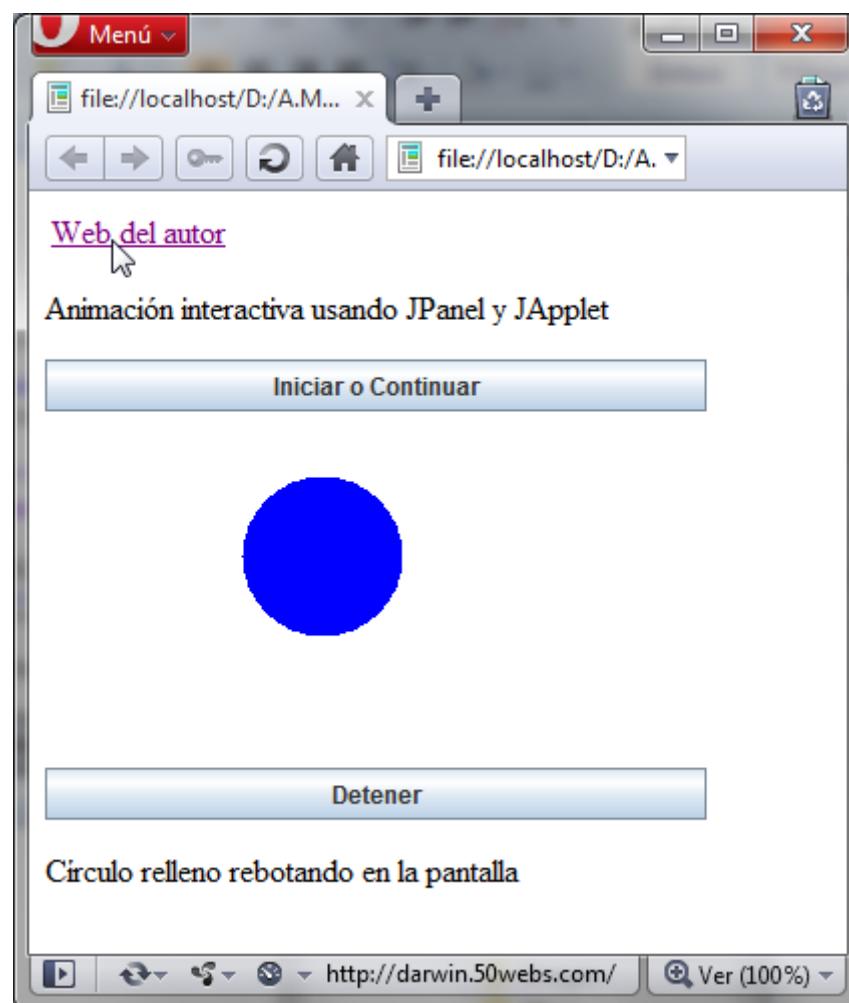


Figura 39: Ejecutando como applet

## 13. Juego reescrito usando el componente JPanel

El juego visto en el capítulo 9 se reescribe usando los controles Swing. Las ventajas de hacer uso de estos componentes son: controles gráficos más ligeros y modernos y el double buffer es automático con los JPanel. Los cambios con respecto al código presentado en el capítulo 9 son pocos.

### InicioAnimacion.java

```
/* Inicia el videojuego */
import java.awt.*;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JFrame;

public class InicioAnimacion extends JApplet
{
    //Objeto que muestra el gráfico animado
    Animar objAnimar = new Animar();

    //Botones que controlan el inicio y fin del hilo
    JButton btnIniciar = new JButton();
    JButton btnDetener = new JButton();

    //Ejecutado si es applet
    public void init()
    {
        ControlesGraficos();
        ControlEventos();
    }

    //Ejecutado si es applet (cuando cierran la página debe detener el hilo)
    public void destroy()
    {
        objAnimar.Detener();
    }

    //Si ejecuta como aplicación
    public static void main(String[] args)
    {
        InicioAnimacion Aplicacion = new InicioAnimacion();
        Aplicacion.EjecutaEscritorio();
    }

    public void EjecutaEscritorio()
    {
        JFrame Ventana = new JFrame("Derribar basura espacial");

        //Tamaño de la ventana
        Ventana.setSize(750, 570);

        //Instancia el applet
        InicioAnimacion applet = new InicioAnimacion();

        //Inicializa el applet
        applet.init();

        //Adiciona el applet al JFrame
        Ventana.add(applet, BorderLayout.CENTER);
        Ventana.setVisible(true);

        //Evento de cerrar ventana (detiene el hilo primero)
        Ventana.addWindowListener(
        {
            new java.awt.event.WindowAdapter()
            {
                public void windowClosing(java.awt.event.WindowEvent e)
                {
                    objAnimar.Detener();
                    System.exit(0);
                }
            }
        });
    }

    public void ControlesGraficos()
    {
        //Adiciona los botones
        btnIniciar.setText("Iniciar o Continuar");
        btnDetener.setText("Detener");
        add(btnIniciar, BorderLayout.NORTH);
        add(btnDetener, BorderLayout.SOUTH);

        //El componente que grafica
        add(objAnimar);
        objAnimar.Iniciar();

        //Ubicación y tamaño de la ventana
        setBounds(0, 0, 400, 400);

        //Muestra la ventana
        setVisible(true);
    }
}
```

```

public void ControlEventos()
{
    //Manejo de eventos. Cuando hacemos clic en botón Iniciar
    btnIniciar.addActionListener(
        (new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e)
            {
                objAnimar.Iniciar();
            }
        }));
    //Manejo de eventos. Cuando hacemos clic en botón Detener
    btnDetener.addActionListener(
        (new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e)
            {
                objAnimar.Detener();
            }
        }));
}
}

```

Nota: Para poder capturar los eventos de teclado dentro de un JPanel es necesario hacer que el JPanel pueda tener el foco con la instrucción setFocusable(true); y con el evento del ratón MouseEntered se le de el foco al JPanel con la instrucción this.requestFocus();

#### Animar.java

```

/* Esa clase se encarga del hilo de ejecución del videojuego */
import java.awt.*;
import java.util.*;
import java.awt.event.*;
import java.applet.AudioClip;
import javax.swing.JPanel;

//Para usar hilos debe heredar de Runnable
public class Animar extends JPanel implements Runnable, MouseMotionListener, MouseListener, KeyListener
{
    //Hilo que controla la animación
    Thread objHilo;

    //Objeto que inicia los números aleatorios
    Random Azar = new Random();

    //Controla el hilo
    boolean Ejecuta = false;

    //Constantes del juego
    boolean INACTIVO = false;

    //Instancia la base y lanzador
    BaseLanzador objBaseLanzador = new BaseLanzador();
    Lanzador objLanzador = new Lanzador();
    int Angulo = 270;

    //Instancia los rayos como lista
    int TOTALRAYOS = 10;
    Rayo ListaRayo[] = new Rayo[TOTALRAYOS];

    //Instancia la chatarra como lista
    int TOTALCHATARRA = 3;
    Blanco ListaChatarra[] = new Blanco[TOTALCHATARRA];

    //Instancia los satélites como lista
    int TOTALSATELITE = 4;
    Blanco ListaSatelite[] = new Blanco[TOTALSATELITE];

    //Puntaje del juego
    int Puntaje = 0;

    //Valor X del mouse anterior
    double MousePosXAnterior = 0;

    //Recursos de imagen y sonido
    GestionRecursos Recurso = new GestionRecursos();
    AudioClip SonidoEjecuta;

    //Activa la captura de eventos
    public Animar()
    {
        setFocusable(true);
        addMouseMotionListener(this);
        addMouseListener(this);
        addKeyListener(this);

        //Inicia la lista de rayos, chatarra, satélites
        for (int Cont=0; Cont<TOTALRAYOS; Cont++)
            ListaRayo[Cont]= new Rayo();

        for (int Cont=0; Cont<TOTALCHATARRA; Cont++)
            ListaChatarra[Cont]= new Blanco(1);
    }
}

```

```

for (int Cont=0; Cont<TOTALSATELITE; Cont++)
    ListaSatelite[Cont] = new Blanco(2);

//Carga el sonido de la explosión de la colisión
SonidoEjecuta = Recurso.RetornaSonido("Explosion.wav");
}

//De MouseMotionListener
public void mouseMoved(MouseEvent objMouse)
{
    //Dependiendo del movimiento del ratón mueve el cañón
    Angulo+=objMouse.getPoint().getX()-MousePosXAnterior;
    if (Angulo > 360) Angulo = 360;
    if (Angulo < 270) Angulo = 270;
    MousePosXAnterior = objMouse.getPoint().getX();
}

public void mouseDragged(MouseEvent objMouse){}

//De MouseListener
public void mousePressed (MouseEvent objMouse){}
public void mouseClicked (MouseEvent objMouse){}
public void mouseEntered (MouseEvent objMouse){ this.requestFocus(); }
public void mouseExited (MouseEvent objMouse){}
public void mouseReleased (MouseEvent objMouse){}

//De KeyListener
public void keyTyped(KeyEvent e){}
public void keyReleased(KeyEvent e){}

public void keyPressed(KeyEvent e)
{
    //Gira el cañón
    if (e.getKeyCode() == KeyEvent.VK_RIGHT) Angulo+=3;
    if (e.getKeyCode() == KeyEvent.VK_LEFT) Angulo-=3;

    //Limita cuanto se puede girar el cañón
    if (Angulo > 360) Angulo = 360;
    if (Angulo < 270) Angulo = 270;

    //Si se dispara un rayo
    if (e.getKeyCode() == KeyEvent.VK_SPACE)
        DispararRayo();
}

public void DispararRayo()
{
    //Busca algún rayo inactivo
    for(int Cont=0; Cont<TOTALRAYOS; Cont++)
        if (ListaRayo[Cont].getEstado ()==INACTIVO)
    {
        ListaRayo[Cont].Disparar((float) objLanzador.getPosX(), (float) objLanzador.getPosY(), (float) objLanzador.getAngulo(),
3.0f);
        break;
    }
}

//Método ejecutado por el hilo
public void run()
{
    while(Ejecuta)
    {
        //Chequea si algún rayo colisiona con alguna chatarra o satélite
        Puntaje += ChequeaColisiones();

        repaint();

        try{Thread.sleep(10);}catch (InterruptedException E){}
    }
}

//Chequea si algún rayo colisiona con chatarra o satélite
public int ChequeaColisiones()
{
    //Va de rayo en rayo
    for (int Cont=0; Cont<TOTALRAYOS; Cont++)
    {
        //Si el rayo está inactivo pasa al siguiente
        if (ListaRayo[Cont].getEstado ()==INACTIVO) continue;

        //Deduces un rectángulo para ese rayo
        Rectangle Rect1 = new Rectangle(ListaRayo[Cont].getPosX(), ListaRayo[Cont].getPosY(), ListaRayo[Cont].getAncho(),
        ListaRayo[Cont].getAlto());

        //Va de chatarra en chatarra
        for (int Chatarra=0; Chatarra<TOTALCHATARRA; Chatarra++)
        {
            //Si la chatarra está inactiva pasa a la siguiente
            if (ListaChatarra[Chatarra].getEstado ()==INACTIVO) continue;

            //Deduces un rectángulo para esa chatarra
            Rectangle Rect2 = new Rectangle(ListaChatarra[Chatarra].getPosX(), ListaChatarra[Chatarra].getPosY(),
            ListaChatarra[Chatarra].getAncho(), ListaChatarra[Chatarra].getAlto());
        }
    }
}

```

```

//Si hay intersección entre esos dos rectángulos, entonces hay colisión
if (Rect1.intersects(Rect2))
{
    //Inactiva el rayo
    ListaRayo[Cont].setEstado(INACTIVO);

    //Inactiva la chatarra
    ListaChatarra[Chatarra].setEstado(INACTIVO);

    //Ejecuta sonido de explosión
    SonidoEjecuta.play();

    //Retorna 1 para el puntaje
    return 1;
}

//Va de satélite en satélite
for (int Satelite=0; Satelite<TOTALSATELITE; Satelite++)
{
    //Si el satélite está inactiva pasa al siguiente
    if (ListaSatelite[Satelite].getEstado()==INACTIVO) continue;

    //Deduce un rectángulo para ese satélite
    Rectangle Rect2 = new Rectangle(ListaSatelite[Satelite].getPosX(), ListaSatelite[Satelite].getPosY(),
    ListaSatelite[Satelite].getAncho(), ListaSatelite[Satelite].getAlto());

    //Si hay intersección entre esos dos rectángulos, entonces hay colisión
    if (Rect1.intersects(Rect2))
    {
        //Inactiva el rayo
        ListaRayo[Cont].setEstado(INACTIVO);

        //Inactiva el satélite
        ListaSatelite[Satelite].setEstado(INACTIVO);

        //Ejecuta sonido de explosión
        SonidoEjecuta.play();

        //Retorna 1 para el puntaje
        return -1;
    }
}

//No hubo colisión, luego no hay puntaje
return 0;
}

//Actualiza y pinta los protagonistas del videojuego
public void paintComponent (Graphics objGrafico)
{
    //Limpia la pantalla primero
    objGrafico.setColor(Color.white);
    objGrafico.fillRect(0, 0, getSize().width, getSize().height);

    objBaseLanzador.Pintar(objGrafico);
    objLanzador.Pintar(objGrafico, Angulo);

    //Actualiza los rayos, chatarra, satélites y los dibuja
    for (int Cont=0; Cont<TOTALRAYOS; Cont++)
        ListaRayo[Cont].Pintar(objGrafico, getSize().width, getSize().height);

    for (int Cont=0; Cont<TOTALCHATARRA; Cont++)
        ListaChatarra[Cont].Pintar(objGrafico, Azar, getSize().width, getSize().height, (int) (getSize().height*0.01), (int)
        (getSize().height*0.5), 2, 5);

    for (int Cont=0; Cont<TOTALSATELITE; Cont++)
        ListaSatelite[Cont].Pintar(objGrafico, Azar, getSize().width, getSize().height, (int) (getSize().height*0.1), (int)
        (getSize().height*0.5), 1, 4);

    //Muestra el puntaje
    objGrafico.drawString("Puntaje: " + Puntaje, 10, 10);
}

//Inicia el hilo
public void Iniciar()
{
    if (Ejecuta==false)
    {
        objHilo = new Thread(this);
        Ejecuta = true;
        objHilo.start();
    }
}

//Detiene el hilo
public void Detener()
{
    Ejecuta = false;
}

```

{

## Rayo.java

```

/* Manejo de los Rayos */
import java.awt.*;
import java.applet.AudioClip;
import java.awt.image.BufferedImage;
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;
import java.awt.image.AffineTransformOp;
import javax.swing.JPanel;

public class Rayo extends JPanel
{
    //Estado del Rayo
    boolean Estado;

    //Posición del Rayo
    float PosX, PosY;

    //Avance del Rayo
    float Angulo, IncX, IncY;

    //Sonido del despegue
    GestionRecursos Recurso = new GestionRecursos();
    AudioClip SonidoEjecuta;

    //Imagen del Rayo
    BufferedImage Imagen;

    //Giro de la imagen
    AffineTransformOp Transformacion;

    //Retorna los valores de los atributos
    public boolean getEstado(){ return Estado; }
    public int getPosX(){ return (int) PosX; }
    public int getPosY(){ return (int) PosY; }
    public int getAncho(){ return Imagen.getWidth(); }
    public int getAlto(){ return Imagen.getHeight(); }

    //Da valores a los atributos
    public void setEstado(boolean Estado) { this.Estado = Estado; }

    //Carga la imagen y sonido del Rayo. Inicializa valores.
    public Rayo()
    {
        SonidoEjecuta = Recurso.RetornaSonido("Disparo.wav");
        Imagen = Recurso.RetornaImagen("Rayo.png");
        PosX = 0;
        PosY = 0;
        Angulo = 0;
        IncX = 0;
        IncY = 0;
        Estado = false;
    }

    //Dispara el rayo, lo activa y pone los valores iniciales
    public void Disparar(float PosX, float PosY, float Angulo, float Velocidad)
    {
        Estado = true;
        this.PosX = PosX;
        this.PosY = PosY;
        this.Angulo = Angulo;
        this.IncX = (float) (Velocidad*Math.cos(Angulo*Math.PI/180));
        this.IncY = (float) (Velocidad*Math.sin(Angulo*Math.PI/180));

        //Las siguientes líneas sirven para transformar la imagen (girarla)
        AffineTransform GiraImagen = new AffineTransform();
        GiraImagen.rotate(this.Angulo*Math.PI/180, Imagen.getWidth()/2, Imagen.getHeight()/2);
        Transformacion = new AffineTransformOp(GiraImagen, AffineTransformOp.TYPE_BILINEAR);

        //Ejecuta sonido de disparo
        SonidoEjecuta.play();
    }

    //Actualiza la posición del rayo si está activo y luego lo dibuja
    public void Pintar(Graphics objGrafico, int Ancho, int Alto)
    {
        if (Estado)
        {
            //Mueve el rayo
            PosX+=IncX;
            PosY+=IncY;

            //Se desactiva si se sale de la pantalla
            if (PosX > Ancho || PosY > Alto || PosX < 0 || PosY < 0)
                Estado = false;

            //Se requiere Java 2D
            Graphics2D Graficos2D = (Graphics2D) objGrafico;
            Graficos2D.drawImage(Imagen, Transformacion, (int) PosX, (int) PosY);
        }
    }
}

```

```
}
```

**Lanzador.java**

```
/*
 * Esta clase se encarga del lanzador */
import java.awt.*;
import java.awt.image.BufferedImage;
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;
import java.awt.image.AffineTransformOp;
import javax.swing.JPanel;

public class Lanzador extends JPanel
{
    //Posición del lanzador al inicio
    int PosX=50, PosY=290;

    //Angulo del lanzador en grados
    int Angulo = 0;

    //Imagen del lanzador
    GestionRecursos Recurso = new GestionRecursos();
    BufferedImage Imagen;

    //retorna valores
    public int getPosX() { return PosX; }
    public int getPosY() { return PosY; }
    public int getAngulo() { return Angulo; }

    //Carga la imagen del lanzador
    public Lanzador()
    {
        Imagen = Recurso.RetornaImagen("Lanzador.png");
    }

    public void Pintar(Graphics objGrafico, int Angulo)
    {
        this.Angulo = Angulo;

        //Las siguientes líneas sirven para transformar la imagen (girarla)
        AffineTransform GiraImagen = new AffineTransform();
        GiraImagen.rotate(this.Angulo*Math.PI/180, Imagen.getWidth()/2, Imagen.getHeight()/2);
        AffineTransformOp Transformacion = new AffineTransformOp(GiraImagen, AffineTransformOp.TYPE_BILINEAR);

        //Se requiere Java 2D
        Graphics2D Graficos2D = (Graphics2D) objGrafico;
        Graficos2D.drawImage(Imagen, Transformacion, PosX, PosY);
    }
}
```

**GestionRecursos.java**

```
/*
 * Esta clase se encarga de los recursos de imagen y sonido
 * que tendrá el juego. Sus funciones son cargar el recurso
 * de disco (haciendo operaciones de archivo) y ponerlo en una
 * lista HashMap para que sea usado rápidamente */

import java.util.HashMap;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.net.*;
import java.applet.*;

public class GestionRecursos
{
    //Almacena los sonidos en una lista
    HashMap ListaSonidos = new HashMap();

    //Almacena las imágenes en una lista
    HashMap ListaImagenes = new HashMap();

    //Carga el sonido de disco
    public AudioClip CargaSonido(String Direccion)
    {
        try
        {
            URL RutaSonido = getClass().getResource(Direccion);
            AudioClip Sonido = Applet.newAudioClip(RutaSonido);
            return Sonido;
        }
        catch (Exception e)
        {
            System.out.println("Problemas en cargar archivo");
            return null;
        }
    }
}
```

```

//Si el sonido no está en memoria (verificando la lista)
//lo carga de disco
public AudioClip RetornaSonido(String NombreSonido)
{
    //Busca el sonido en la lista
    AudioClip Sonido = (AudioClip) ListaSonidos.get(NombreSonido);

    //Si no encuentra el sonido en la lista, lo carga de disco
    if (Sonido == null)
    {
        Sonido = CargaSonido(NombreSonido);
        ListaSonidos.put(NombreSonido, Sonido);
    }
    return Sonido;
}

//Carga la imagen de disco
public BufferedImage CargaImagen(String Direccion)
{
    try
    {
        URL RutaImagen = getClass().getResource(Direccion);
        BufferedImage Imagen = ImageIO.read(RutaImagen);
        return Imagen;
    }
    catch (Exception e)
    {
        System.out.println("Problemas en cargar archivo");
        return null;
    }
}

//Si la imagen no está en memoria (verificando la lista)
//la carga de disco
public BufferedImage RetornaImagen(String NombreImagen)
{
    //Busca la imagen en la lista
    BufferedImage Imagen = (BufferedImage) ListaImagenes.get(NombreImagen);

    //Si no encuentra la imagen en la lista, la carga de disco
    if (Imagen == null)
    {
        Imagen = CargaImagen(NombreImagen);
        ListaImagenes.put(NombreImagen, Imagen);
    }
    return Imagen;
}
}

```

**Blanco.java**

```

/* Esta clase maneja la chatarra y los satélites */
import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.*;
import javax.swing.JPanel;

public class Blanco extends JPanel
{
    //Activo o inactivo el blanco
    boolean Estado;

    //Posición del blanco
    int PosX, PosY;

    //Desplazamiento del blanco
    int Velocidad;

    //Recurso de imagen
    GestionRecursos Recurso = new GestionRecursos();

    //Imagen de la chatarra
    BufferedImage Imagen;

    //Retorna los valores de los atributos
    public boolean getEstado(){ return Estado; }
    public int getPosX(){ return (int) PosX; }
    public int getPosY(){ return (int) PosY; }
    public int getAncho(){ return Imagen.getWidth(); }
    public int getAlto(){ return Imagen.getHeight(); }

    //Da valores a los atributos
    public void setEstado(boolean Estado) { this.Estado = Estado; }

    //Carga la imagen del blanco
    public Blanco(int TipoBlanco)
    {
        if (TipoBlanco==1) Imagen = Recurso.RetornaImagen("BasuraEspacial.png");
        if (TipoBlanco==2) Imagen = Recurso.RetornaImagen("Satelite.png");
        PosX = 0;
        PosY = 0;
    }
}

```

```

Velocidad = 0;
Estado = false;
}

public void Pintar(Graphics objGrafico, Random Azar, int Ancho, int Alto, int AlturaMax, int AlturaMin, int VelocidadMax, int
VelocidadMin)
{
if (Estado==true)
{
//Mueve la chatarra
PosX+=Velocidad;

//Chequea si se sale de la ventana
if (PosX<0 || PosX+Imagen.getWidth() > Ancho)
Estado = false;

//Muestra la imagen del blanco
objGrafico.drawImage(Imagen, PosX, PosY, this);
}
else
{ //Activa la chatarra/satélite con altura y velocidad al azar
Estado = true;
this.PosX = 0;
PosY = (int) (Azar.nextFloat()*(AlturaMax-AlturaMin)+AlturaMin);
Velocidad = (int) (Azar.nextFloat()*(VelocidadMax-VelocidadMin)+VelocidadMin);
}
}
}
}

```

**BaseLanzador.java**

```

/* Esta clase maneja la base del lanzador */
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.swing.JPanel;

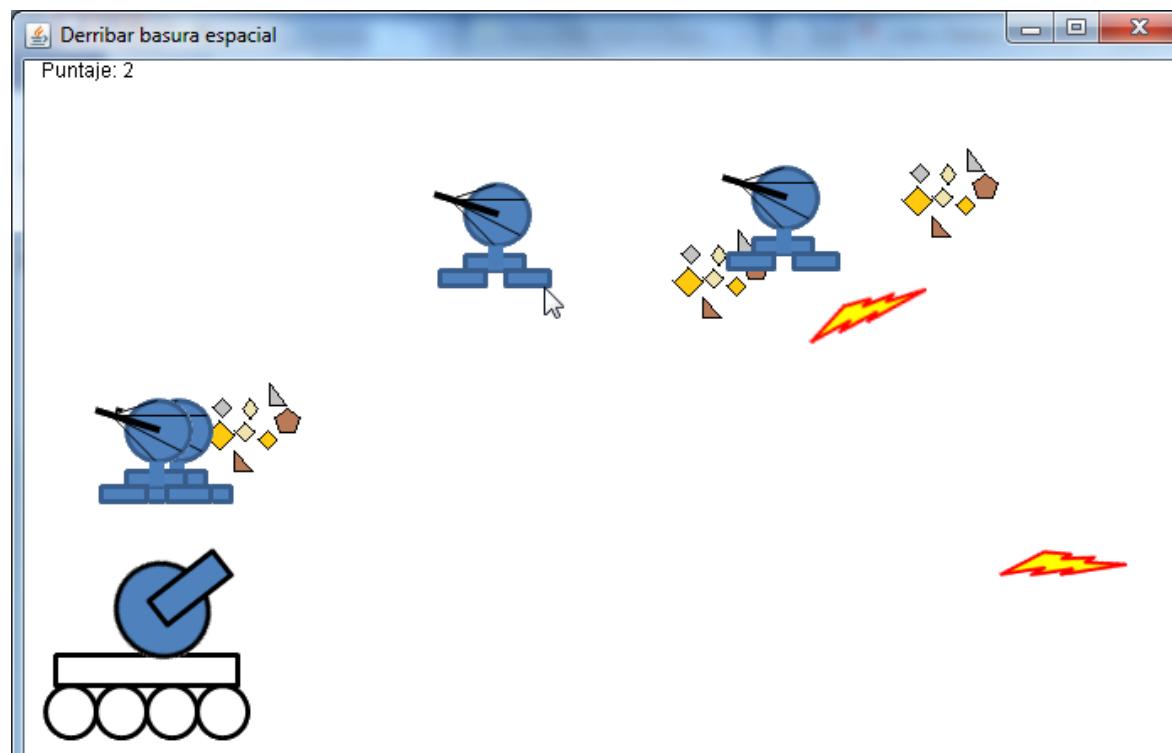
public class BaseLanzador extends JPanel
{
//Posición de la base del lanzador
int PosX=10, PosY=350;

//Imagen del Lanzador
GestionRecursos Recurso = new GestionRecursos();
BufferedImage Imagen;

//Carga la imagen del lanzador
public BaseLanzador()
{
Imagen = Recurso.RetornaImagen("BaseLanzador.png");
}

public void Pintar(Graphics objGrafico)
{
//Muestra la imagen de la base del lanzador
objGrafico.drawImage(Imagen, PosX, PosY, this);
}
}

```

**Figura 40: El juego implementado con JPanel**

## 14. XNA: Instalación de Microsoft XNA Game Studio 3.1

Para trabajar con XNA (Reed, 2008) (Cawood, y otros, 2007) se requiere instalar Visual C# 2008 .NET (Harris, 2002) y el paquete XNA Game Studio 3.1 que pueden descargárselo de este sitio <http://www.microsoft.com/downloads/details.aspx?FamilyID=80782277-d584-42d2-8024-893fc9d3e82&displaylang=en> como se observa en la imagen:

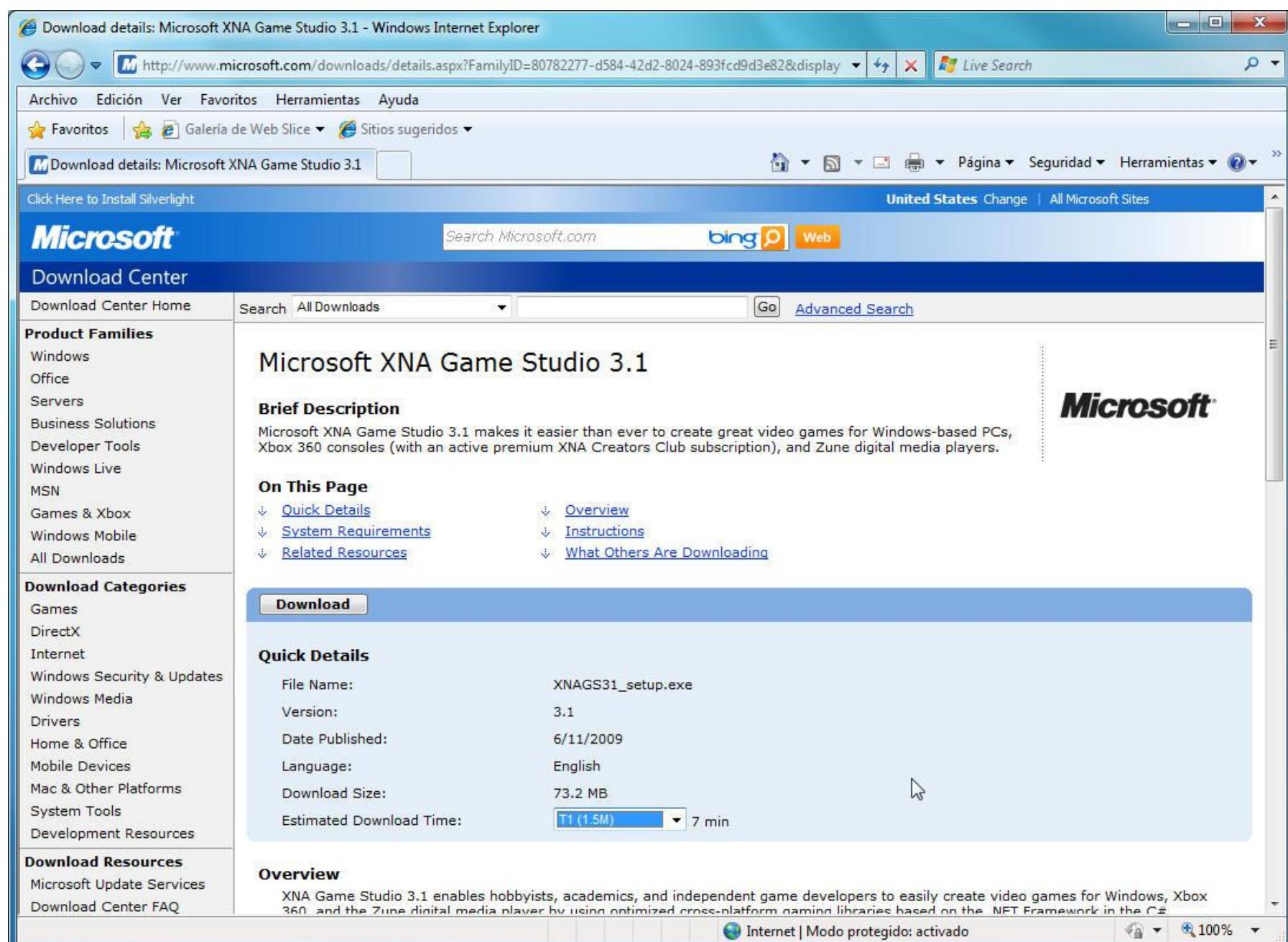
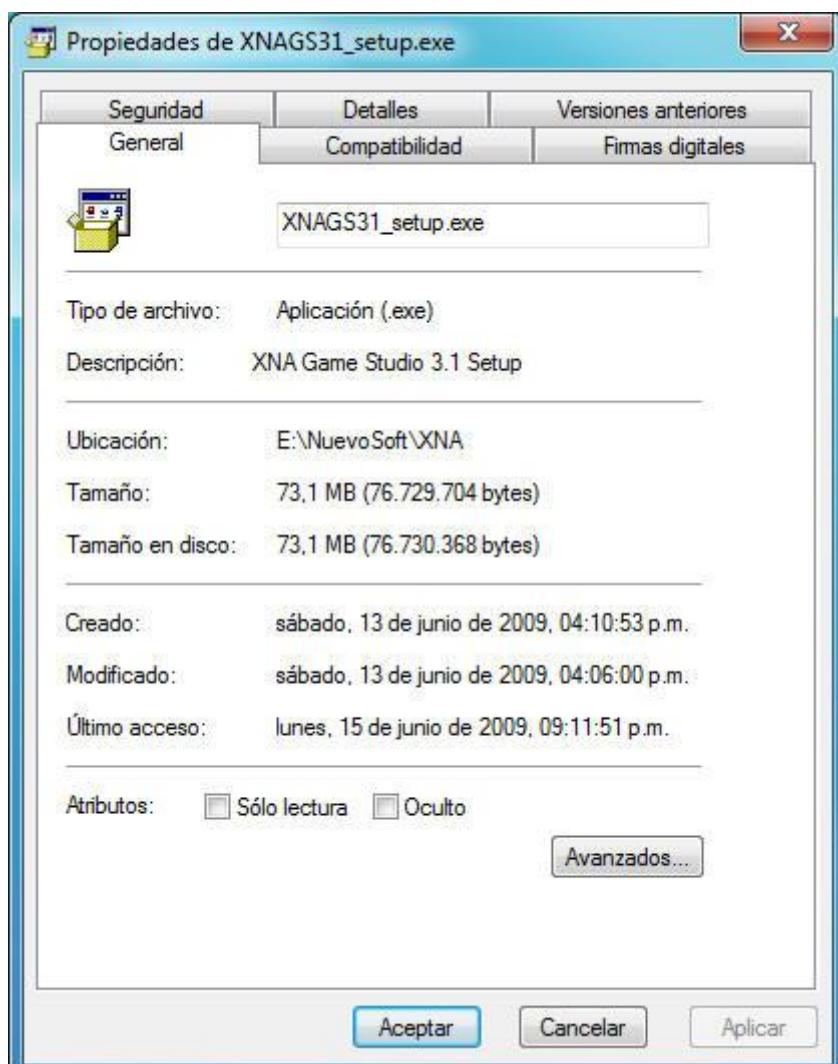


Figura 41: Dirección en la cual se descarga XNA 3.1

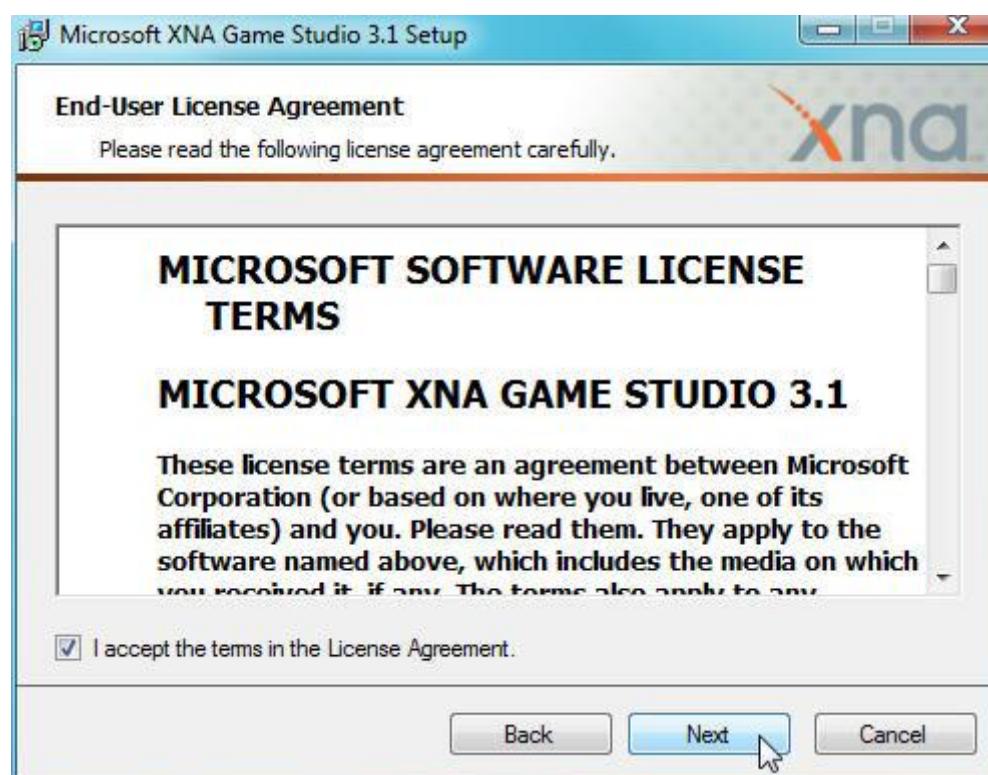


Figura 42: Tamaño del instalador de XNA 3.1

**Figura 43: Características del instalador de XNA 3.1**

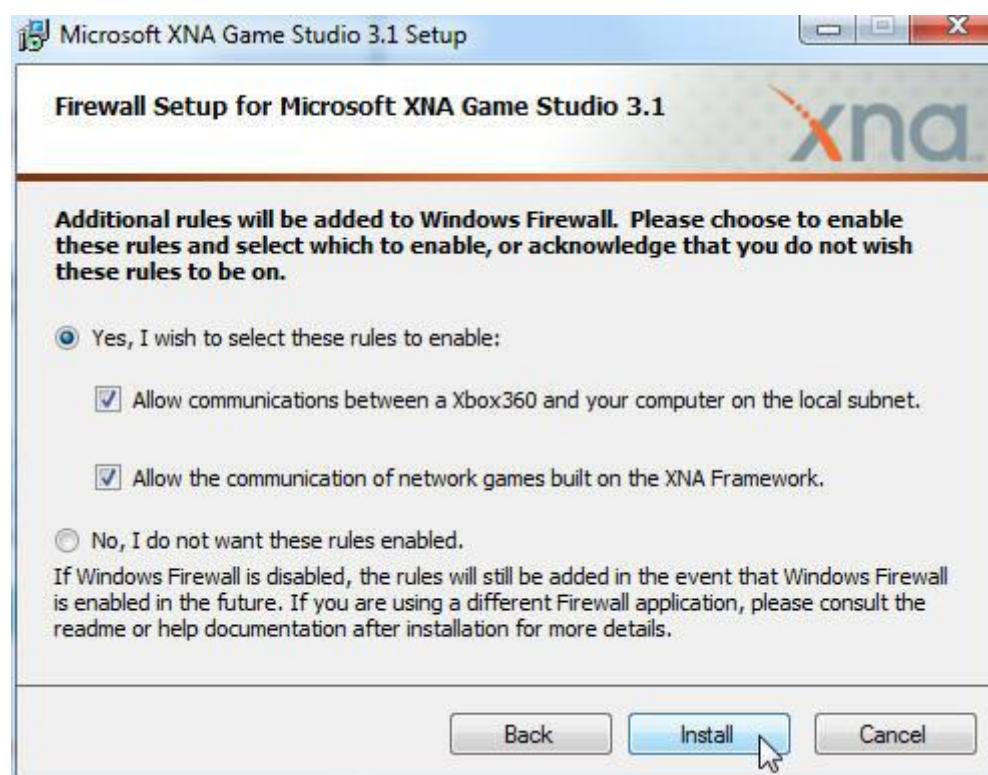
Al dar doble clic en este empieza la instalación del paquete

**Figura 44: Inicio de la instalación de Microsoft XNA Game Studio 3.1**

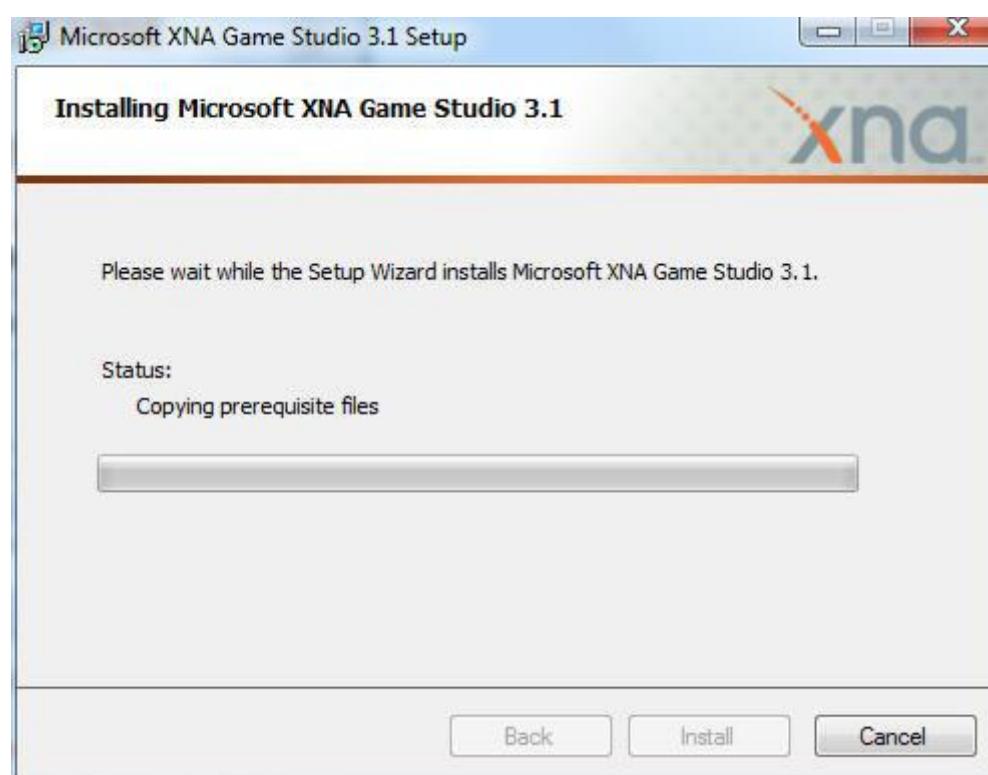


**Figura 45:** Licencia de uso de Microsoft XNA Game Studio 3.1

Permita que el instalador agregue algunas reglas al "firewall" de Windows



**Figura 46:** Configuración del "firewall" con XNA



**Figura 47:** Copying prerequisite files

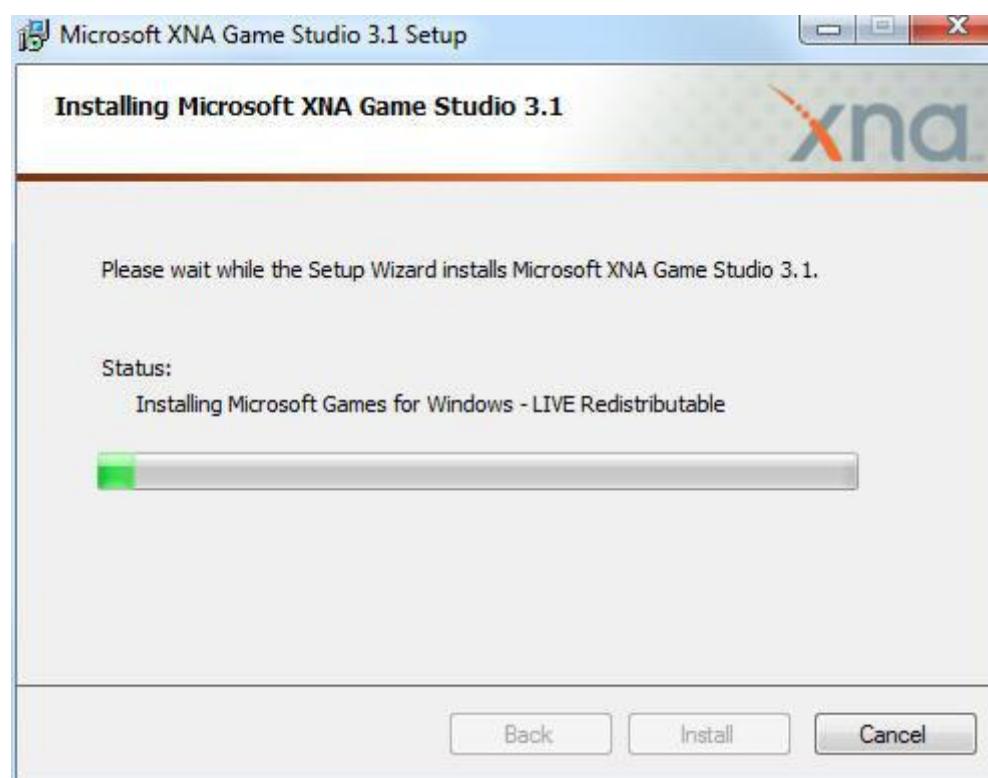


Figura 48: Installing Microsoft Games for Windows

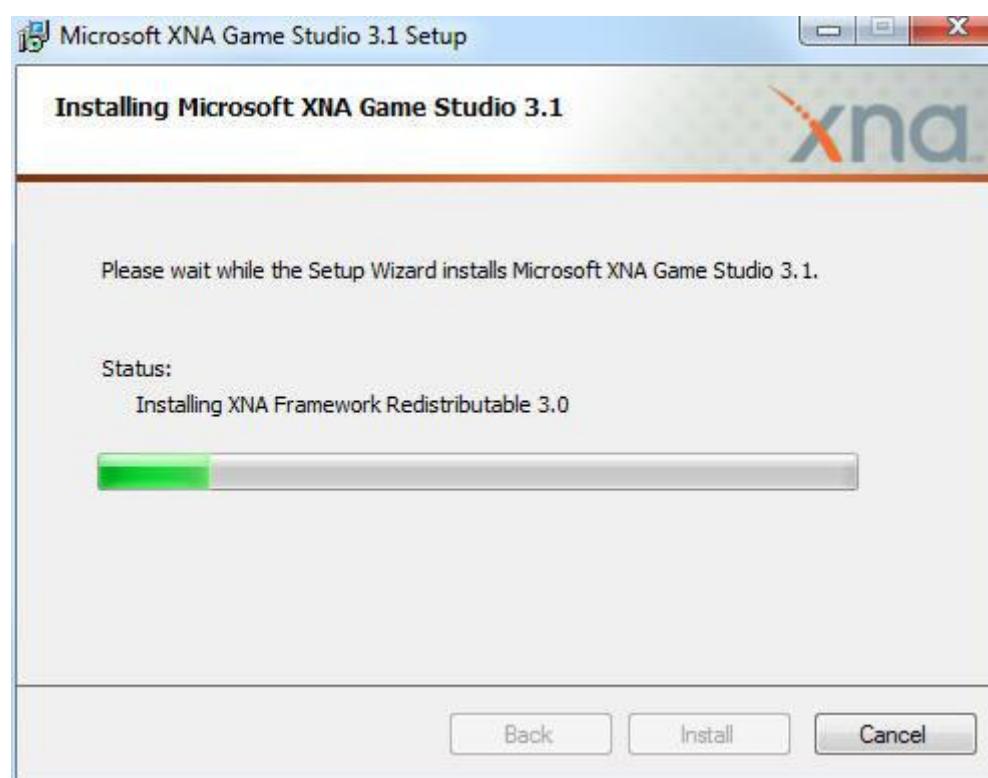


Figura 49: Installing XNA Framework Redistributable 3.0

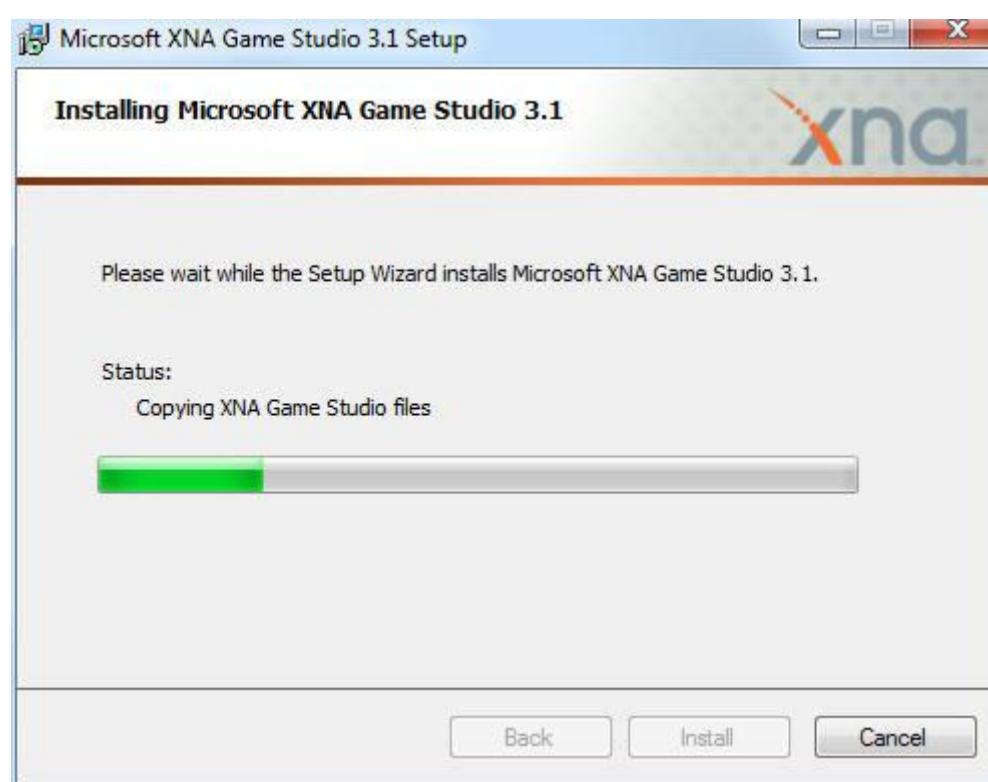
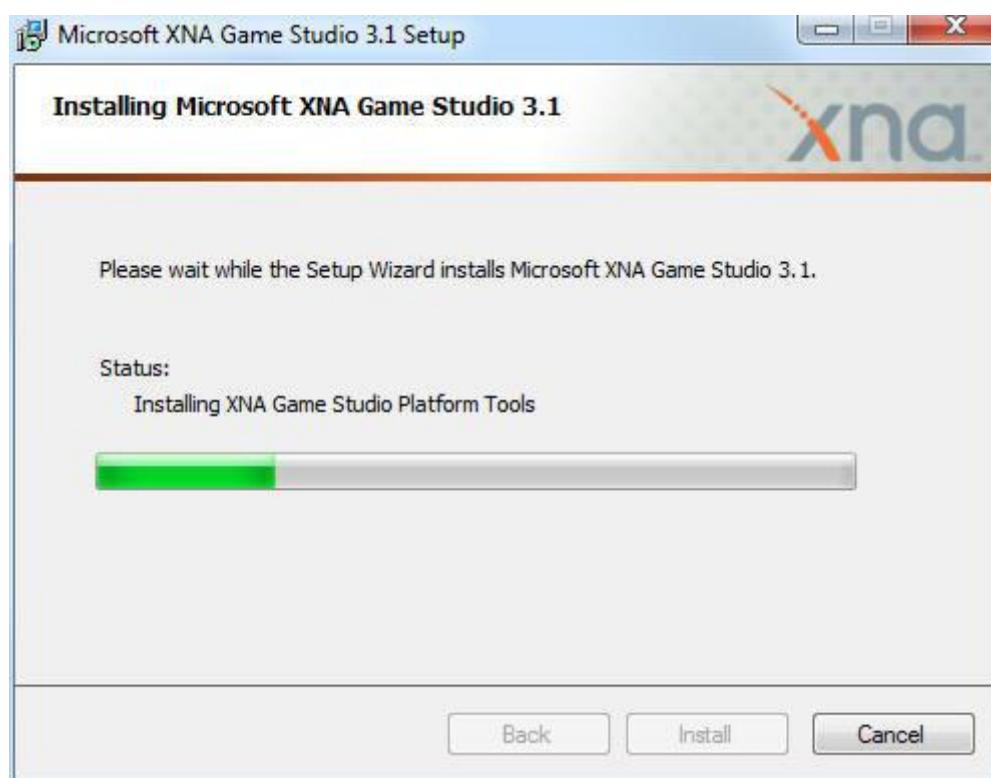
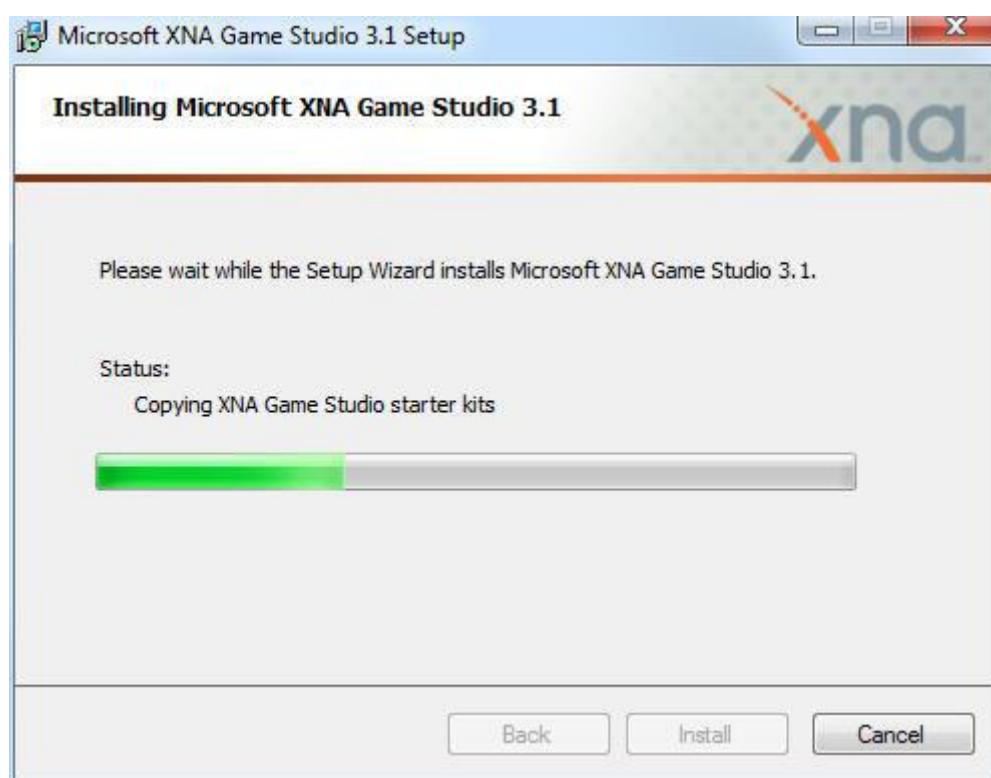


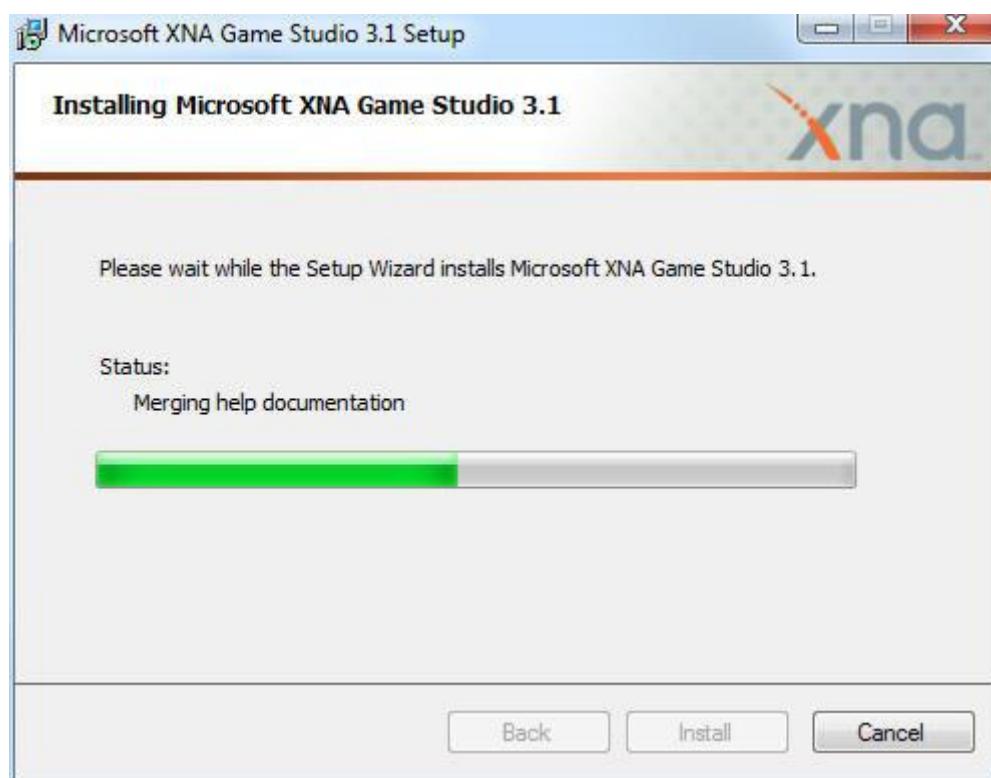
Figura 50: Copying XNA Game Studio files



**Figura 51: Installing XNA Game Studio Platform Tools**



**Figura 52: Copying XNA Game Studio starter kits**



**Figura 53: Merging help documentation**

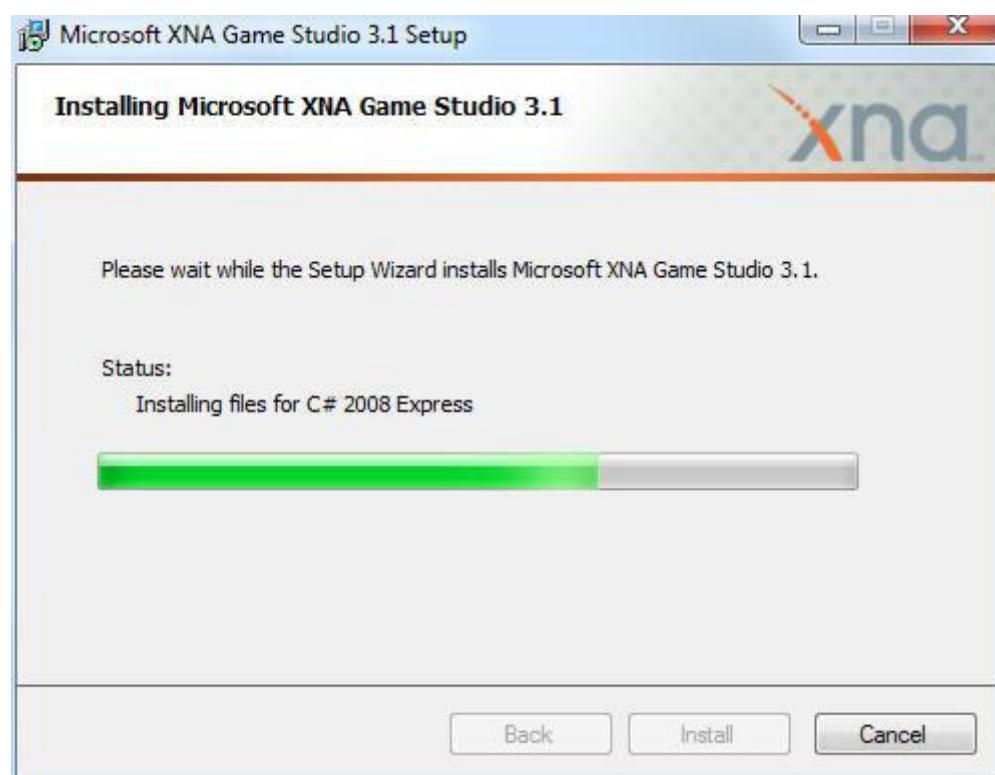


Figura 54: Installing files for C# 2008 Express

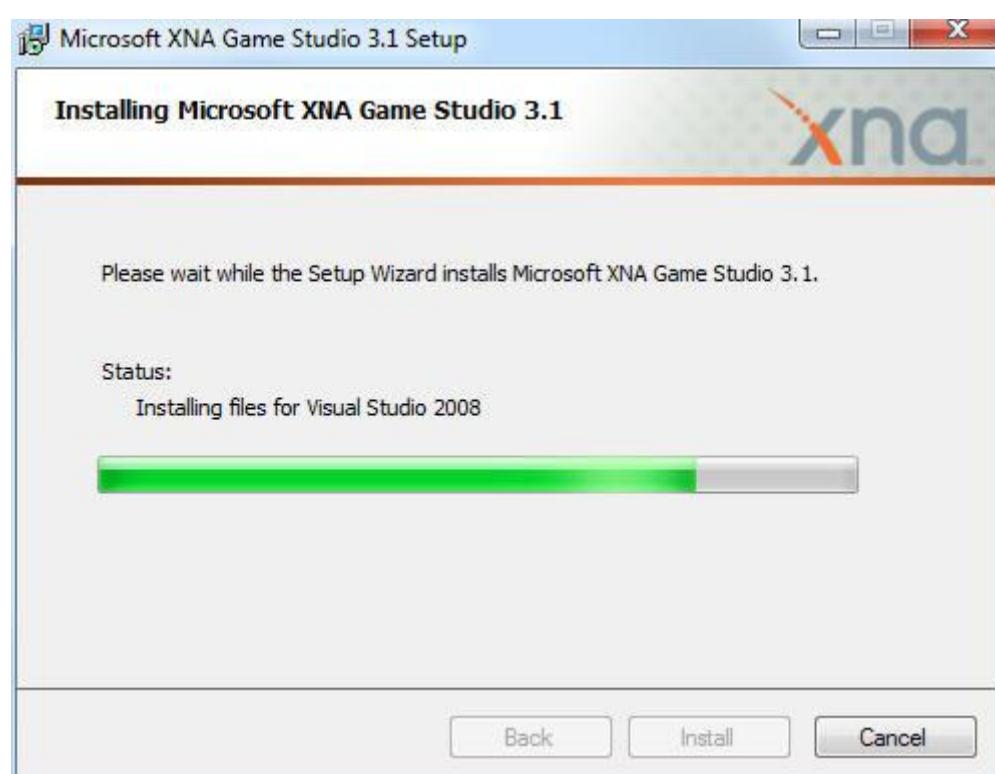


Figura 55: Installing files for Visual Studio 2008

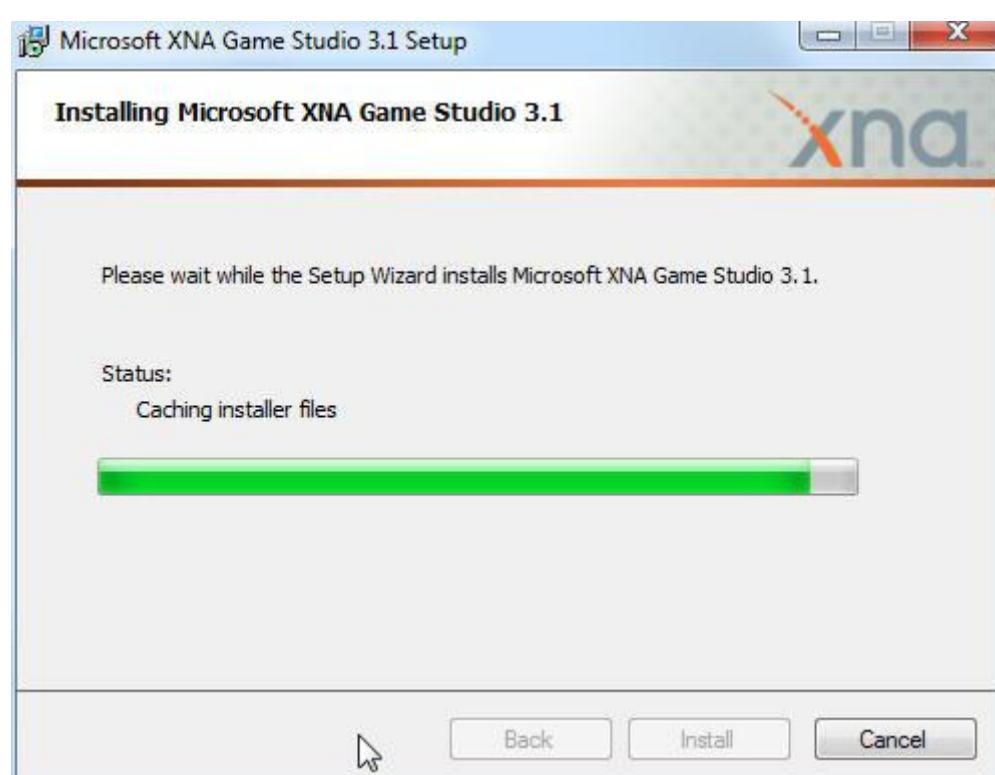


Figura 56: Caching installer files

Lo dirige a: <http://creators.xna.com/es-ES>

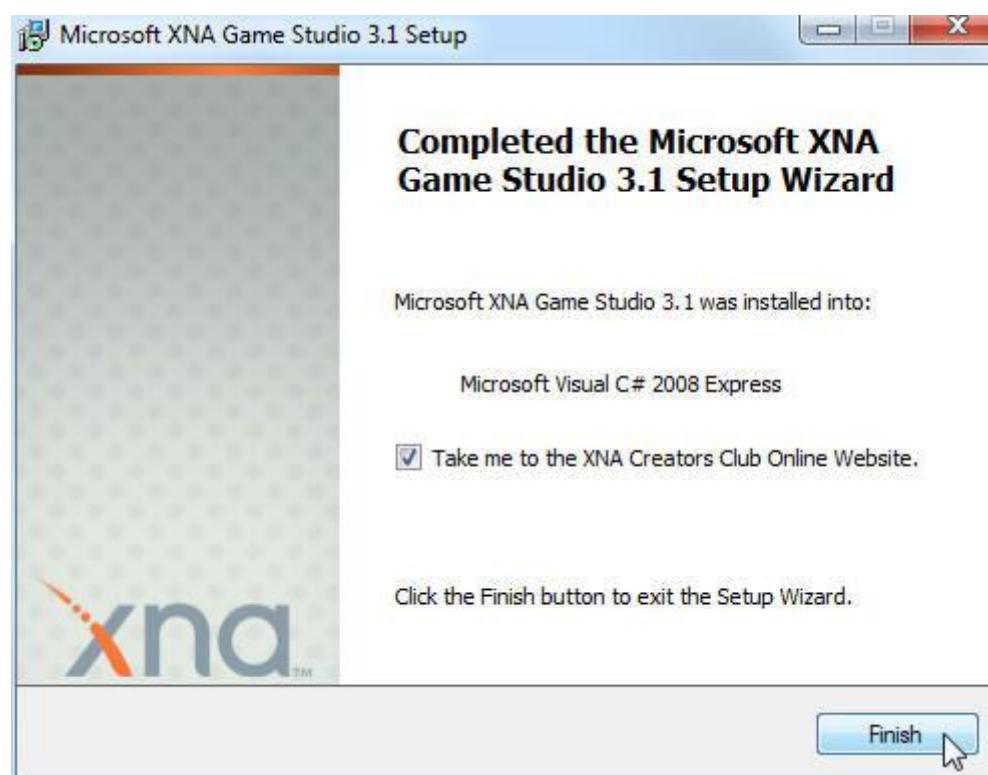


Figura 57: Instalación finalizada

Estos son las entradas que son adicionadas al menú de Inicio



Figura 58: Menú de Windows que muestra que XNA se ha instalado

## 15. XNA: Empezando a desarrollar en XNA

Empieza ejecutando Visual C# 2008 Express Edition:

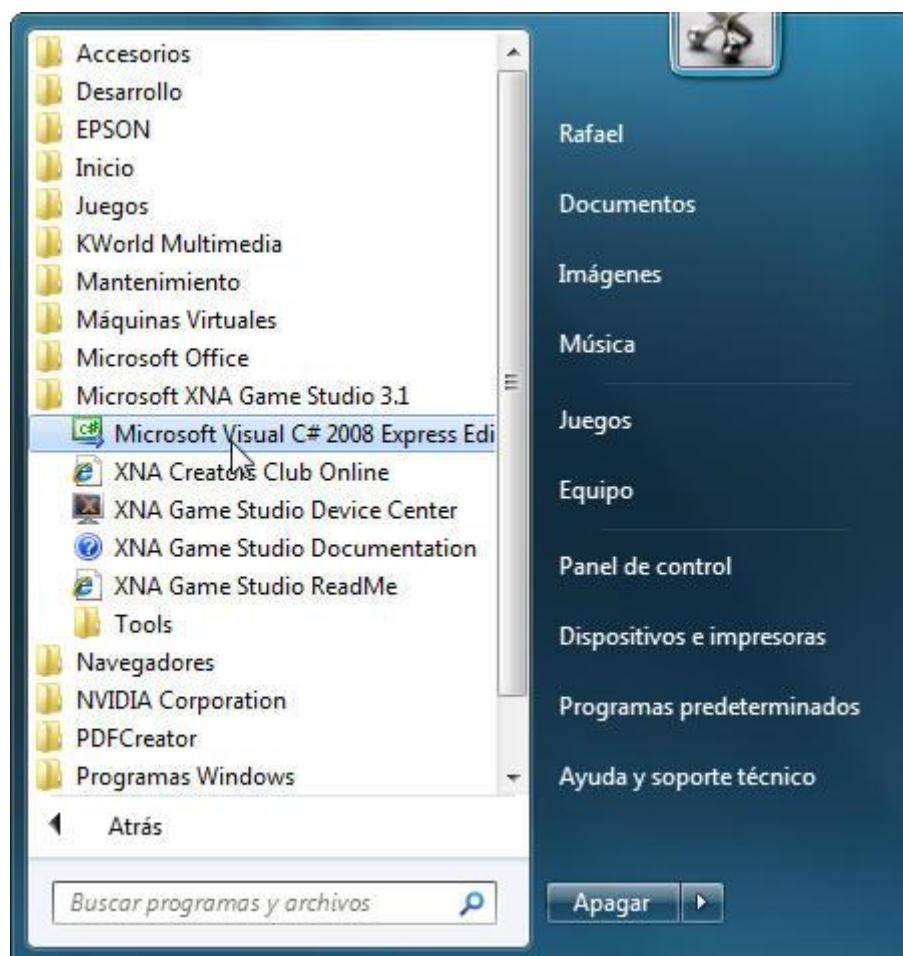


Figura 59: Iniciando Microsoft Visual C# 2008

Se crea un "Nuevo proyecto..."

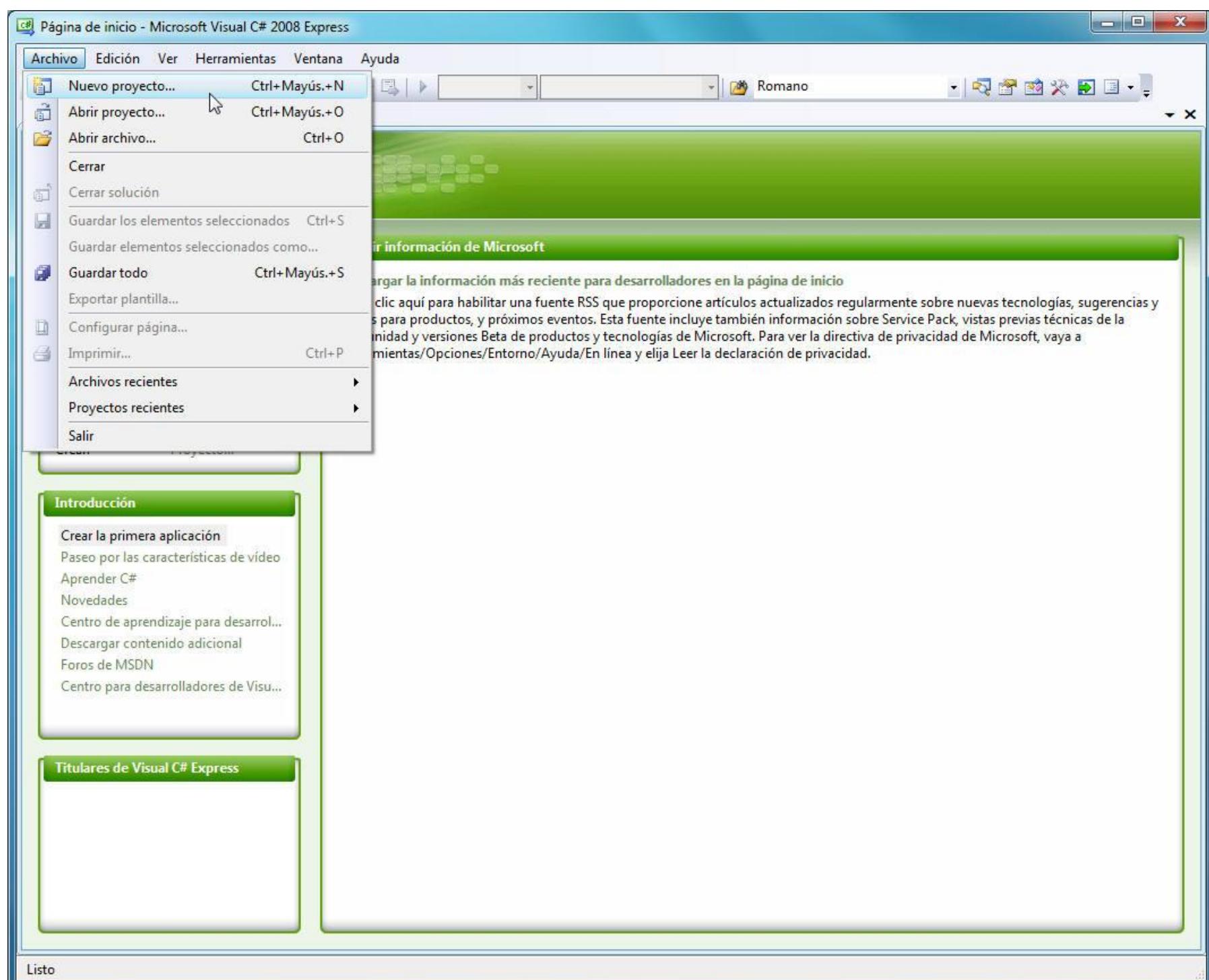


Figura 60: Se crea un nuevo proyecto de Microsoft Visual C# 2008

Seleccione en "XNA Game Studio 3.1", el "Windows Game (3.1)", escriba un nombre a ese proyecto, en el ejemplo se ha puesto "PrimerJuego" Rafael Alberto Moreno Parra. <http://darwin.50webs.com>

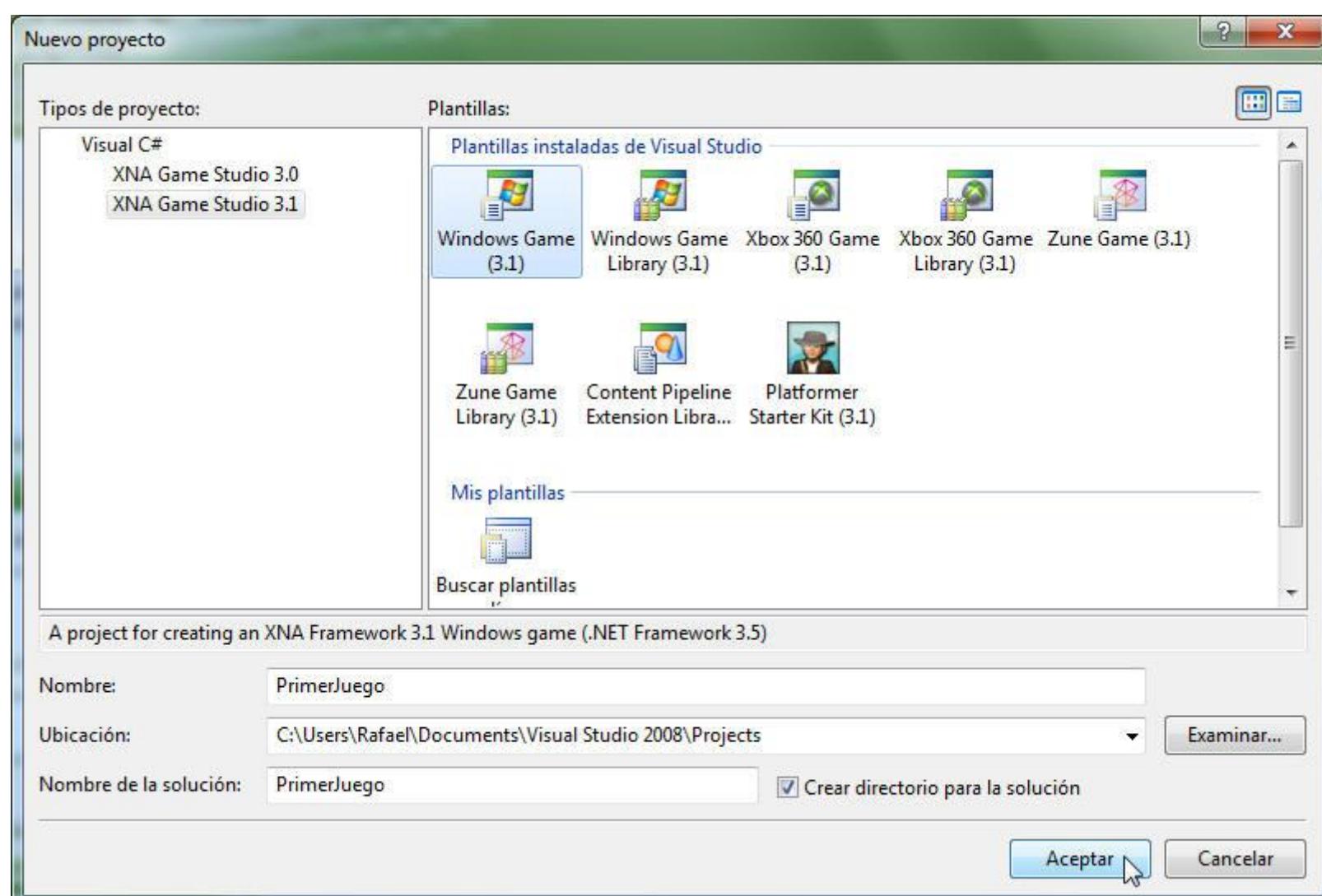


Figura 61: Un proyecto de tipo Windows Game (3.1)

Se genera automáticamente el esqueleto para poder empezar a desarrollar nuestro juego

```

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

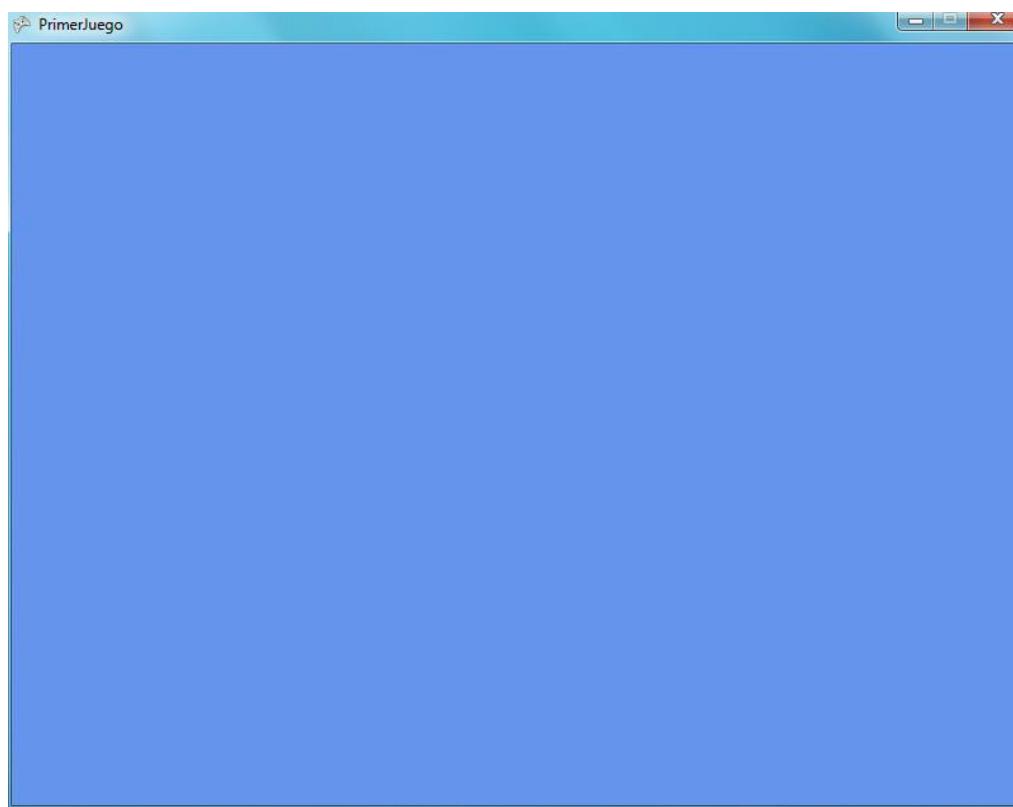
        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here
        }
    }
}

```

Figura 62: Esqueleto de una aplicación en XNA

Incluso puede ejecutar la aplicación, que por supuesto, no muestra nada porque tan sólo se ha creado el ambiente para desarrollar un videojuego.  
Rafael Alberto Moreno Parra. <http://darwin.50webs.com>



**Figura 63: Ejecutando por primera vez una aplicación en XNA**

## 16. XNA: ¿Cómo es una aplicación?

Una aplicación en XNA está escrita en el lenguaje C# y tiene la siguiente estructura:

Una serie de librerías propias, como se ven a continuación

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
```

El namespace propio de cualquier aplicación en Visual C#

```
namespace WindowsGame1
{
    .....
}
```

La clase principal que controla todo el juego. Esta hereda de la clase Microsoft.Xna.Framework.Game

```
public class Game1 : Microsoft.Xna.Framework.Game
{
    ...
}
```

Dentro de esa clase tenemos los siguientes métodos:

El constructor que viene por defecto. En este libro no modificaremos el constructor.

```
public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";
}
```

El método Initialize(), que tampoco será modificado en este libro. Lo dejaremos como está.

```
/// <summary>
/// Allows the game to perform any initialization it needs to before starting to run.
/// This is where it can query for any required services and load any non-graphic
/// related content. Calling base.Initialize will enumerate through any components
/// and initialize them as well.
/// </summary>
protected override void Initialize()
{
    // TODO: Add your initialization logic here

    base.Initialize();
}
```

Vienen luego, una serie de métodos que si serán usados activamente en este libro, como lo es LoadContent(). Este método es llamado al inicio, allí habrán una serie de instrucciones que sirven para cargar en memoria todas las imágenes (texturas, fondos) y sonidos que tendrá el juego.

```
/// <summary>
/// LoadContent will be called once per game and is the place to load
/// all of your content.
/// </summary>
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here
}
```

El método UnloadContent() no será modificado en este libro, lo dejamos como está.

```
/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}
```

El método Update(GameTime gameTime) es en el que reposará toda la lógica del juego (por ejemplo, el tratamiento de las colisiones) y también el que leerá los dispositivos de entrada (por ejemplo, teclado, mouse y gamepad) y de tocar los sonidos.

```
/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // TODO: Add your update logic here

    base.Update(gameTime);
}
```

Finalmente el método Draw(GameTime gameTime), el cual se encargará de pintar todo: los personajes y el fondo.

```
/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here

    base.Draw(gameTime);
}
```

Principalmente nos concentraremos en tres métodos: LoadContent(), Update(GameTime gameTime) y Draw(GameTime gameTime)

## 17. XNA: Agregando un fondo de pantalla estático al juego

En nuestro proyecto de juego en C#

```

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

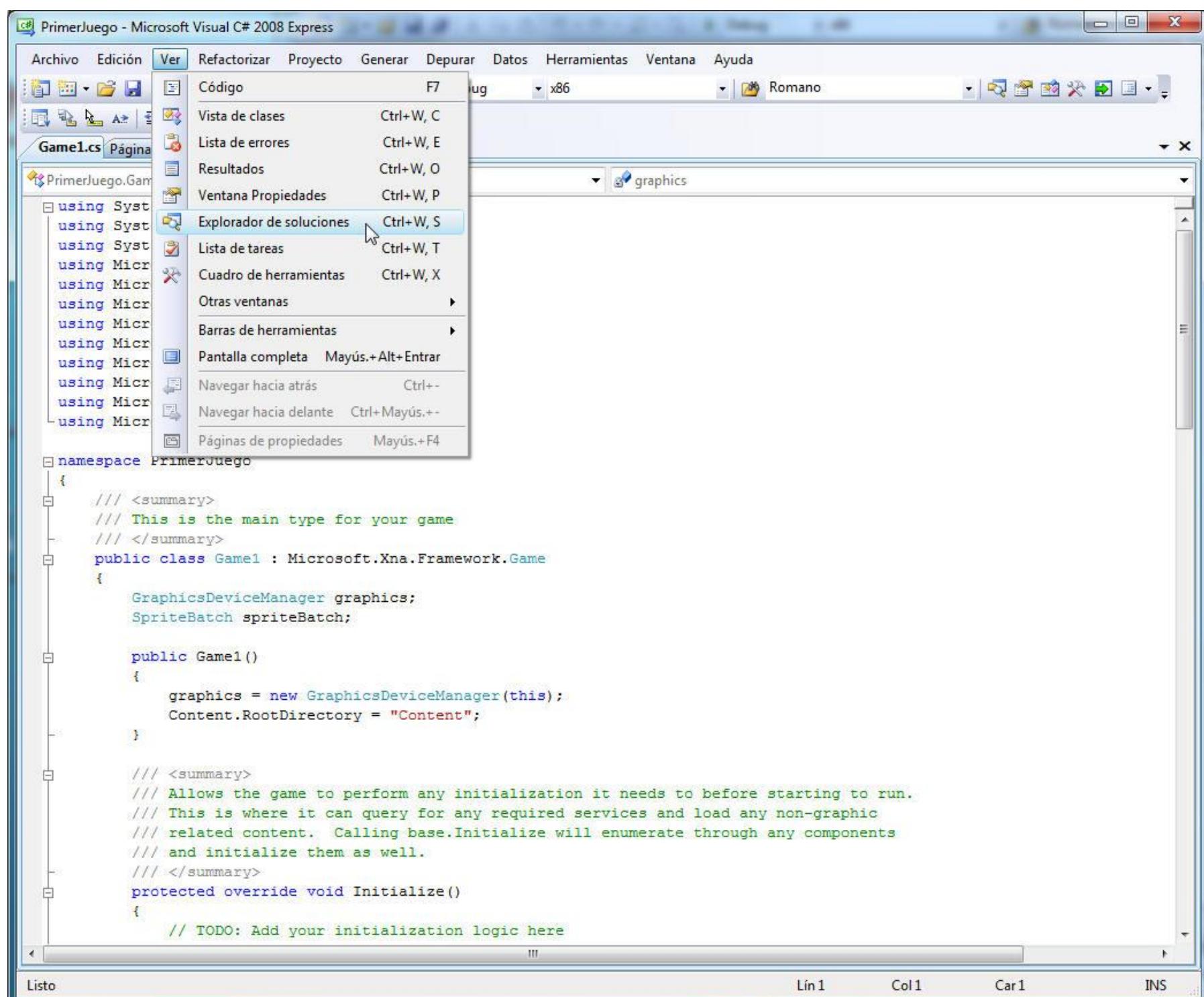
        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here
        }
    }
}

```

Figura 64: Esqueleto de una aplicación en XNA usando Visual C#

Damos clic en Explorador de Soluciones

**Figura 65: Explorador de soluciones**

De allí vamos al ítem "Content"

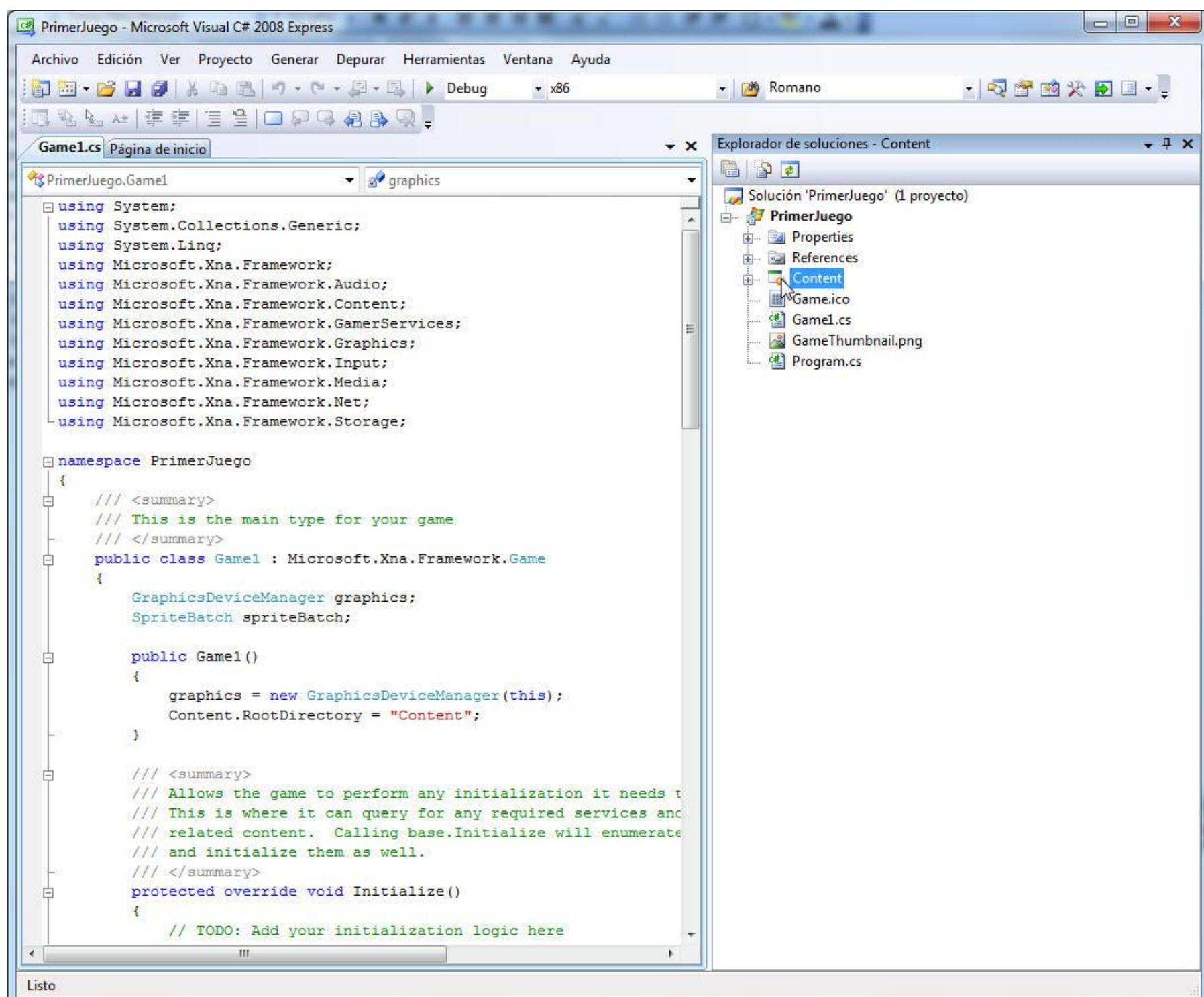


Figura 66: En "Content" se administran los recursos usados por el videojuego

Clic botón derecho sobre "Content" y seleccionamos "Aregar -> Nueva Carpeta"

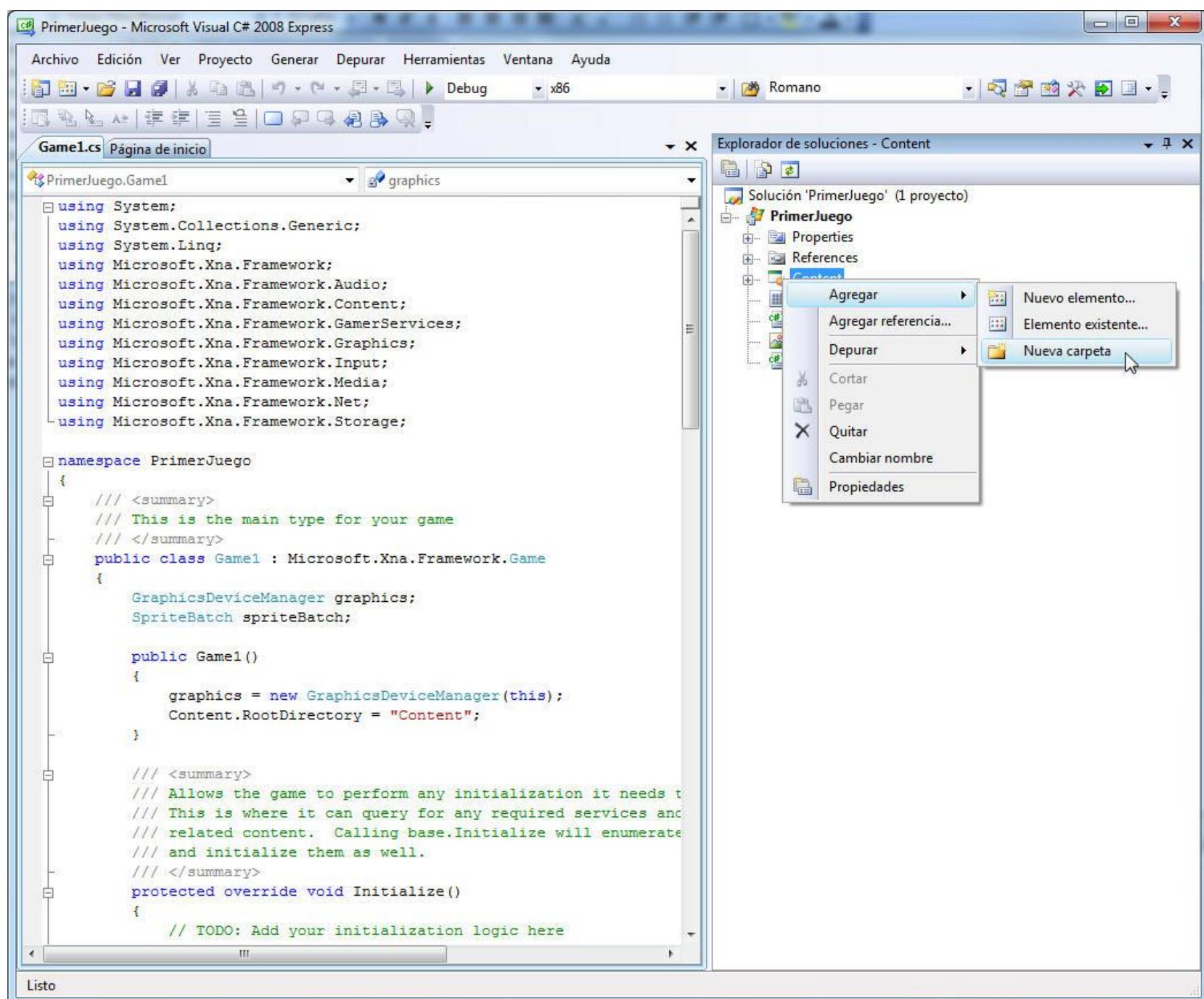
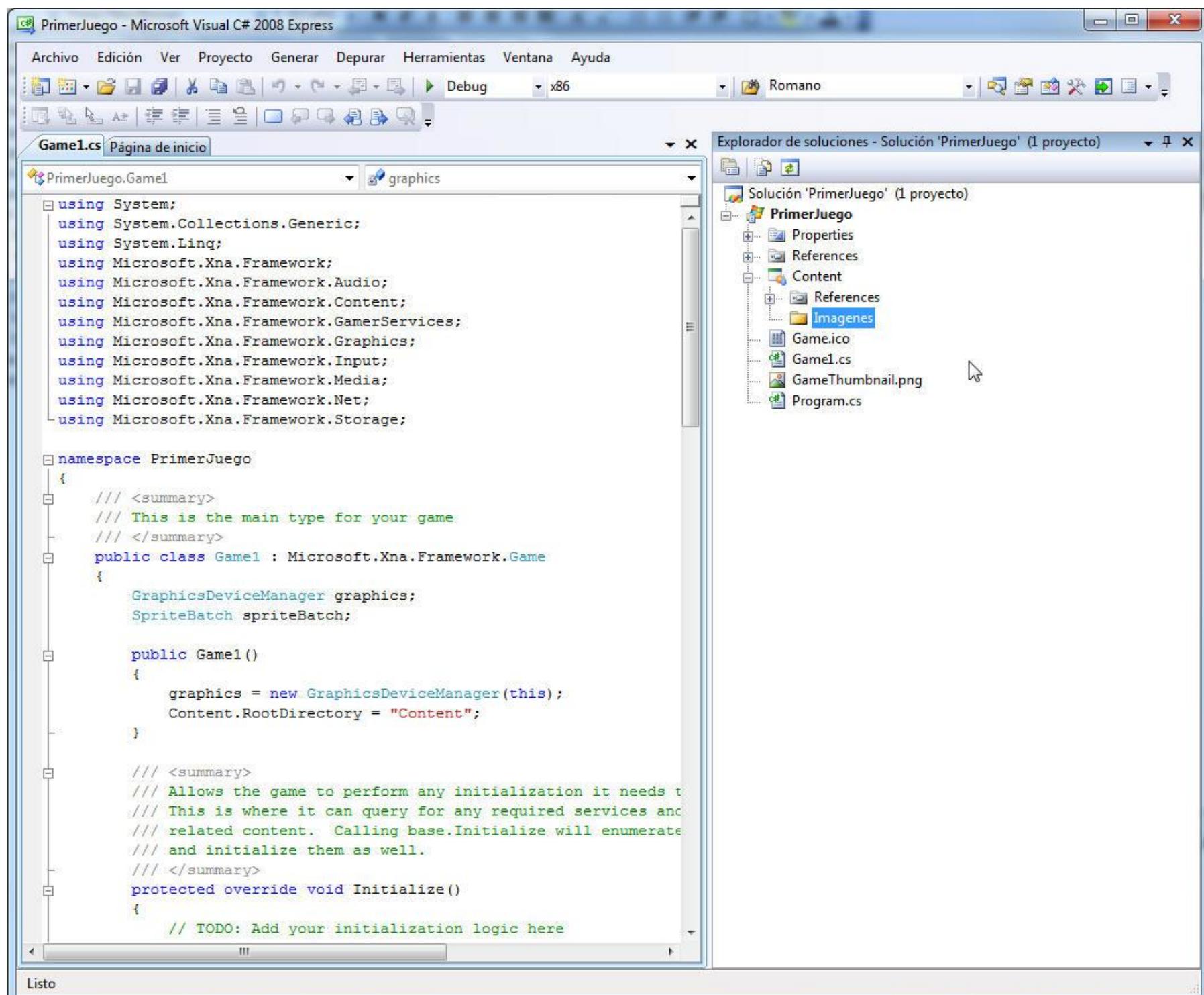


Figura 67: Organizando los recursos con carpetas

Le damos un nombre a esa carpeta, por ejemplo, "Imagenes" (sin tilde)



**Figura 68: Carpeta donde estará los recursos de imagen**

Ahora es tener una imagen que sirva de fondo a nuestro juego



**Figura 69: La imagen puede ser .JPG**

Arrastramos desde donde está el archivo de imagen a la carpeta "Imagenes" del proyecto. En este caso Visual C# hace una copia del archivo y la deja en la carpeta del juego.

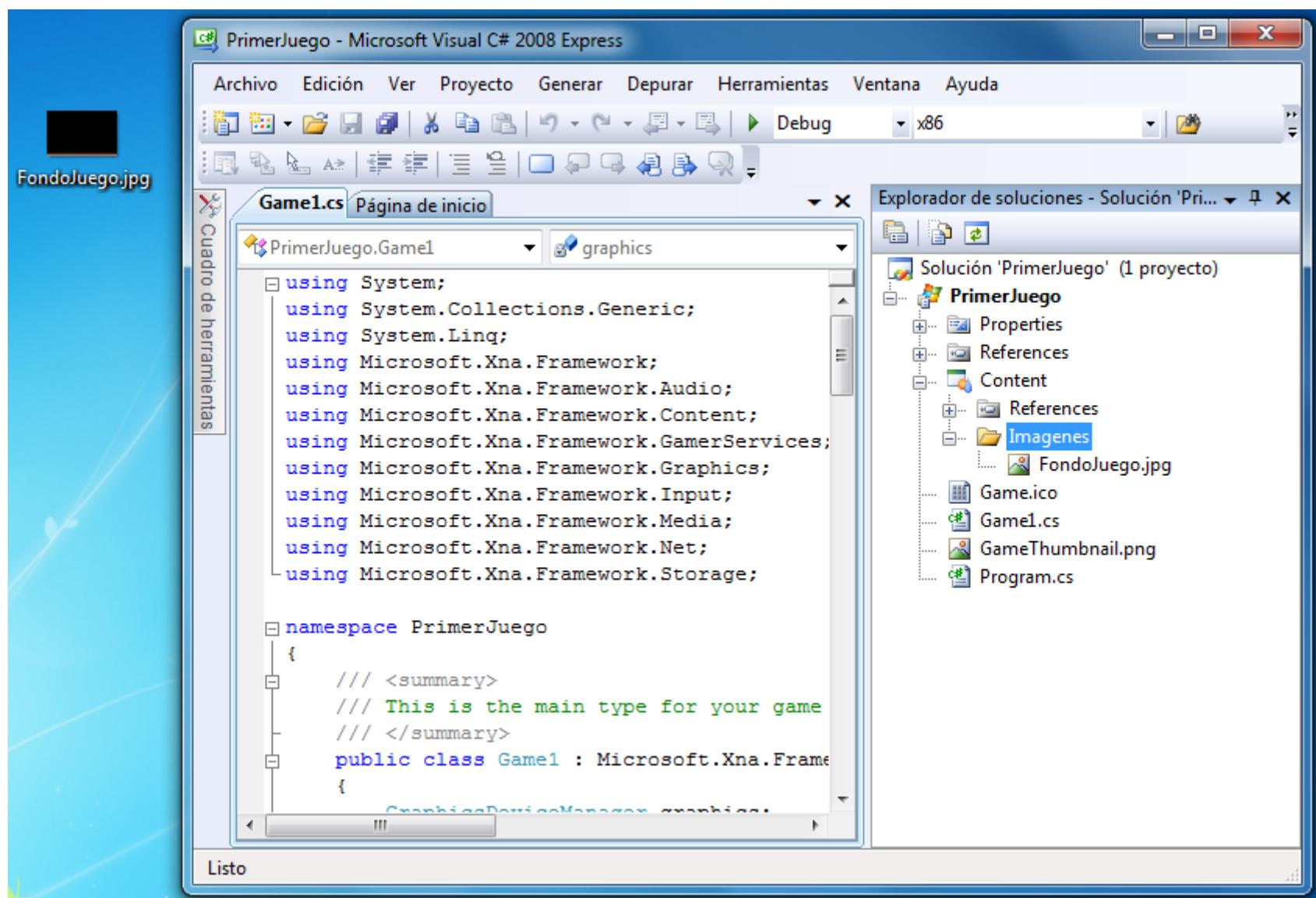
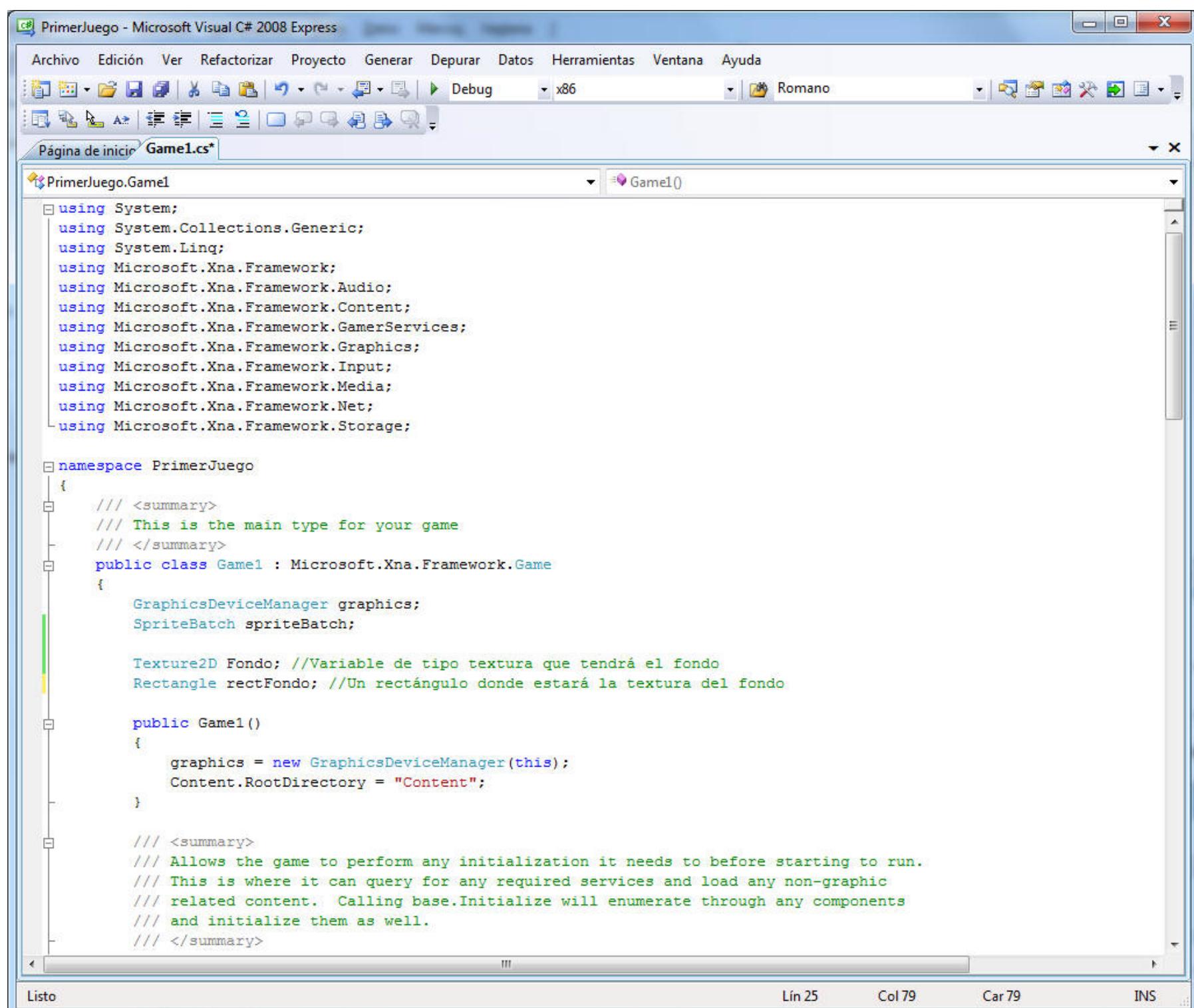


Figura 70: Adicionando la imagen a los recursos del videojuego

Ahora agregamos estas dos instrucciones al código



**Figura 71: Agregando variables de clase para manejar el fondo**

Son variables de clase

```

Texture2D Fondo; //Variable de tipo textura que tendrá el fondo

Rectangle rectFondo; //Un rectángulo donde estará la textura del fondo

```

Luego agregamos estas instrucciones en el método LoadContent() justo después de spriteBatch. LoadContent() es el método en el cual cargaremos todas las imágenes en sus variables respectivas.

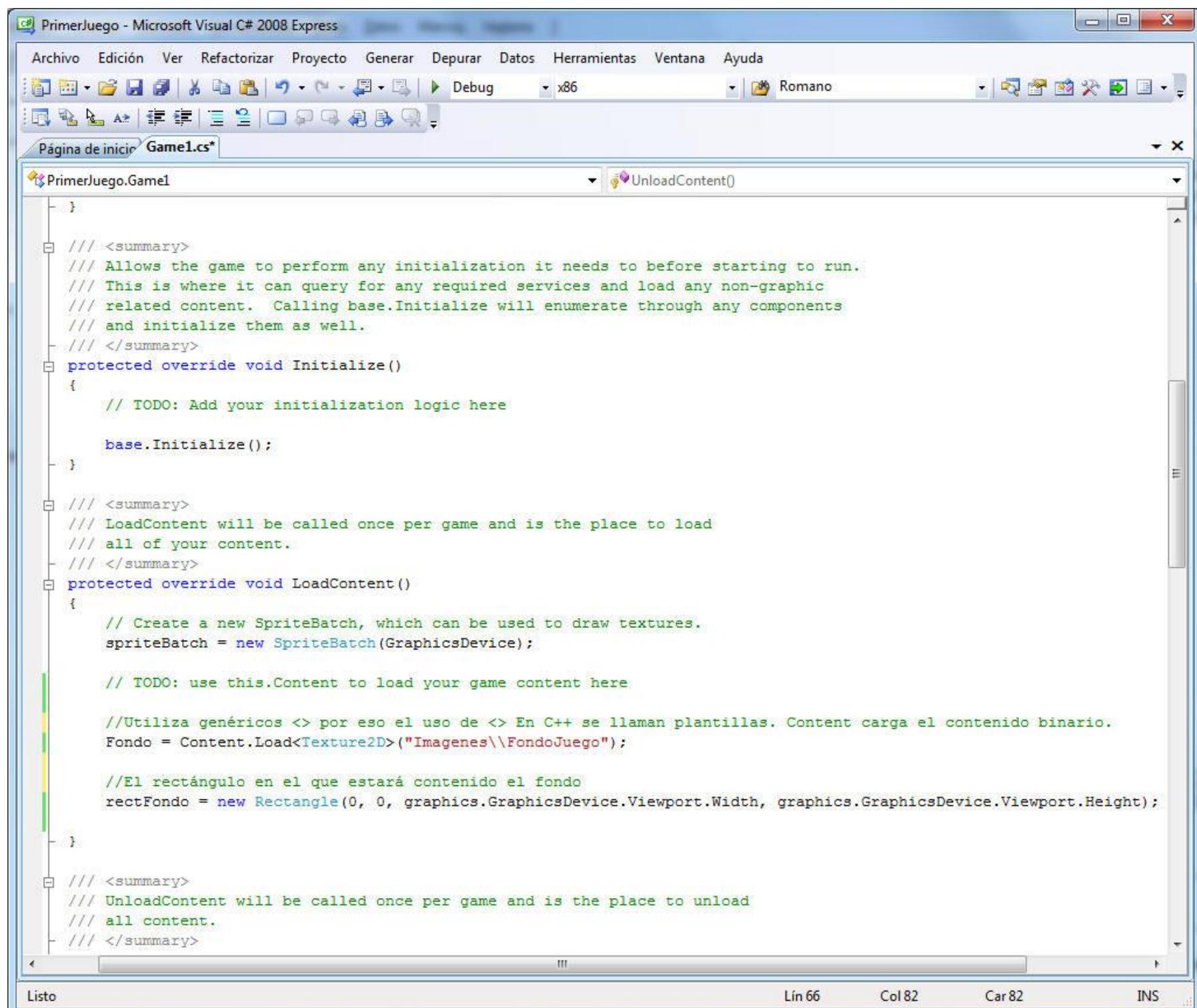


Figura 72: Cargando el recurso en LoadContent()

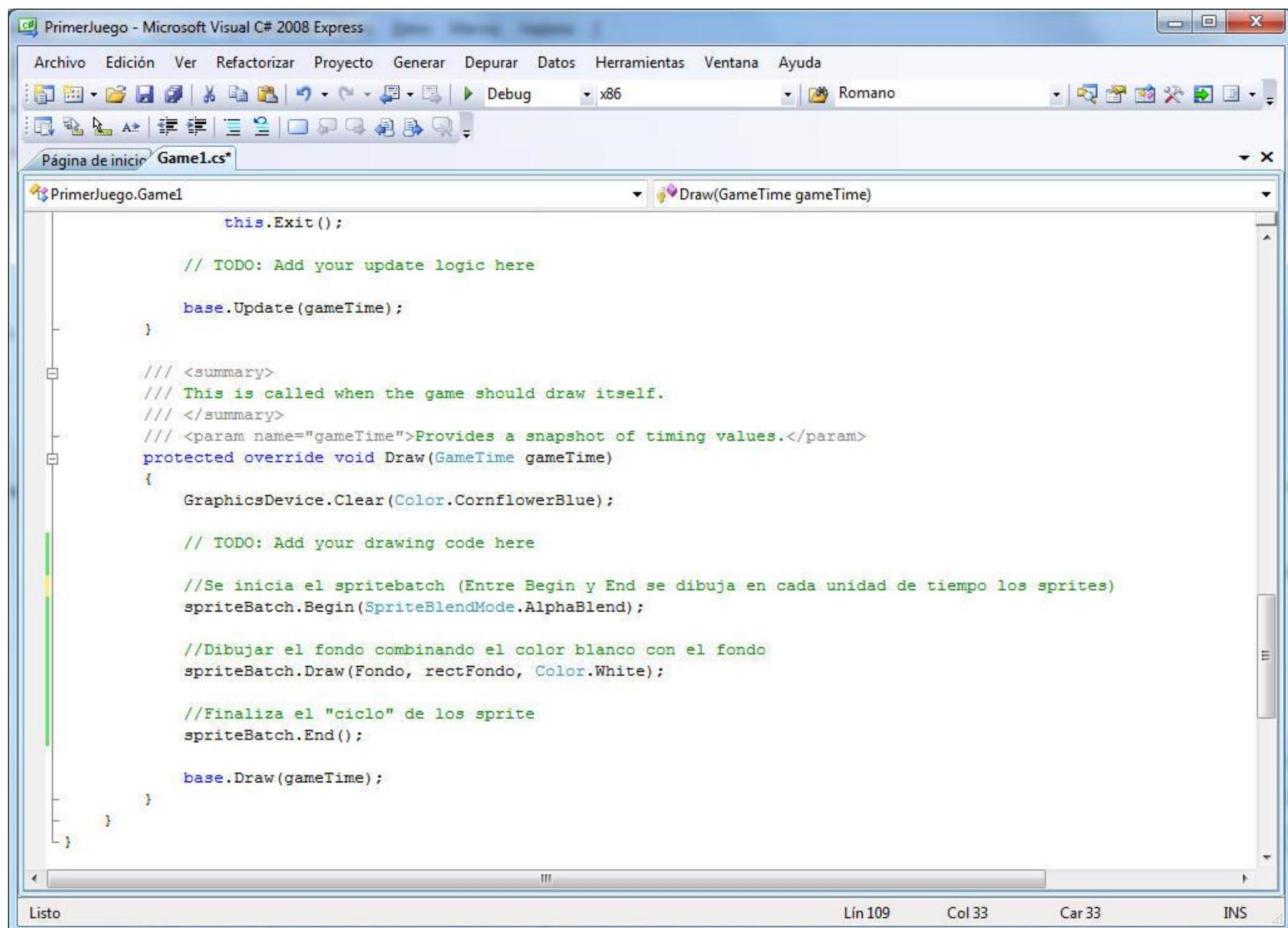
```

//Utiliza genéricos <> por eso el uso de <> En C++ se llaman plantillas. Content carga el contenido binario.
Fondo = Content.Load<Texture2D>("Imagenes\\FondoJuego");

//El rectángulo en el que estará contenido el fondo
rectFondo = new Rectangle(0, 0, graphics.GraphicsDevice.Viewport.Width, graphics.GraphicsDevice.Viewport.Height);

```

Observamos que FondoJuego (que es la imagen) no requiere adicionarle la extensión .JPG  
 Ya preparadas las variables con los datos, es momento de mostrarlas, para eso nos dirigimos al método Draw(), allí debemos escribir las siguientes instrucciones:



**Figura 73: Cambiando el método Draw para mostrar el fondo**

Estas son las instrucciones que debemos adicionar.

```

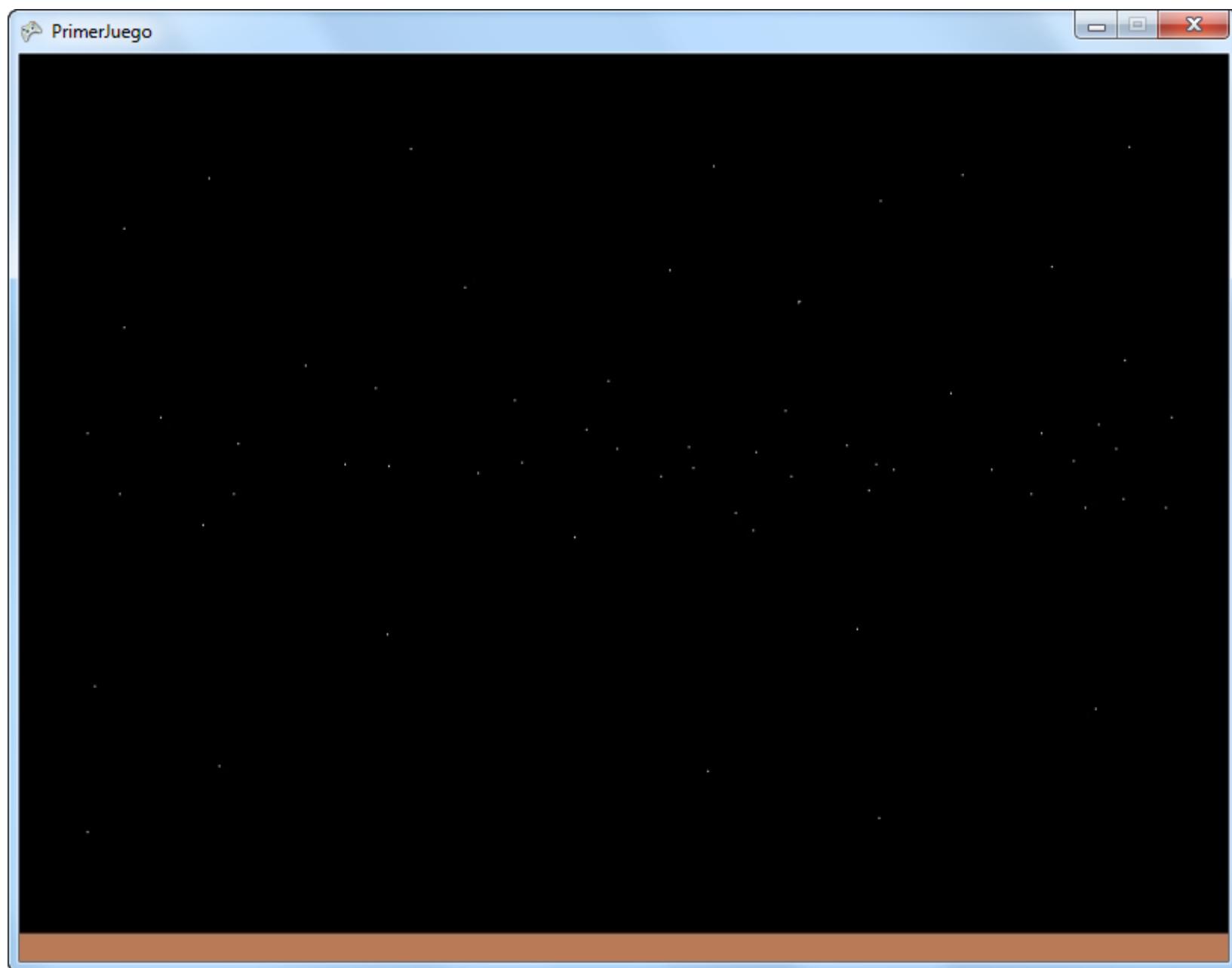
//Se inicia el spritebatch (Entre Begin y End se dibuja en cada unidad de tiempo los sprites)
spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

//Dibujar el fondo combinando el color blanco con el fondo
spriteBatch.Draw(Fondo, rectFondo, Color.White);

//Finaliza el "ciclo" de los sprite
spriteBatch.End();

```

Los "sprite" son el término técnico para referirse a todo lo que se ve en la ventana del juego (protagonistas, enemigos, fondo). Ahora ya podemos ejecutar el proyecto y ver que se carga el fondo.



**Figura 74: Ejecutando el programa en XNA mostrando el fondo**

Nota: Al pasar el mouse al interior de la ventana, el cursor desaparece, ese es un comportamiento por defecto.

## 18. XNA: Manejo de imágenes con Paint .NET

El manejo de las imágenes para el juego requiere este programa llamado Paint .NET (que es gratuito) que lo podemos descargar de: <http://www.getpaint.net/index.html>

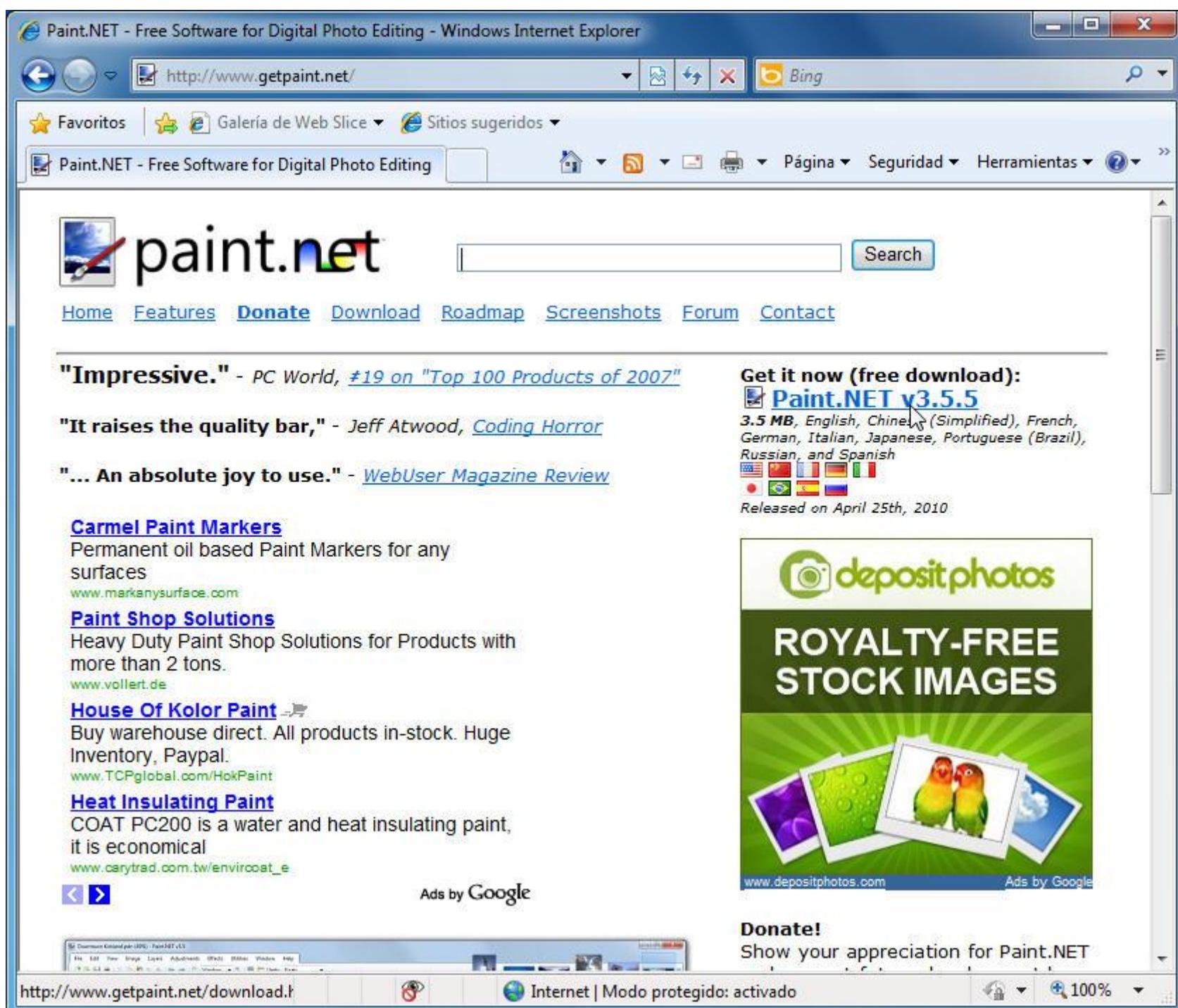


Figura 75: Sitio web oficial de paint .net

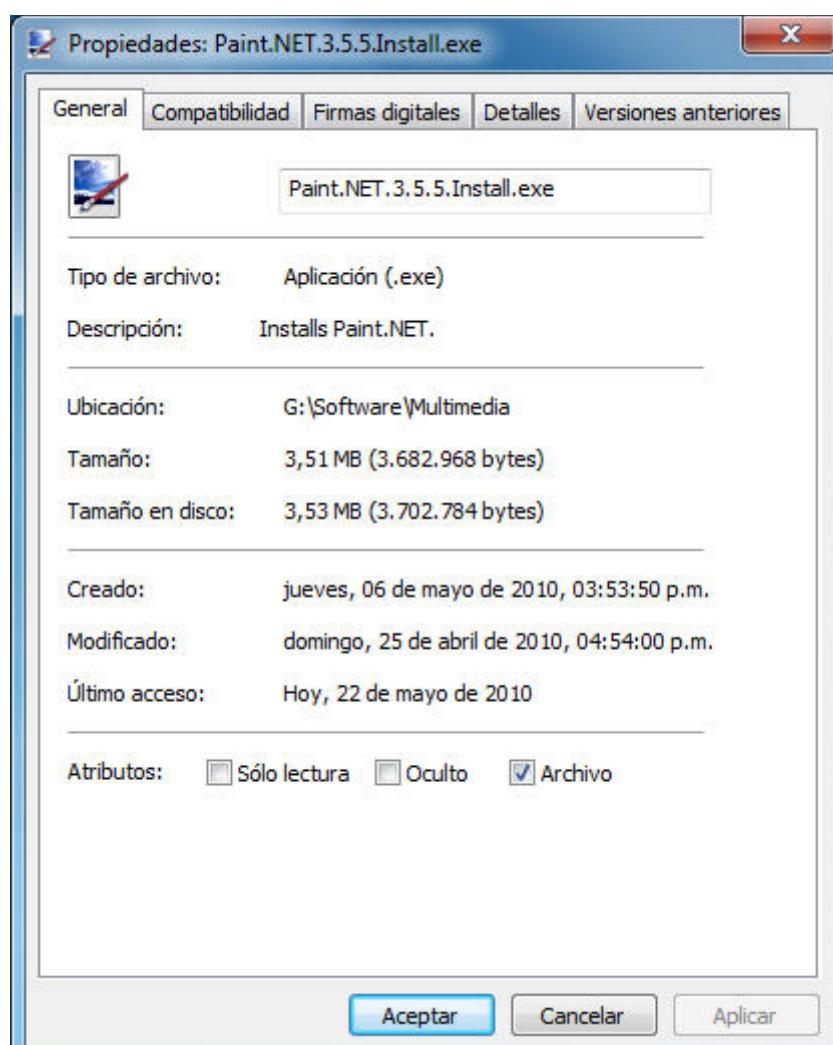


Figura 76: Características del instalador de Paint .NET

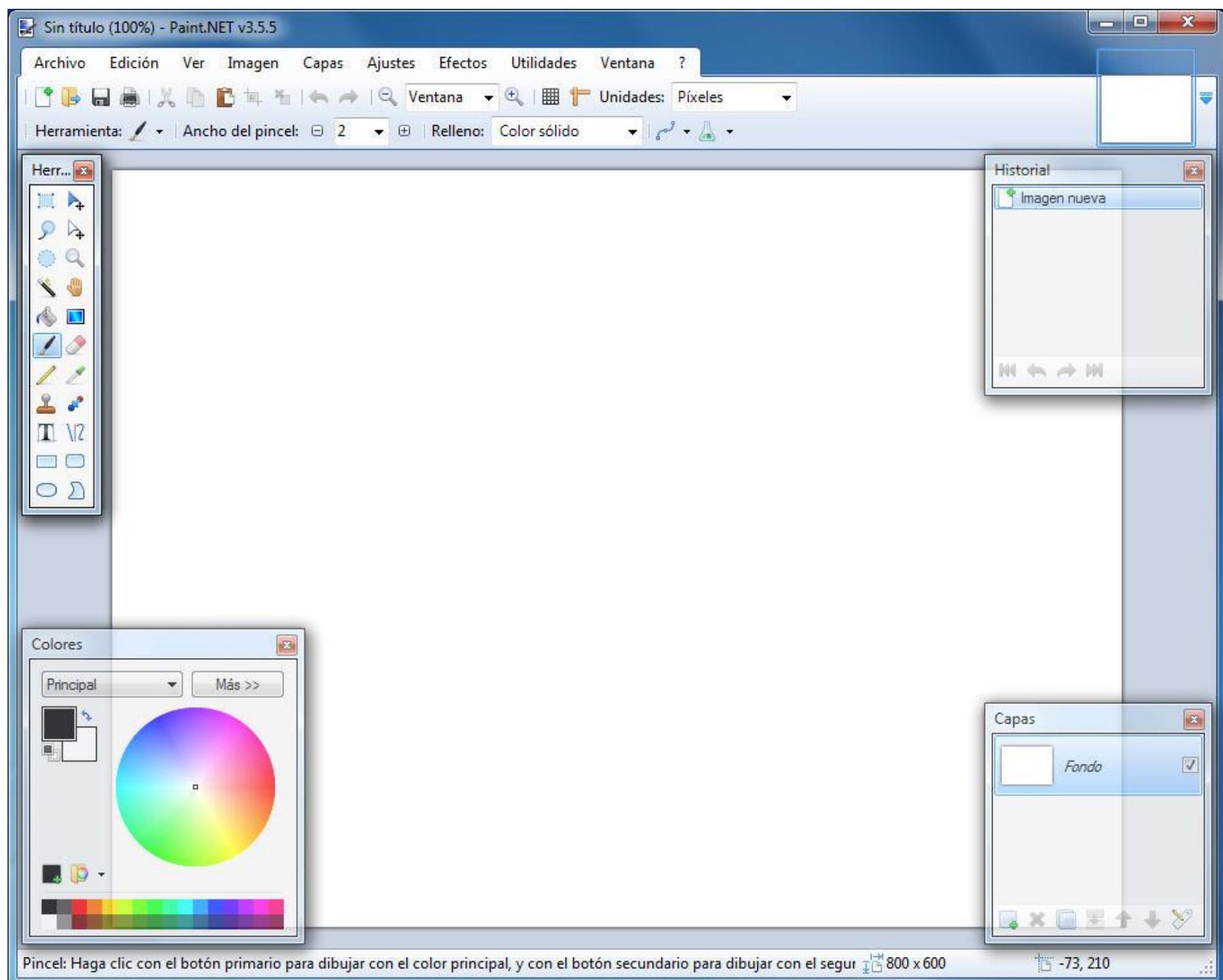


Figura 77: Área de trabajo de Paint.Net

### Usando Paint .NET para las figuras del juego

Pegamos la imagen que hemos seleccionado de alguna fuente (por ejemplo esta que fue dibujada usando Paint).

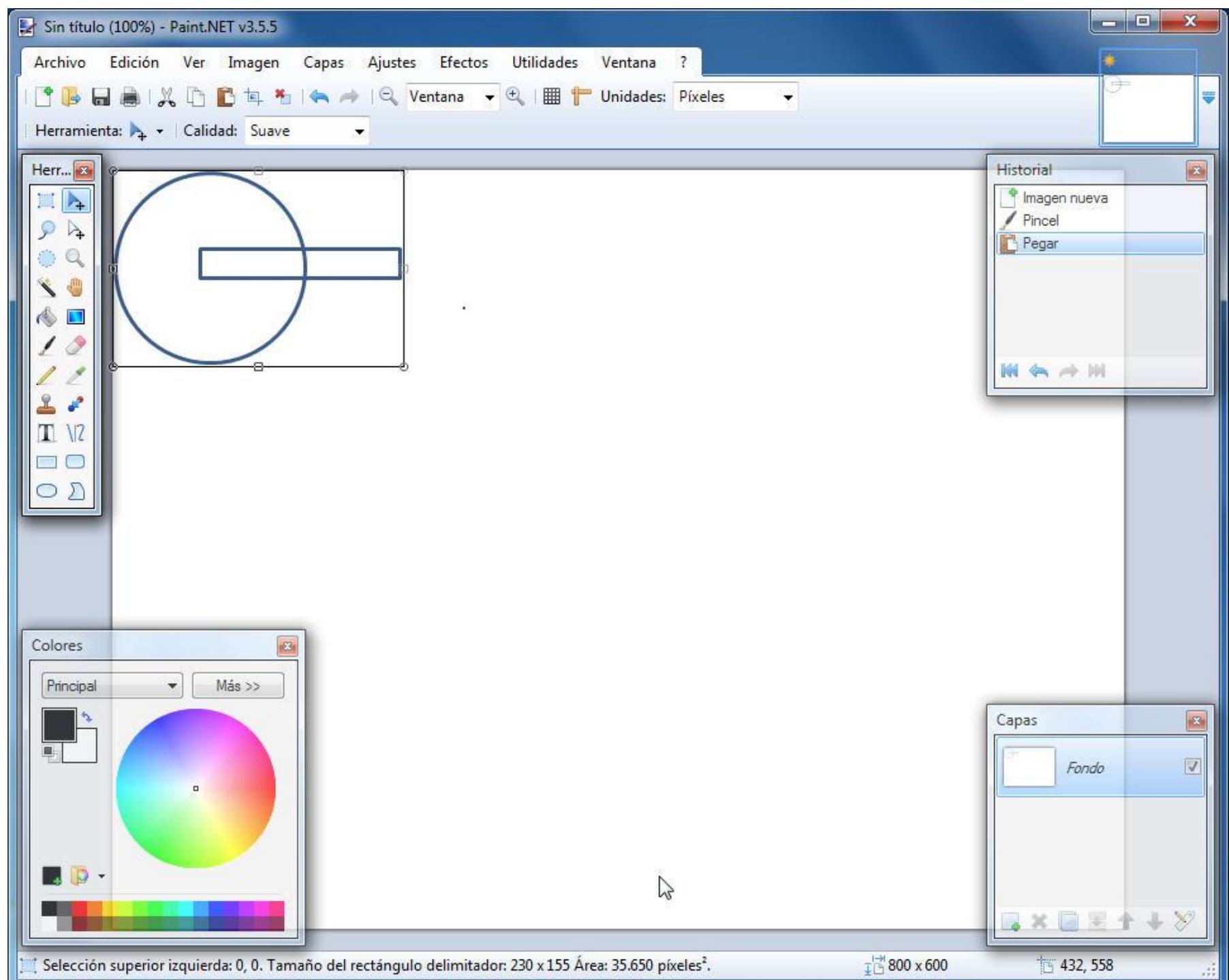
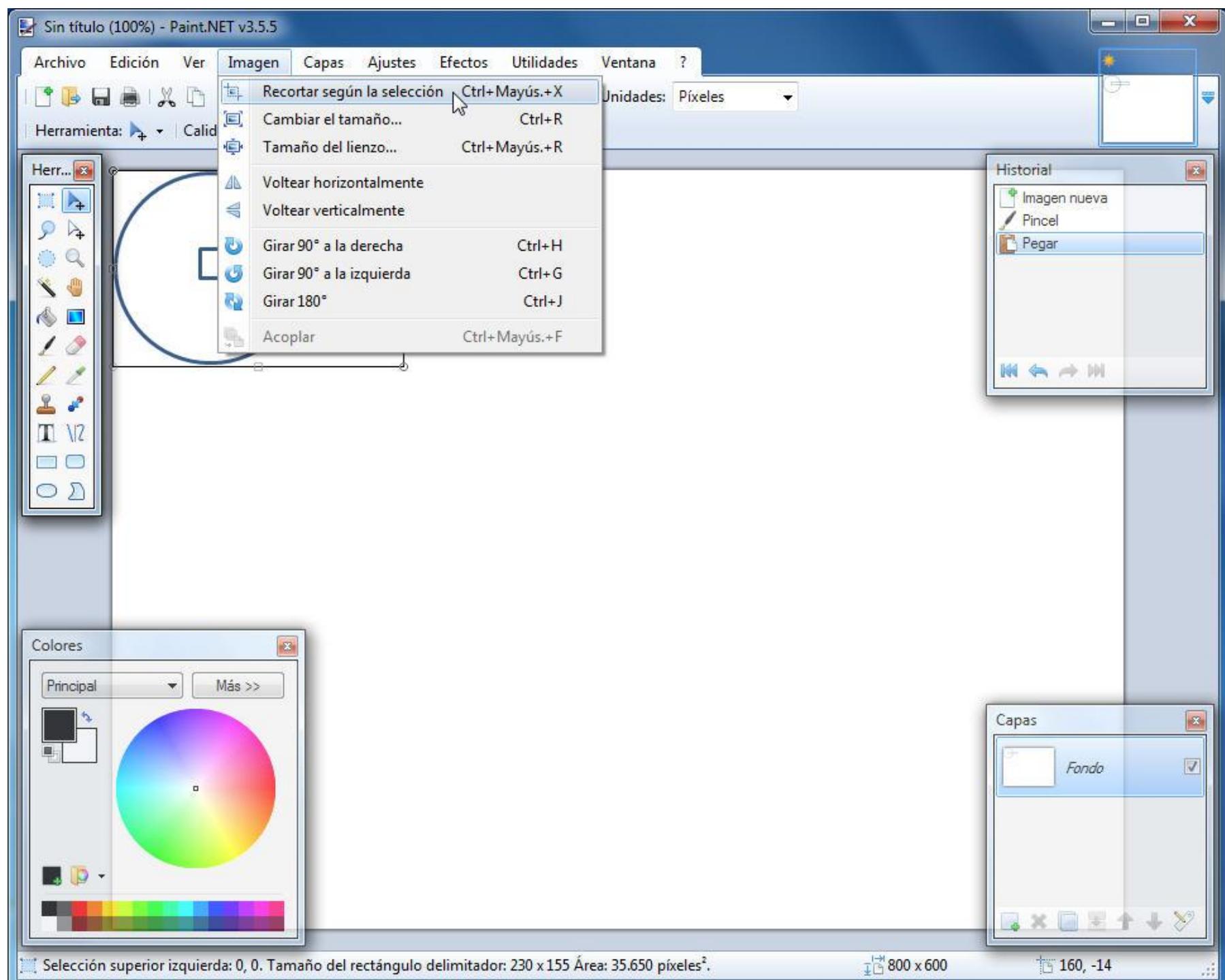


Figura 78: Pega una imagen en Paint.NET

Esa es la imagen, ahora hay que hacer que el lienzo coincida con la imagen, damos clic en "Recortar según la selección"



**Figura 79: Recortar según la selección**

Ya tenemos que el lienzo coincida con la imagen. Ahora el siguiente paso es quitar ese fondo blanco de la imagen. La razón es que esta imagen debe sobreponerse con el fondo pero solo la parte del cañón.

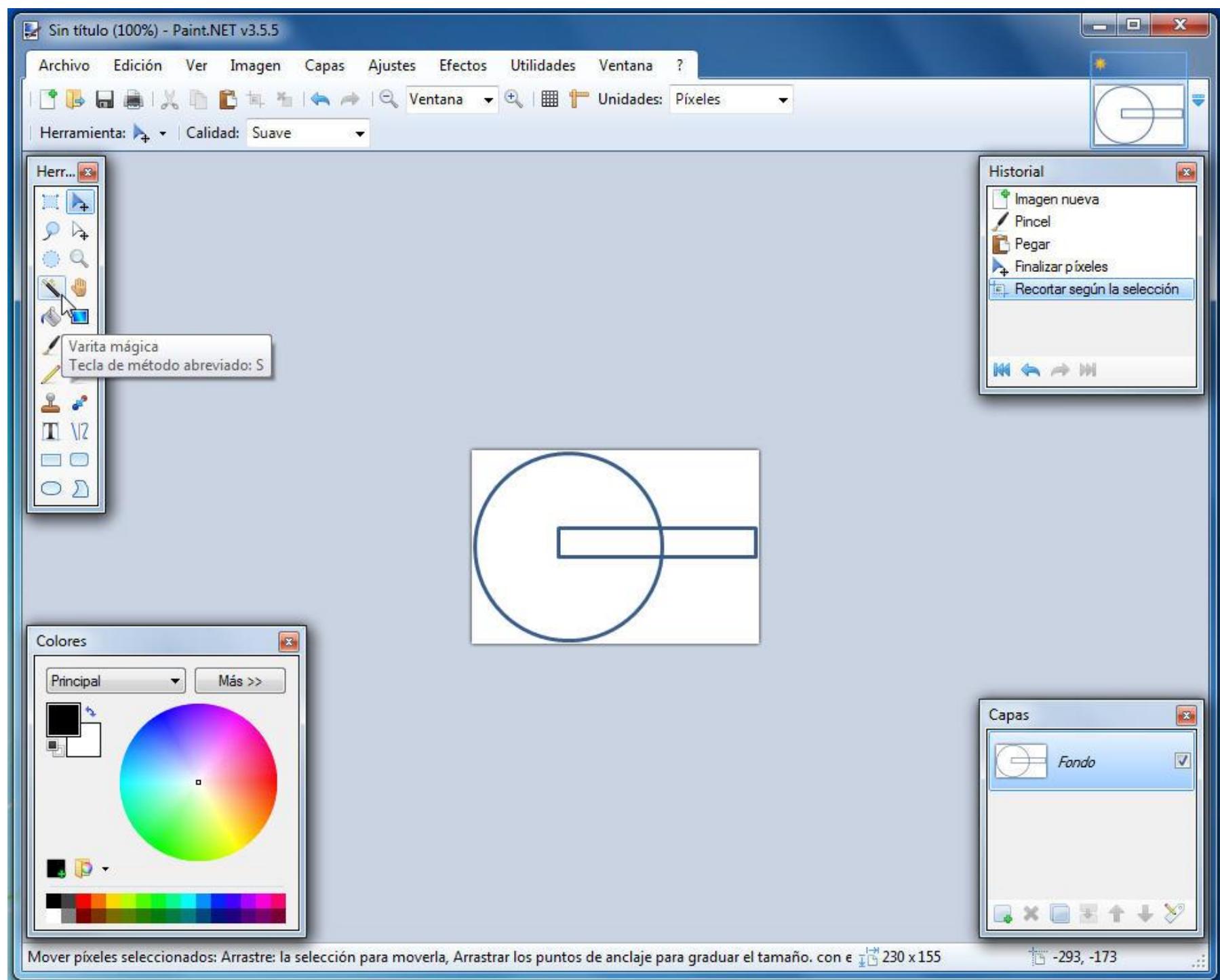


Figura 80: El tamaño de la imagen una vez recortada

Usamos la varita mágica de Paint .NET y damos clic sobre el área a borrar (el área a borrar se marca), luego oprimimos la tecla de borrado del teclado, y se observa lo siguiente:

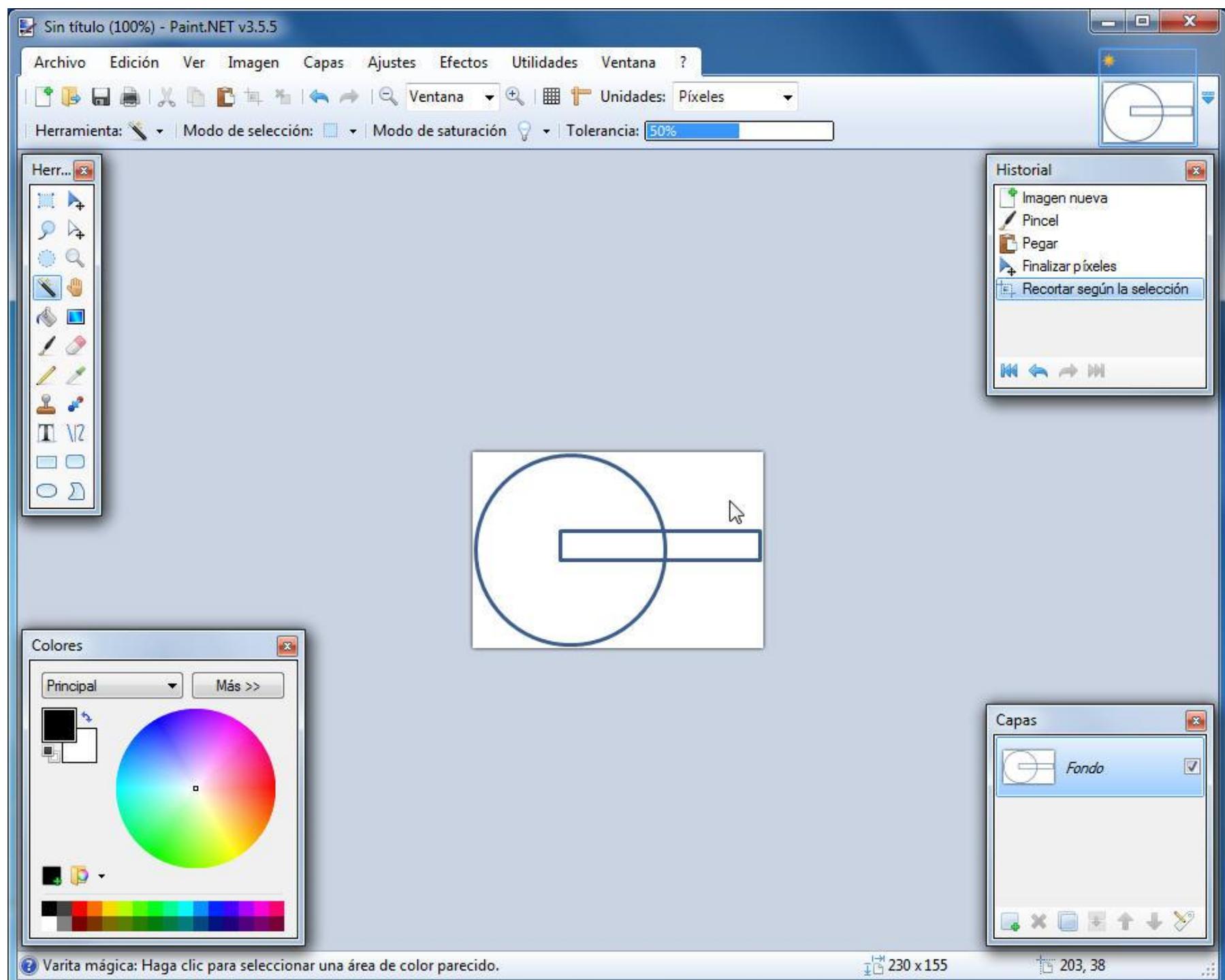


Figura 81: Uso de la varita mágica para hacer transparencias

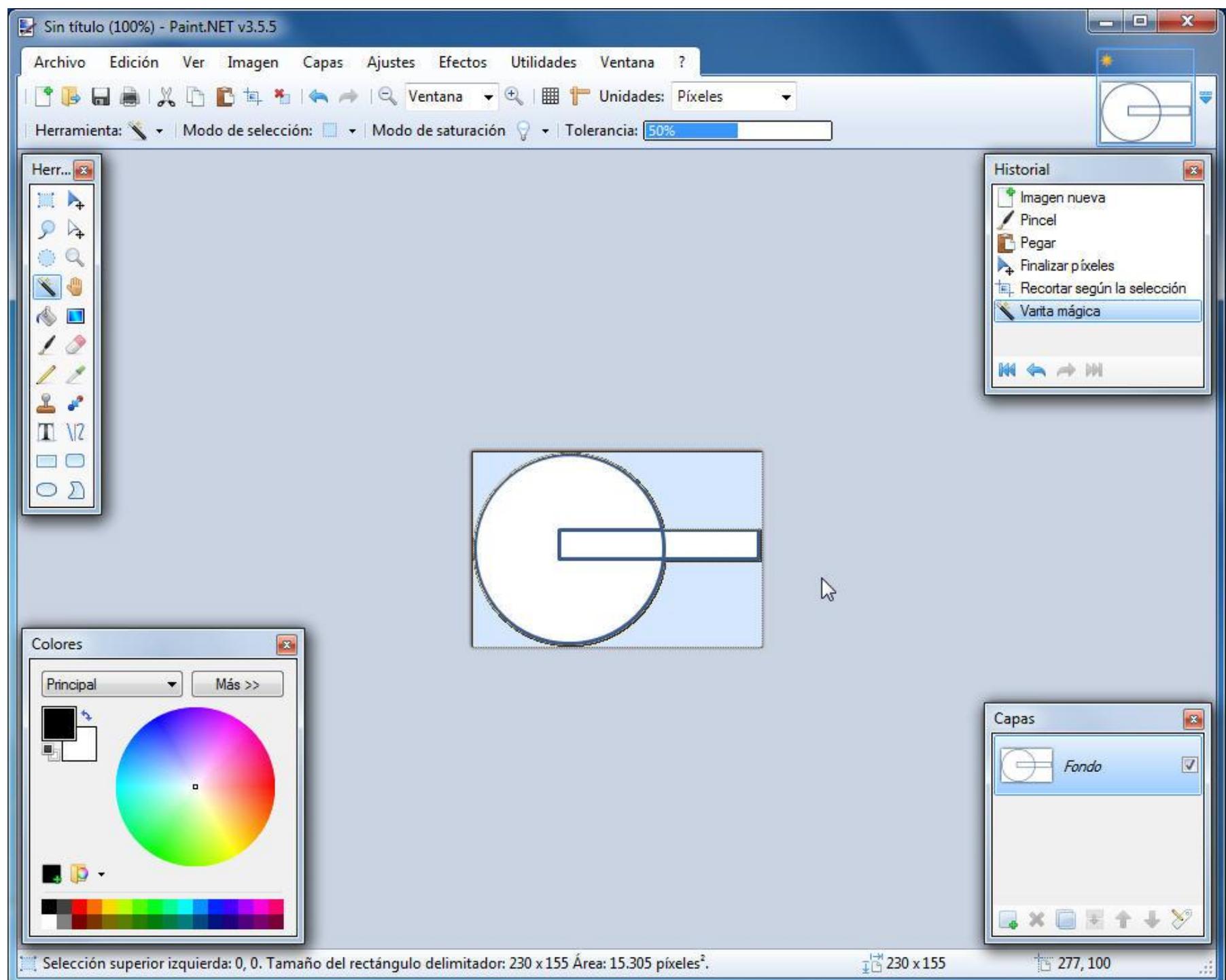


Figura 82: Área seleccionada para hacerla transparente

Ya tenemos nuestra imagen sin fondo blanco o fondo transparente

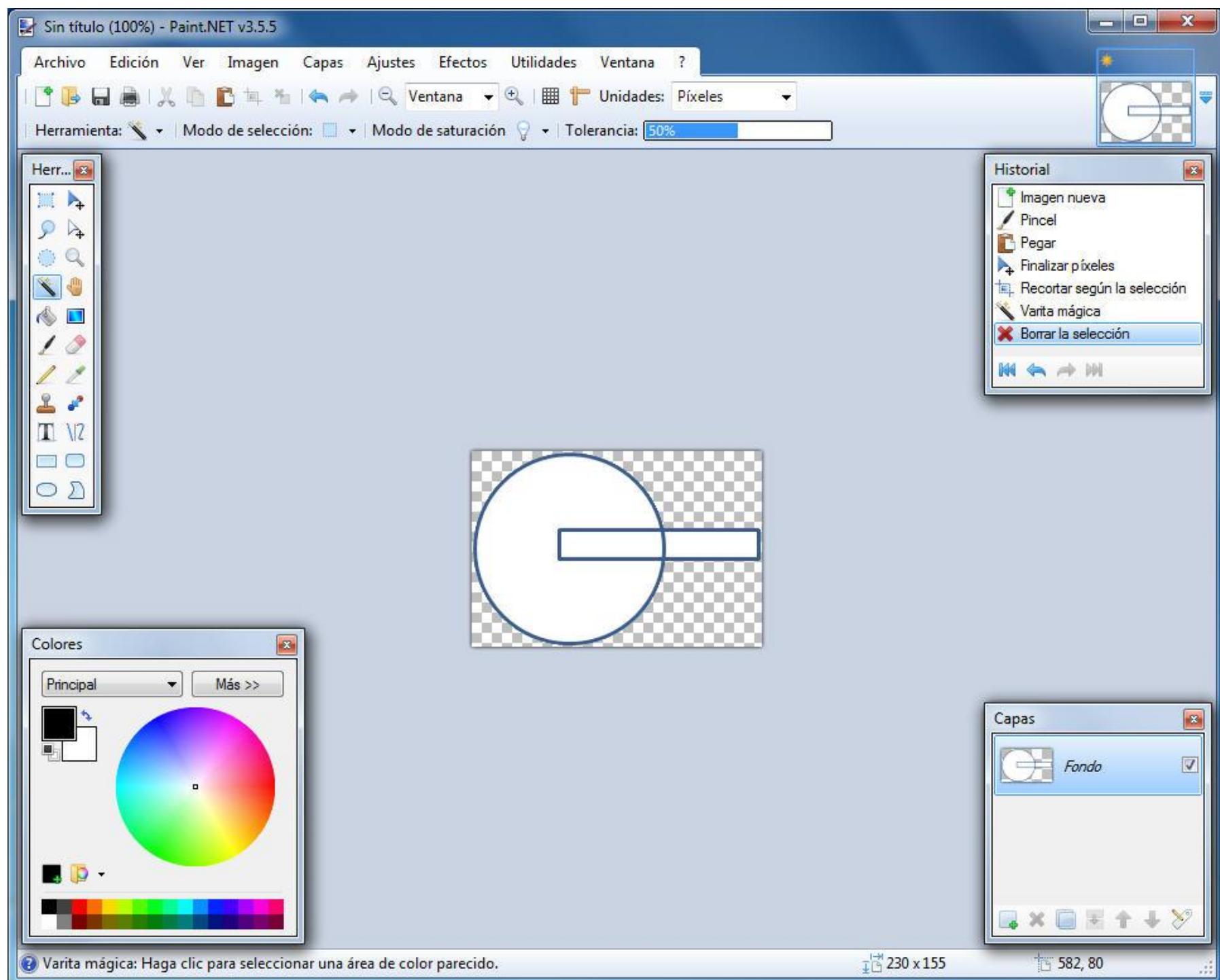


Figura 83: Imagen con área transparente

Guardamos esa imagen con formato .TGA

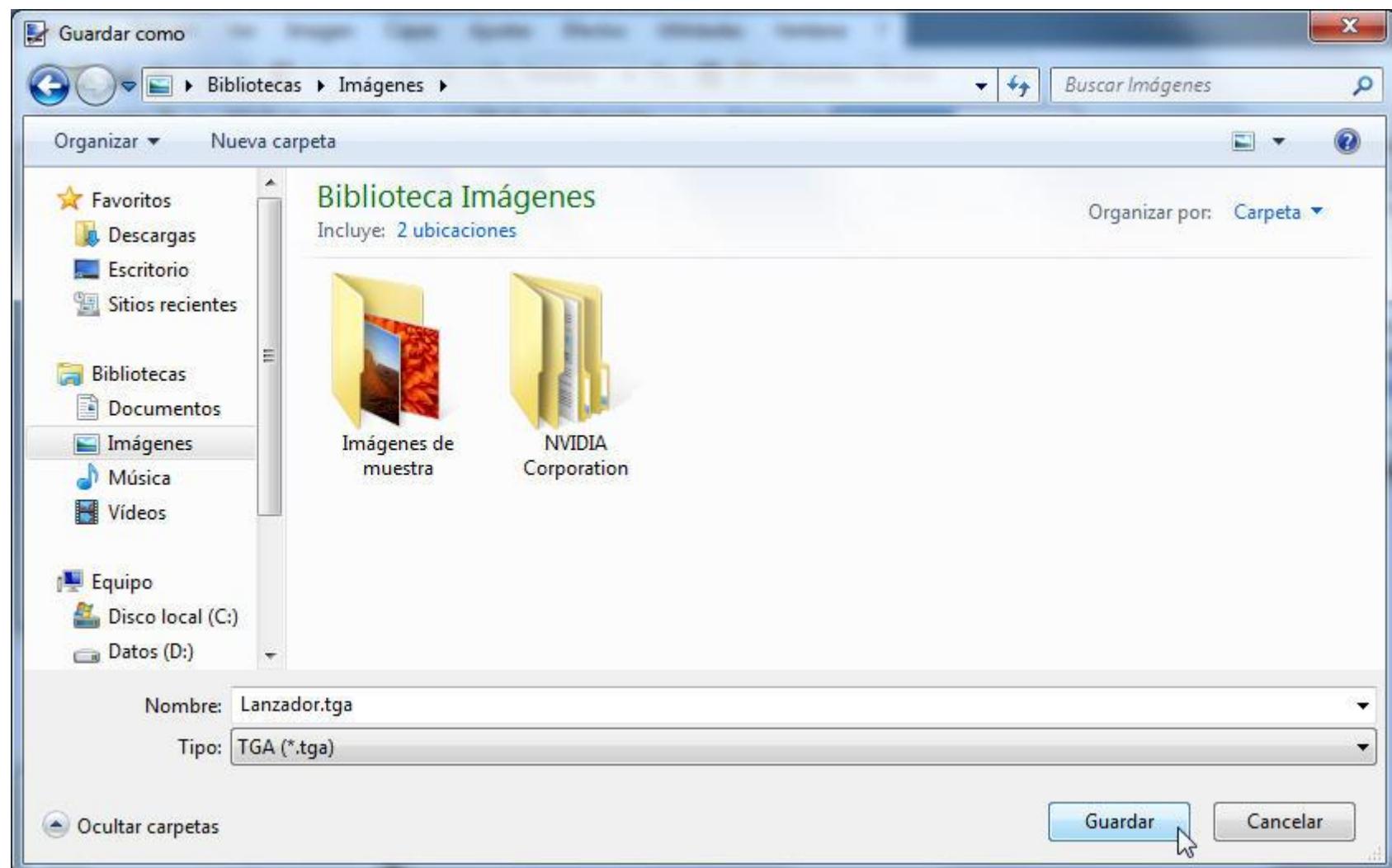
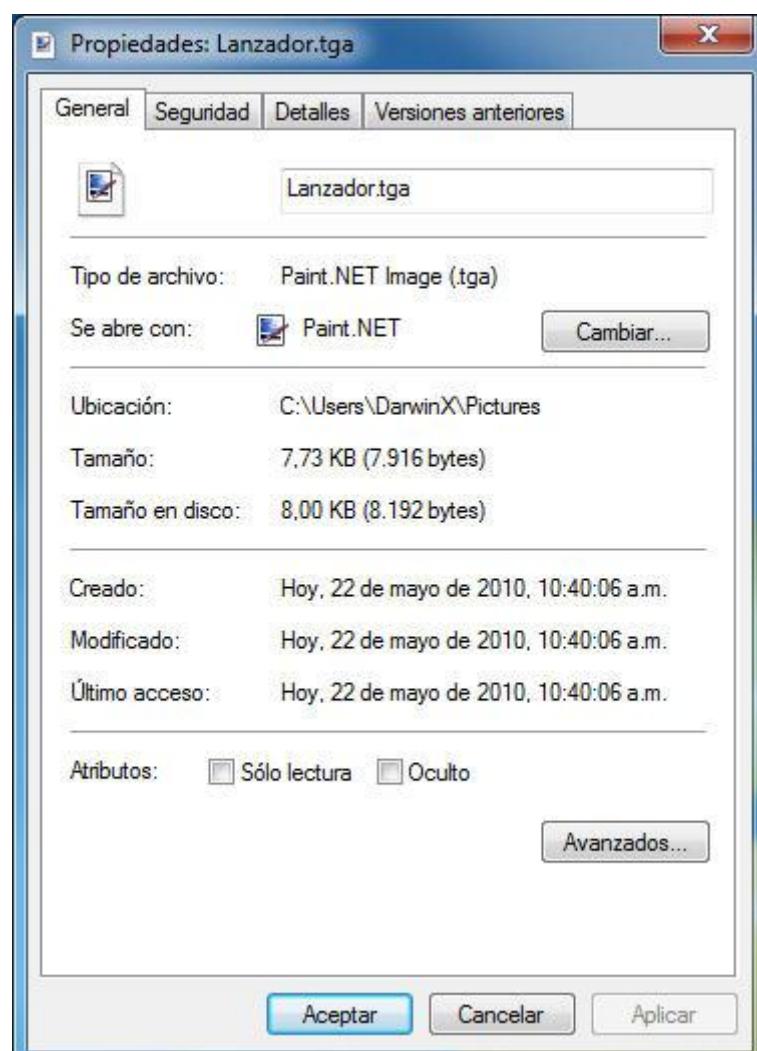


Figura 84: Guardar imagen en formato .TGA

**Figura 85: Configuración del guardado**

Esa es la imagen que debemos llevar al juego

**Figura 86: Característica del archivo a guardar**

## 19. XNA: Agregar el protagonista a nuestro juego

Es el turno de agregar al juego a nuestro héroe, es decir, al protagonista, aquel que controlamos con el teclado, con el mouse o con el gamepad. Debemos tener presente que si desarrollamos en XNA, el juego puede ser llevado a la consola XBOX y allí solo hay gamepad como se ve en la imagen.



**Figura 87: Gamepad**

**Paso 0:** ¿En qué consiste el juego?

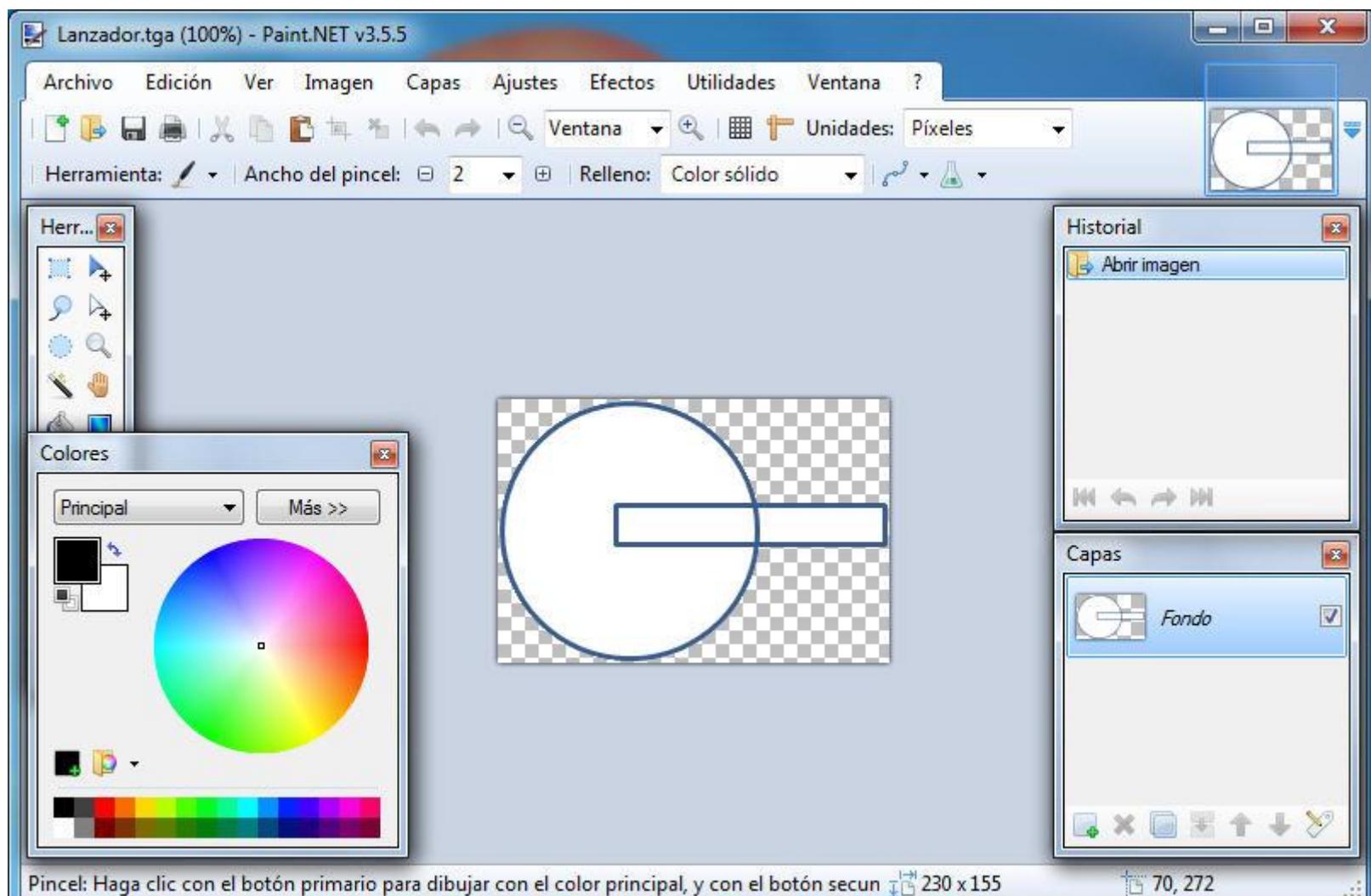
Antes de empezar, debemos saber en qué consiste el juego, para este tutorial se trata de un cañón o lanzador que se encarga de destruir chatarra espacial porque se ha convertido en un problema grave de contaminación espacial.

**Paso 1:** ¿Cómo se comportará nuestro protagonista?

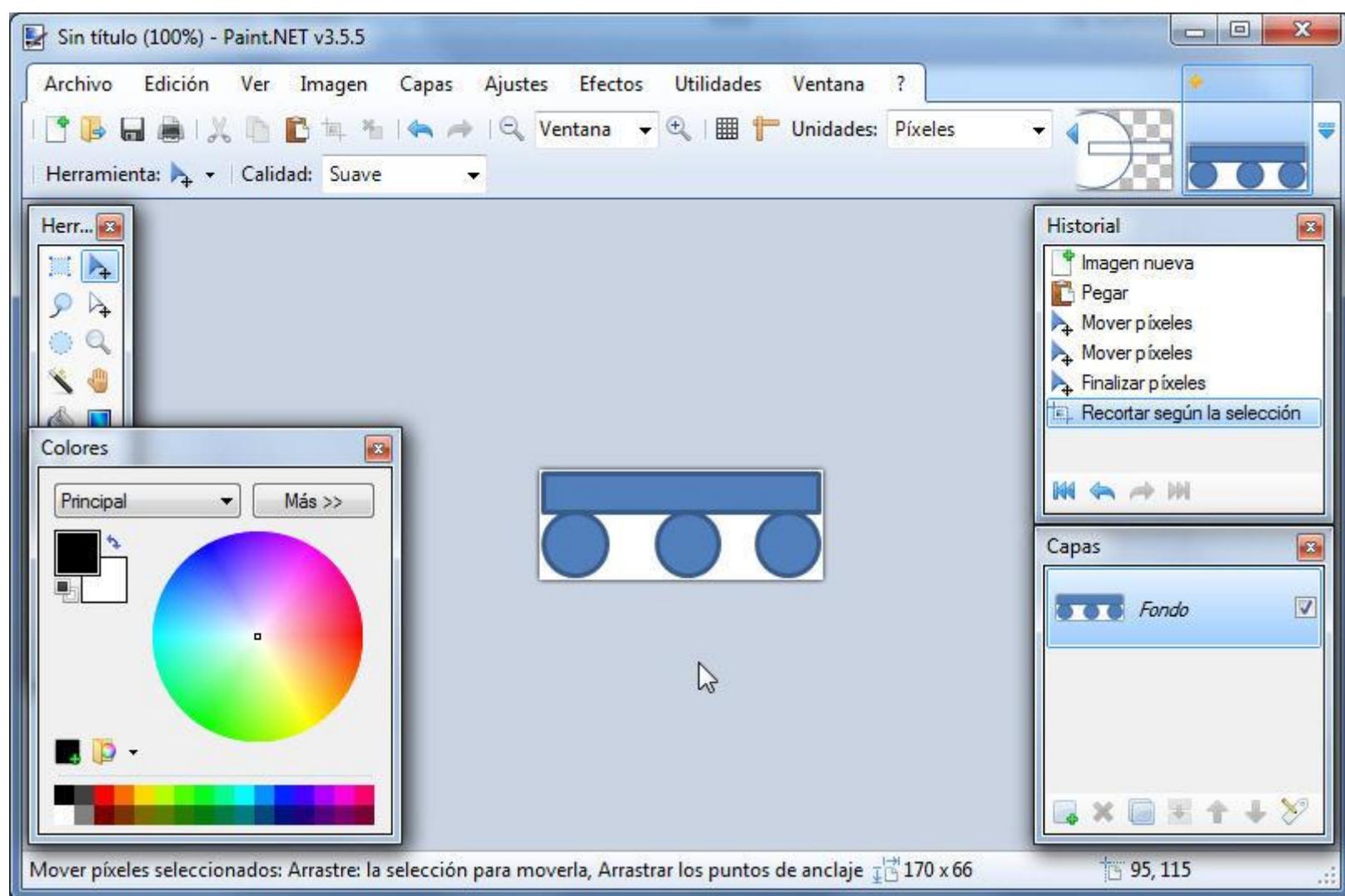
Es un lanzador estacionario, el cual varía su ángulo de lanzamiento y a determinada orden, lanzará un rayo. Luego la animación del protagonista es variación del ángulo de salida del misil.

**Paso 2:** Buscando una imagen acorde a la animación

Podremos buscar una imagen o hacer una propia. Requerimos una base y un lanzador.

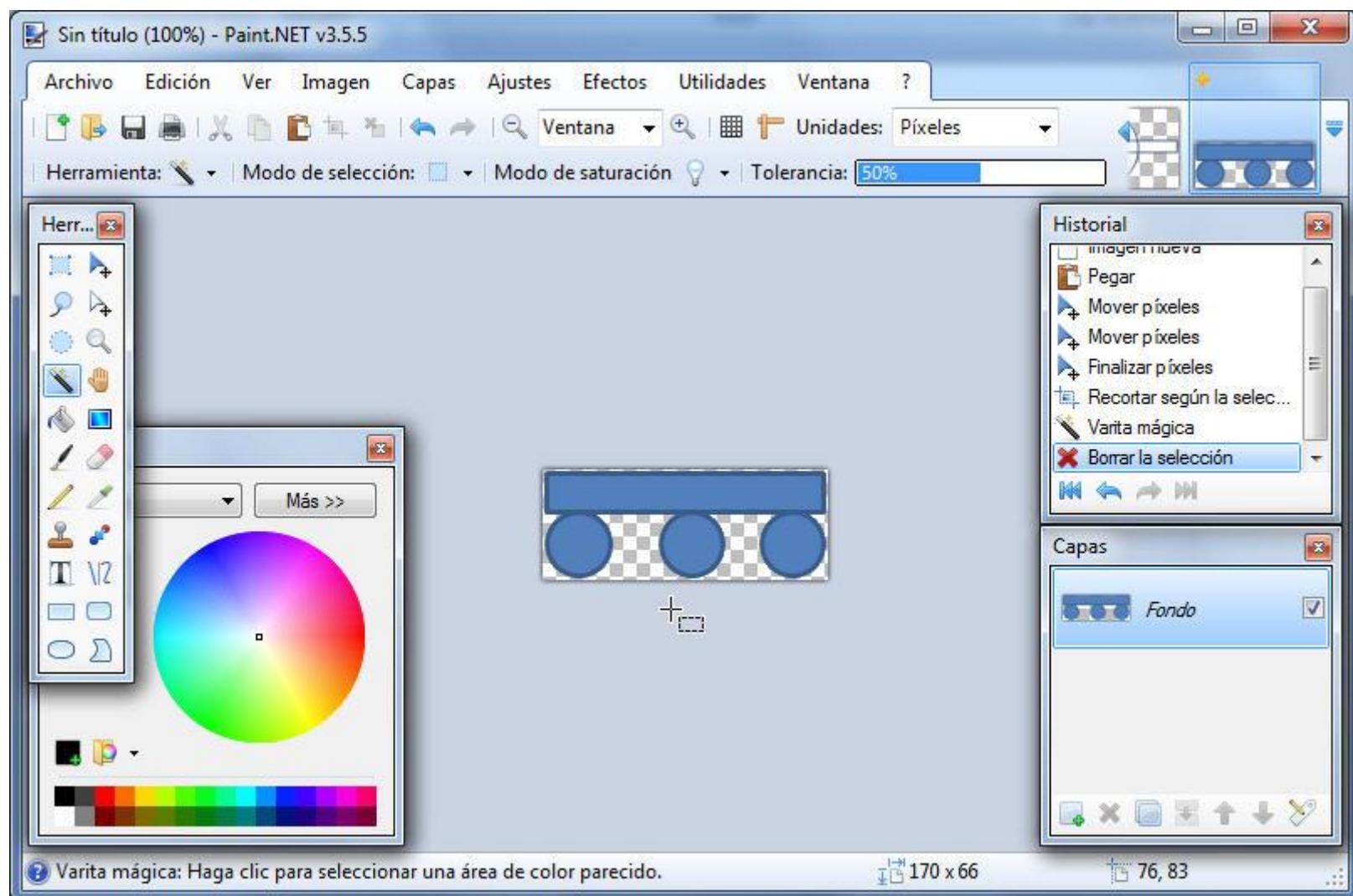


**Figura 88: Imagen del lanzador con la transparencia**



**Figura 89: Imagen de la base del lanzador**

**Paso 3:** Convirtiendo las imágenes con Paint .NET (deben tener fondo transparente) y luego guardarlas en formato .TGA



**Figura 90: Imagen de la base del lanzador con la transparencia**

**Paso 4:** Ahora los movemos al proyecto del juego que se está haciendo.

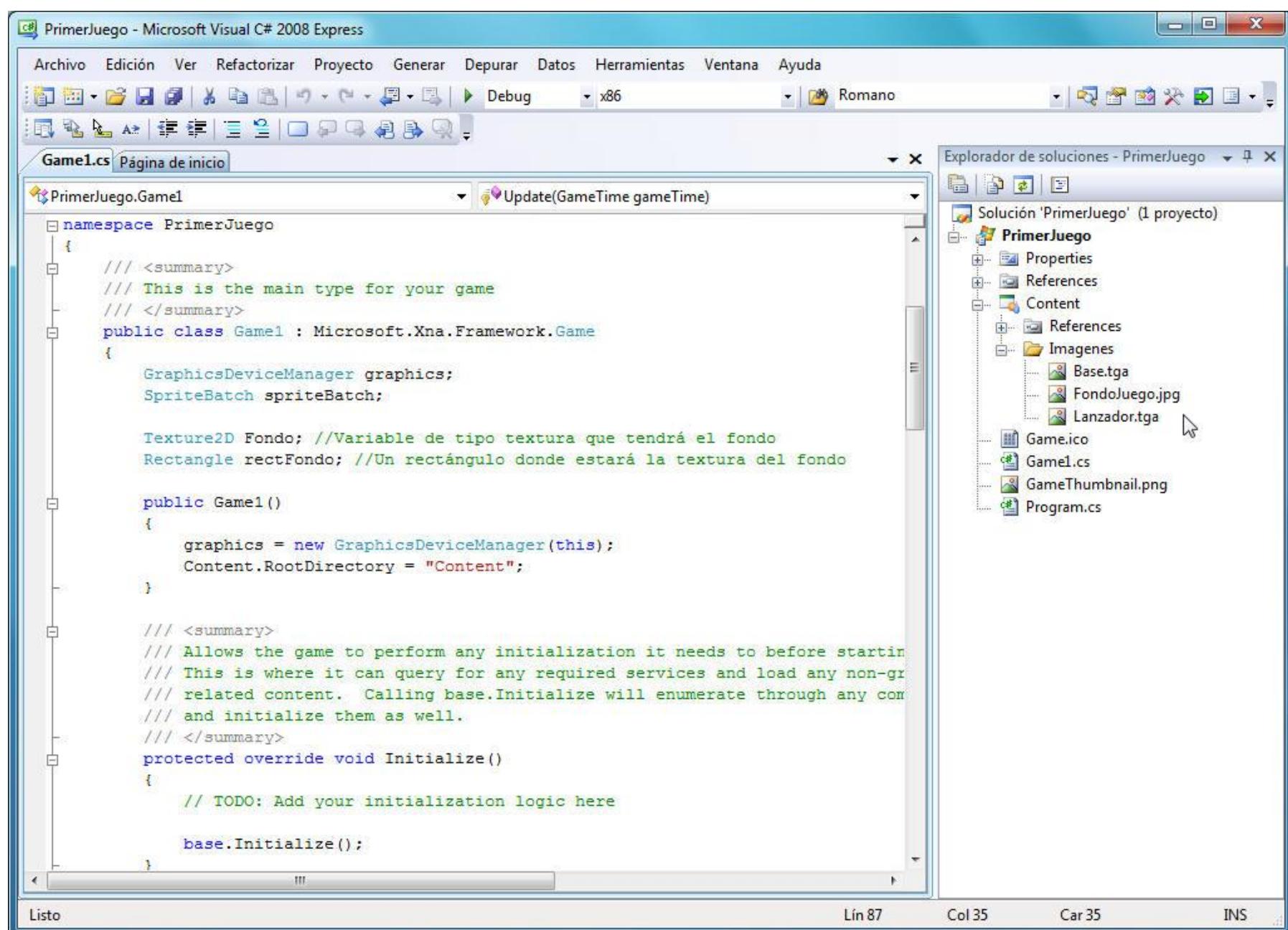


Figura 91: Agregando los recursos (imagenes) al proyecto en XNA

#### Paso 5: Generar la clase que soporta los actores del juego

Los actores (héroe, enemigos, balas) del juego son objetos, por tanto debemos generar una clase que soporte ese tipo de objetos, a la que llamaremos ObjetosJuego

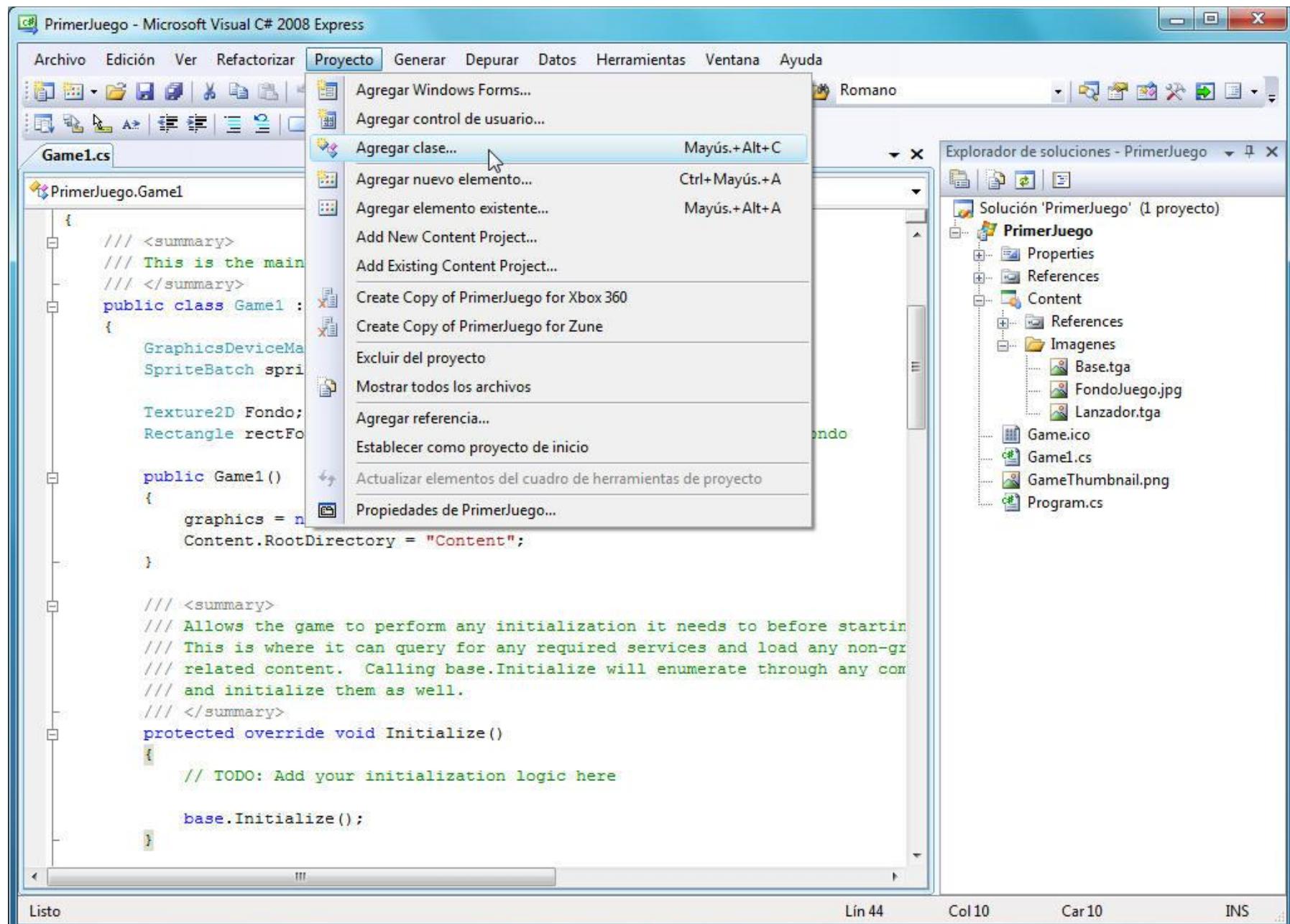


Figura 92: Agregar una clase en Visual C#

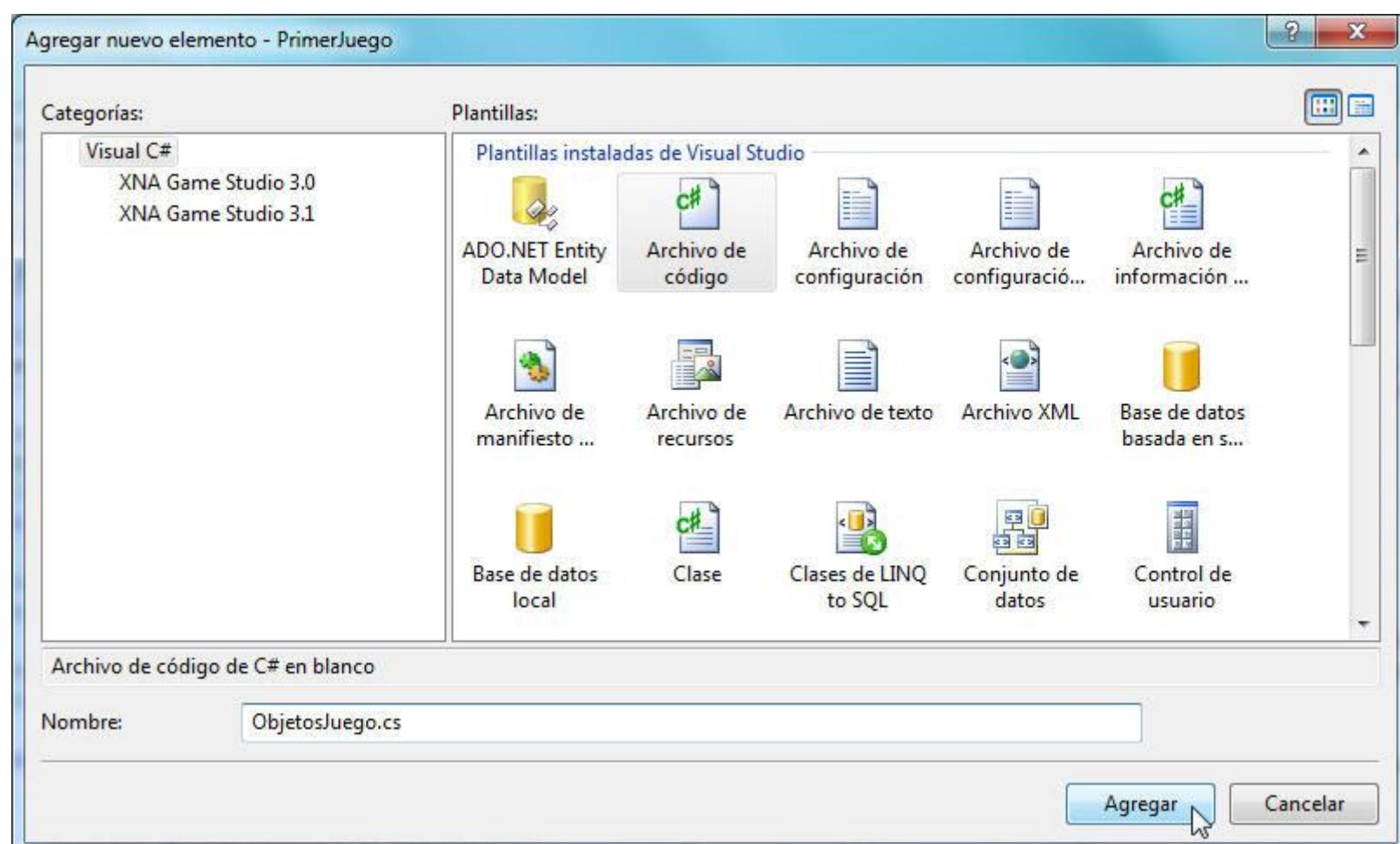


Figura 93: Agregando una clase al proyecto en Visual C#

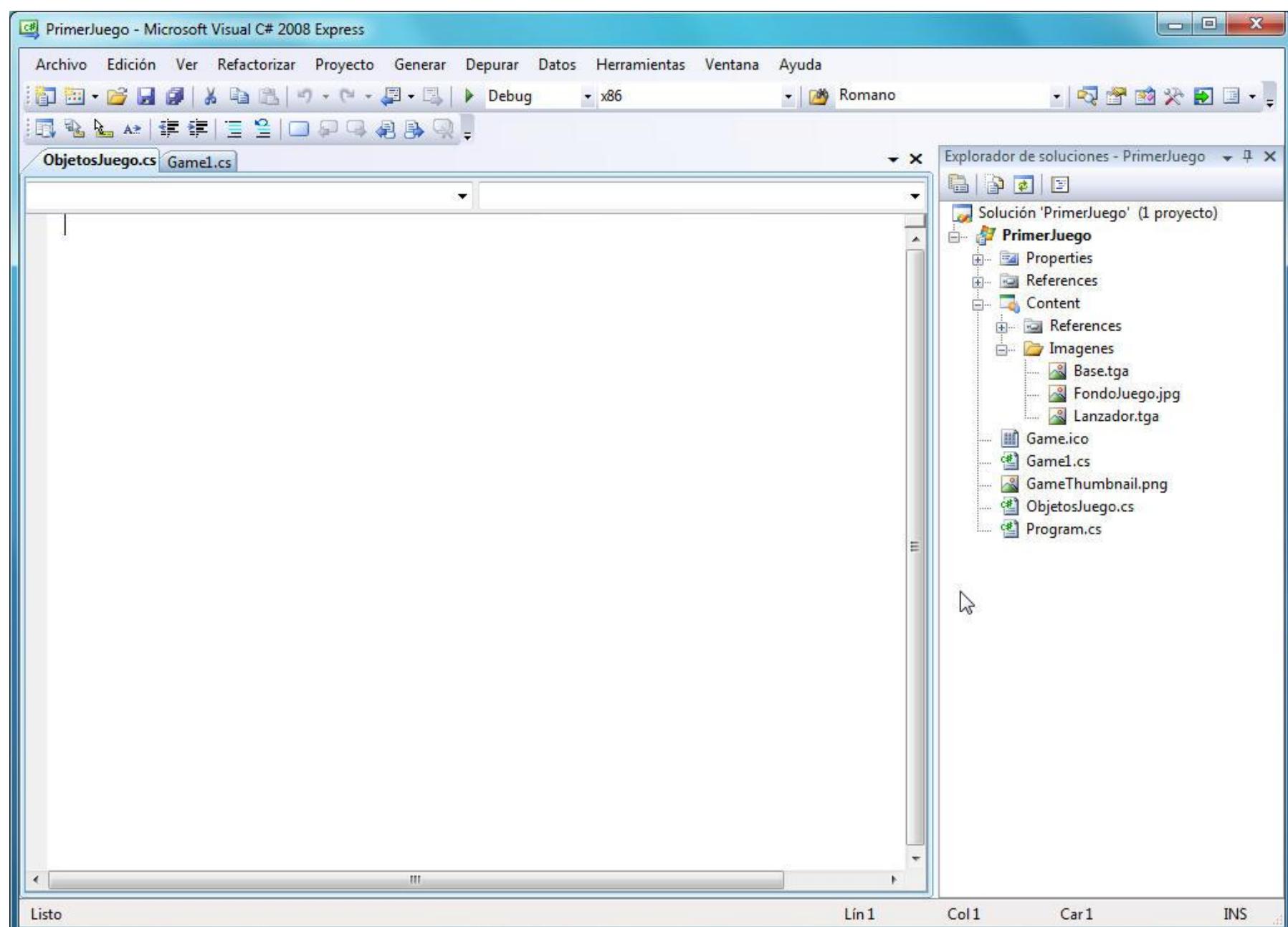


Figura 94: La nueva clase en el proyecto en Visual C#

En esa clase nueva agregamos todas las librerías que importa o usa Game1.cs y escribimos el esqueleto de la clase en sí.

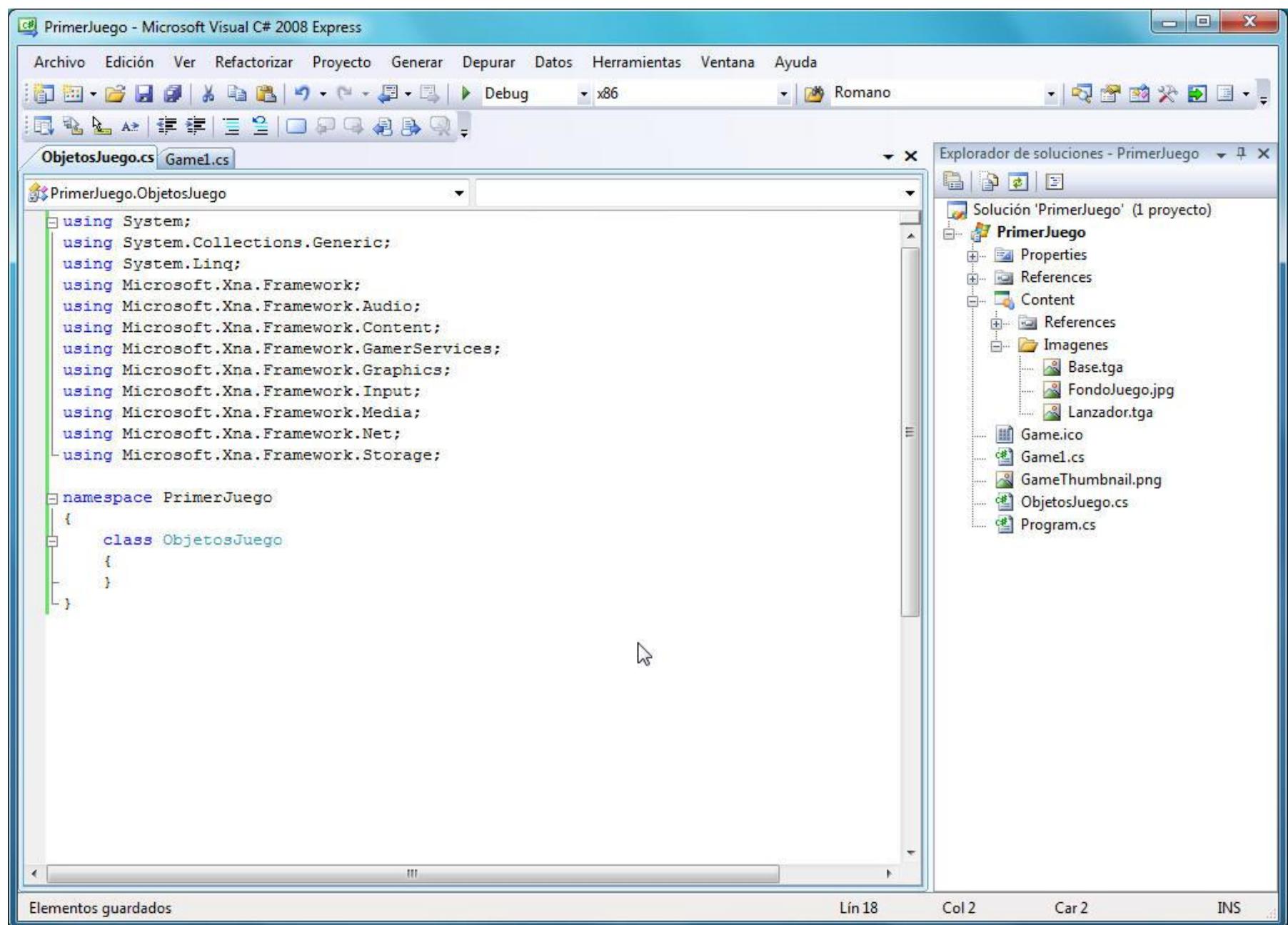


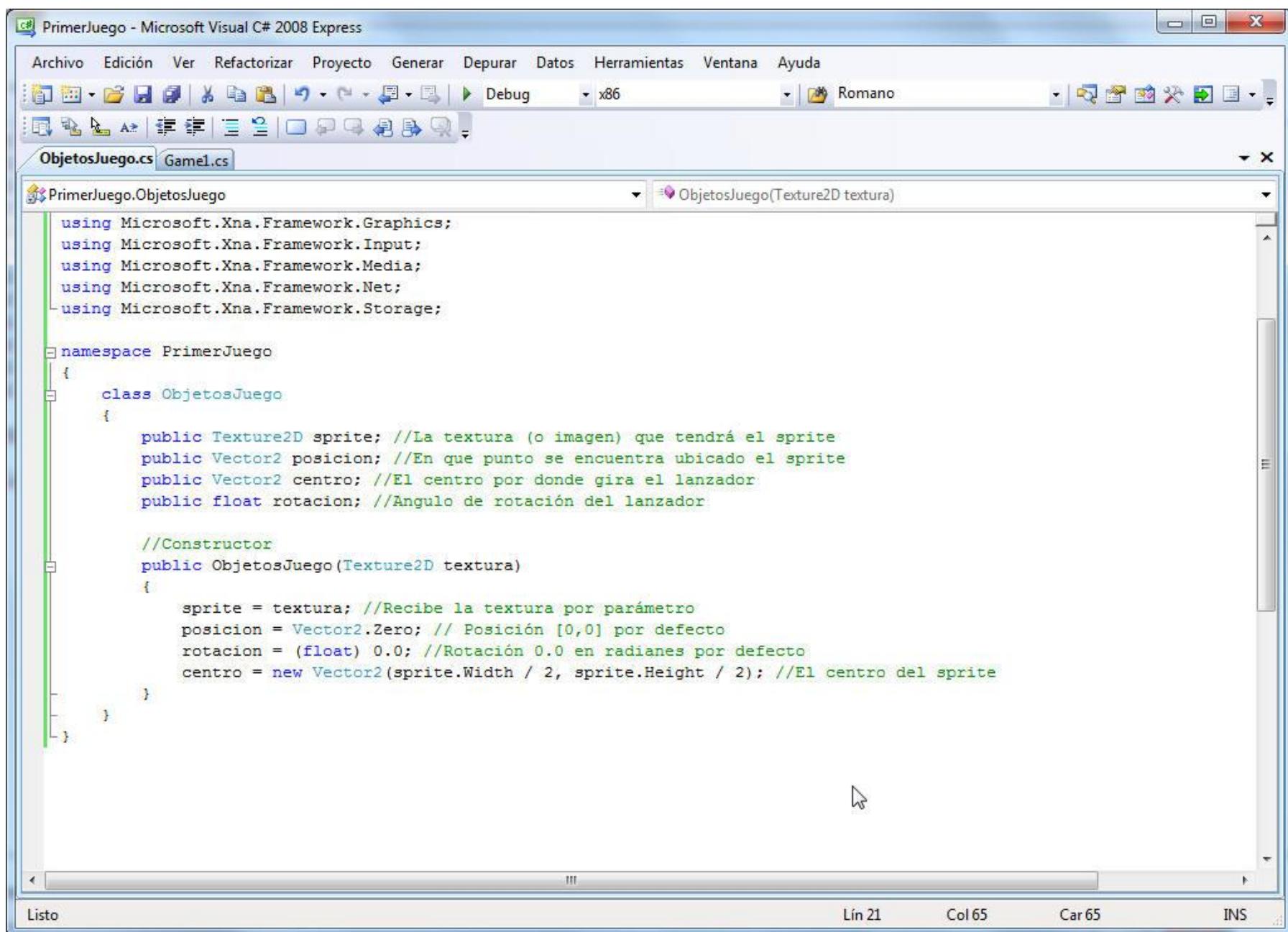
Figura 95: Se agrega el código mínimo para trabajar con XNA

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    class ObjetosJuego
    {
    }
}
```

#### Paso 6: ¿Qué debe ir en la clase que soporta los actores del juego?

Debemos recordar que todos los protagonistas del juego reciben el nombre de "sprite" (un término manejado por la industria). En la nueva clase ObjetosJuego se añaden las propiedades que tendrá nuestro "sprite" lanzador y se genera el constructor por defecto, así queda el código de la clase.



**Figura 96: La clase que controla los diversos objetos en el juego**

Este es el código:

```

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    class ObjetosJuego
    {
        public Texture2D sprite; //La textura (o imagen) que tendrá el sprite
        public Vector2 posicion; //En qué punto se encuentra ubicado el sprite
        public Vector2 centro; //El centro por donde gira el lanzador
        public float rotacion; //Ángulo de rotación del lanzador

        //Constructor
        public ObjetosJuego(Texture2D textura)
        {
            sprite = textura; //Recibe la textura por parámetro
            posicion = Vector2.Zero; // Posición [0,0] por defecto
            rotacion = (float) 0.0; //Rotación 0.0 en radianes por defecto
            centro = new Vector2(sprite.Width / 2, sprite.Height / 2); //El centro del sprite
        }
    }
}

```

**Paso 7:** Hacemos uso de la clase que soporta los actores.

Volvemos de nuevo a Game1.cs y agregamos las dos variables objeto que manejarán nuestro lanzador.

```

namespace PrimerJuego
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch; //Contenedor de los sprite
        Texture2D Fondo; //Variable de tipo textura que tendrá el fondo
        Rectangle rectFondo; //Un rectángulo donde estará la textura del fondo

        ObjetosJuego lanzador; //El lanzador
        ObjetosJuego baselanzador; //Base del lanzador

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
        }
    }
}

```

Figura 97: Se instancia la clase ObjetosJuego para hacer el lanzador y su base

**Paso 8:** Se cargan las texturas de los actores

Ahora debemos inicializar esas variables con las texturas del lanzador y la base y eso se hace en LoadContent()

```

/// <summary>
/// LoadContent will be called once per game and is the place to load
/// all of your content.
/// </summary>
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here

    //Utiliza genéricos <> por eso el uso de <> En C++ se llaman plantillas. Content carga el contenido binario.
    Fondo = Content.Load<Texture2D>("Imagenes\\FondoJuego");

    //El rectángulo en el que estará contenido el fondo
    rectFondo = new Rectangle(0, 0, graphics.GraphicsDevice.Viewport.Width, graphics.GraphicsDevice.Viewport.Height);

    //Cargar e inicializar el Lanzador
    lanzador = new ObjetosJuego(Content.Load<Texture2D>("Imagenes\\Lanzador")); //No es necesaria la extensión .tga
    lanzador.posicion = new Vector2(120, 520); //Posición del lanzador

    //Cargar e inicializar la base
    baselanzador = new ObjetosJuego(Content.Load<Texture2D>("Imagenes\\Base")); //No es necesaria la extensión .tga
    baselanzador.posicion = new Vector2(120, 550); //Posición de la base del lanzador
}

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// </summary>

```

Figura 98: Carga de recursos del videojuego

**Paso 9:** Se muestran las texturas de los actores.

Por último es mostrar nuestro protagonista en pantalla, para eso vamos al método Draw y escribimos el siguiente código.

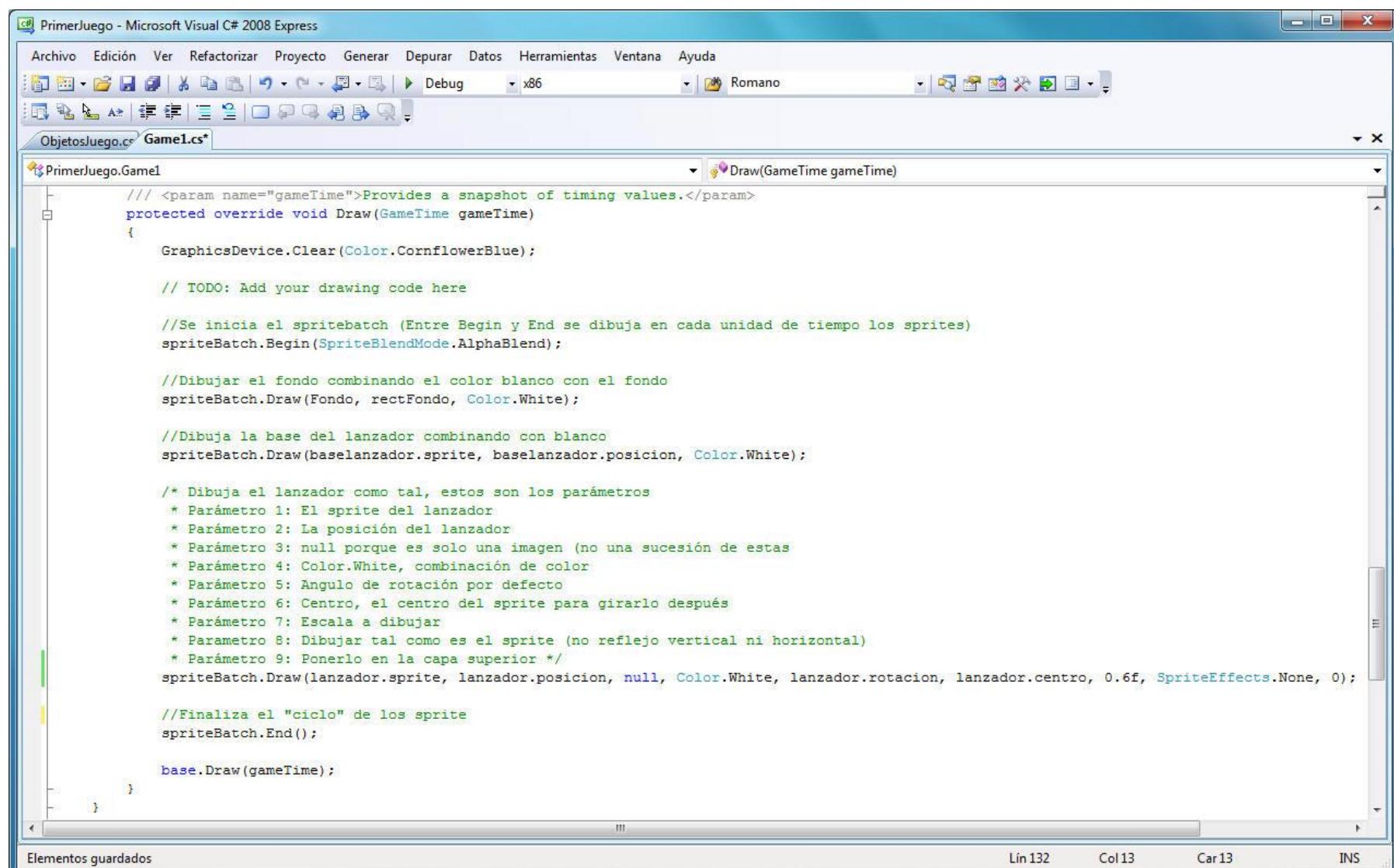


Figura 99: Se agrega código en Draw() para mostrar el lanzador y su base

El código completo de Game1.cs es este que se presenta a continuación

```

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>

    public class Game1 : Microsoft.Xna.Framework.Game
    {

        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch; //Contenedor de los sprite
        Texture2D Fondo; //Variable de tipo textura que tendrá el fondo
        Rectangle rectFondo; //Un rectángulo donde estará la textura del fondo

        ObjetosJuego lanzador; //El lanzador
        ObjetosJuego baselanzador; //Base del lanzador

        public Game1()
        {

            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";

        }

    }
}

```

```

/// Allows the game to perform any initialization it needs to before starting to run.
/// This is where it can query for any required services and load any non-graphic
/// related content. Calling base.Initialize will enumerate through any components
/// and initialize them as well.
/// </summary>

protected override void Initialize()
{
    // TODO: Add your initialization logic here
    base.Initialize();
}

/// <summary>
/// LoadContent will be called once per game and is the place to load
/// all of your content.
/// </summary>
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here

    //Utiliza genéricos <> por eso el uso de <> En C++ se llaman plantillas. Content carga el contenido binario.
    Fondo = Content.Load<Texture2D>("Imagenes\\FondoJuego");

    //El rectángulo en el que estará contenido el fondo
    rectFondo = new Rectangle(0, 0, graphics.GraphicsDevice.Viewport.Width, graphics.GraphicsDevice.Viewport.Height);

    //Cargar e inicializar el Lanzador
    lanzador = new ObjetosJuego(Content.Load<Texture2D>("Imagenes\\Lanzador")); //No es necesaria la extensión .tga
    lanzador.posicion = new Vector2(160, 530); //Posición del lanzador

    //Cargar e inicializar la base
    baselanzador = new ObjetosJuego(Content.Load<Texture2D>("Imagenes\\Base")); //No es necesaria la extensión .tga
    baselanzador.posicion = new Vector2(70, 560); //Posición de la base del lanzador

}

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // TODO: Add your update logic here
    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here

    //Se inicia el spritebatch (Entre Begin y End se dibuja en cada unidad de tiempo los sprites)
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

    //Dibujar el fondo combinando el color blanco con el fondo
    spriteBatch.Draw(Fondo, rectFondo, Color.White);

    //Dibuja la base del lanzador combinando con blanco
    spriteBatch.Draw(baselanzador.sprite, baselanzador.posicion, Color.White);
}

```

```
/* Dibuja el lanzador como tal, estos son los parámetros
 * Parámetro 1: El sprite del lanzador
 * Parámetro 2: La posición del lanzador
 * Parámetro 3: null porque es solo una imagen (no una sucesión de estas)
 * Parámetro 4: Color.White, combinación de color
 * Parámetro 5: Angulo de rotación por defecto
 * Parámetro 6: Centro, el centro del sprite para girarlo después
 * Parámetro 7: Escala a dibujar
 * Parametro 8: Dibujar tal como es el sprite (no reflejo vertical ni horizontal)
 * Parámetro 9: Ponerlo en la capa superior */
spriteBatch.Draw(lanzador.sprite, lanzador.posicion, null, Color.White, lanzador.rotacion, lanzador.centro, 1.0f,
SpriteEffects.None, 0);

//Finaliza el "ciclo" de los sprite
spriteBatch.End();

base.Draw(gameTime);

}

}

}
```

Este es el resultado

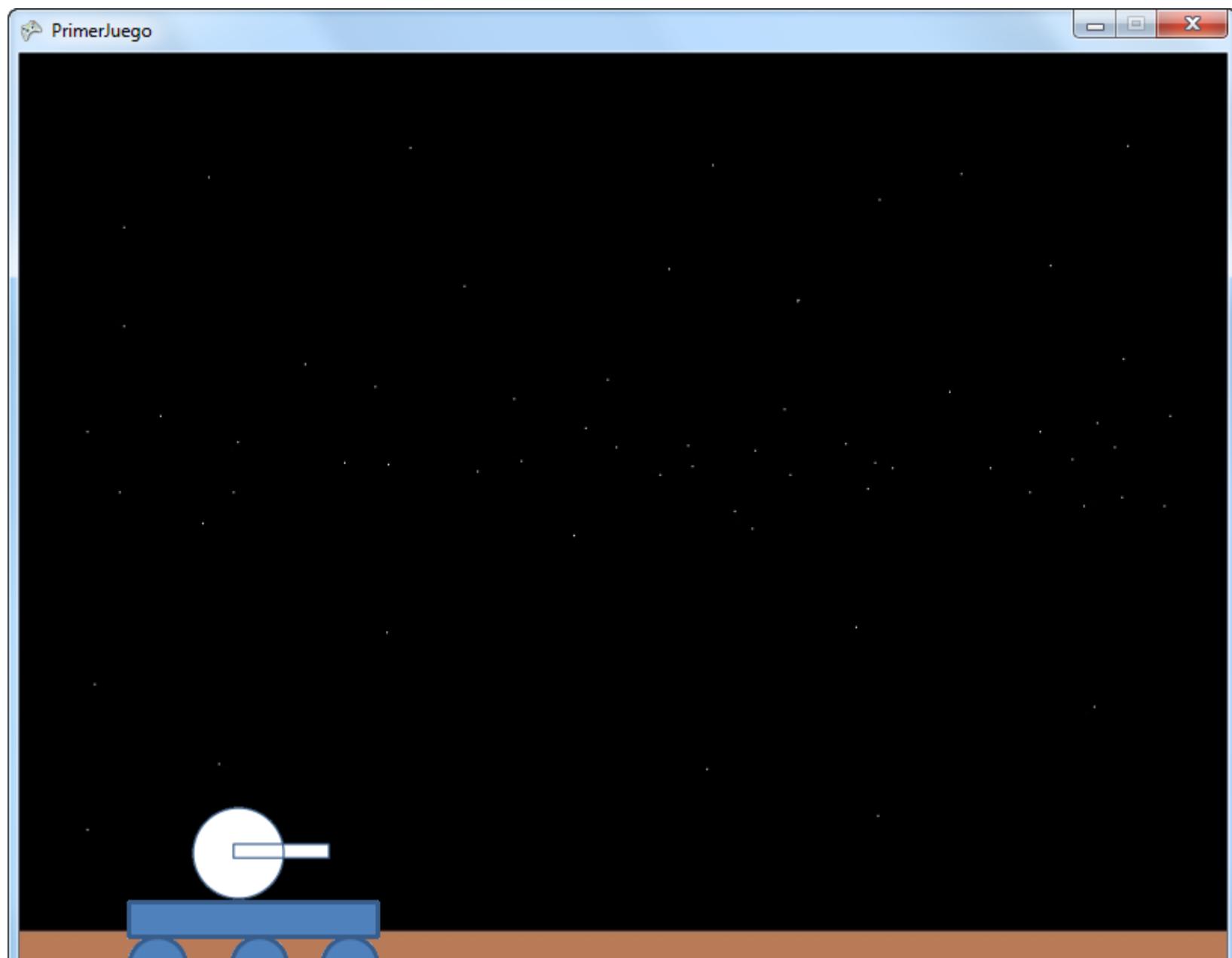


Figura 100: Ejecución del juego mostrando el protagonista

## 20. XNA: Buenas prácticas en el código fuente

Como todo software, es necesario seguir unas buenas prácticas de codificación (no acceder directamente a los atributos de una clase, identación, interlineado), el código anterior se arregla de esta forma.

La clase `ObjetosJuego.cs` ahora se presenta de la siguiente forma

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    //Clase que se encarga de los protagonistas del videojuego
    class ObjetosJuego
    {
        private Texture2D Textura; //La textura (o imagen) que tendrá el sprite
        private Vector2 Posicion; //En que punto se encuentra ubicado el sprite
        private Vector2 Centro; //El centro por donde gira el lanzador
        private float Rotacion; //Angulo de rotación del lanzador

        //Constructor
        public ObjetosJuego(Texture2D Textura)
        {
            this.Textura = Textura; //Recibe la textura por parámetro
            Posicion = Vector2.Zero; // Posición [0,0] por defecto
            Rotacion = (float)0.0; //Rotación 0.0 en radianes por defecto
            Centro = new Vector2(Textura.Width / 2, Textura.Height / 2); //El centro del sprite
        }

        //Leyendo los valores de los atributos
        public Texture2D getTexture() { return Textura; }
        public Vector2 getPosition() { return Posicion; }
        public Vector2 getCentro() { return Centro; }
        public float getRotacion() { return Rotacion; }

        //Dando valores a los atributos
        public void setTexture(Texture2D Textura) { this.Textura = Textura; }
        public void setPosition(Vector2 Posicion) { this.Posicion = Posicion; }
        public void setCentro(Vector2 Centro) { this.Centro = Centro; }
        public void setRotacion(float Rotacion) { this.Rotacion = Rotacion; }
    }
}
```

Y la clase principal (`Game1.cs`), la que ejecuta el juego tiene:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch; //Contenedor de los sprite

        Texture2D Fondo; //Variable de tipo textura que tendrá el fondo
        Rectangle RectFondo; //Un rectángulo donde estará la textura del fondo

        ObjetosJuego Lanzador; //El lanzador
        ObjetosJuego BaseLanzador; //Base del lanzador

        //Constructor
        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }
}
```

```

/// <summary>
/// Allows the game to perform any initialization it needs to before starting to run.
/// This is where it can query for any required services and load any non-graphic
/// related content. Calling base.Initialize will enumerate through any components
/// and initialize them as well.
/// </summary>
protected override void Initialize()
{
    // TODO: Add your initialization logic here
    base.Initialize();
}

/// <summary>
/// LoadContent will be called once per game and is the place to load
/// all of your content.
/// </summary>
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here

    //Utiliza genéricos <> por eso el uso de <> En C++ se llaman plantillas. Content carga el contenido binario.
    Fondo = Content.Load<Texture2D>("Imagenes\\FondoJuego");

    //El rectángulo en el que estará contenido el fondo
    RectFondo = new Rectangle(0, 0, graphics.GraphicsDevice.Viewport.Width, graphics.GraphicsDevice.Viewport.Height);

    //Cargar e inicializar el Lanzador
    Lanzador = new ObjetosJuego(Content.Load<Texture2D>("Imagenes\\Lanzador")); //No es necesaria la extensión .tga
    Lanzador.setPosicion(new Vector2(160, 530)); //Posición del lanzador

    //Cargar e inicializar la base
    BaseLanzador = new ObjetosJuego(Content.Load<Texture2D>("Imagenes\\Base")); //No es necesaria la extensión .tga
    BaseLanzador.setPosicion(new Vector2(70, 560)); //Posición de la base del lanzador
}

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // TODO: Add your update logic here
    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here

    //Se inicia el spritebatch (Entre Begin y End se dibuja en cada unidad de tiempo los sprites)
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

    //Dibujar el fondo combinando el color blanco con el fondo
    spriteBatch.Draw(Fondo, RectFondo, Color.White);

    //Dibuja la base del lanzador combinando con blanco
    spriteBatch.Draw(BaseLanzador.getTextura(), BaseLanzador.getPosicion(), Color.White);

    /* Dibuja el lanzador como tal, estos son los parámetros
     * Parámetro 1: El sprite del lanzador
     * Parámetro 2: La posición del lanzador
     * Parámetro 3: null porque es solo una imagen (no una sucesión de estas)
     * Parámetro 4: Color.White, combinación de color
     * Parámetro 5: Ángulo de rotación por defecto
     * Parámetro 6: Centro, el centro del sprite para girarlo después
     * Parámetro 7: Escala a dibujar
     * Parámetro 8: Dibujar tal como es el sprite (no reflejo vertical ni horizontal)
     * Parámetro 9: Ponerlo en la capa superior */
    spriteBatch.Draw(Lanzador.getTextura(), Lanzador.getPosicion(), null, Color.White, Lanzador.getRotacion(),
Lanzador.getCentro(), 1.0f, SpriteEffects.None, 0);
}

```

```
//Finaliza el "ciclo" de los sprite  
spriteBatch.End();  
  
    base.Draw(gameTime);  
}  
}
```

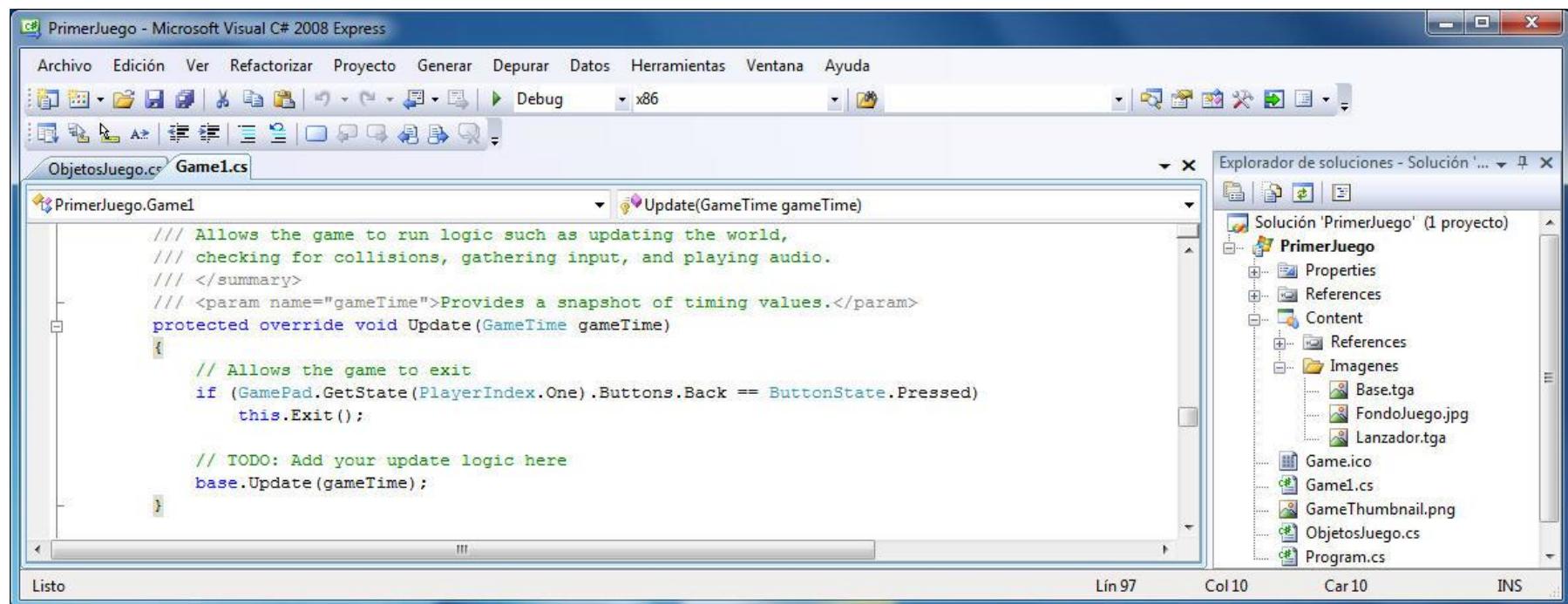
## 21. XNA: Controlando al protagonista de nuestro juego

El control de un juego se logra con el ratón, con el teclado o con el gamepad. Si jugamos con XBOX tenemos gamepad pero no tenemos teclado.



**Figura 101: Gamepad**

El código a escribir para leer lo que hace el usuario con algún control es en `Update()`, como se ve en la imagen



**Figura 102: El control del juego se programa en `Update()`**

El código original de este método es el siguiente:

```
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

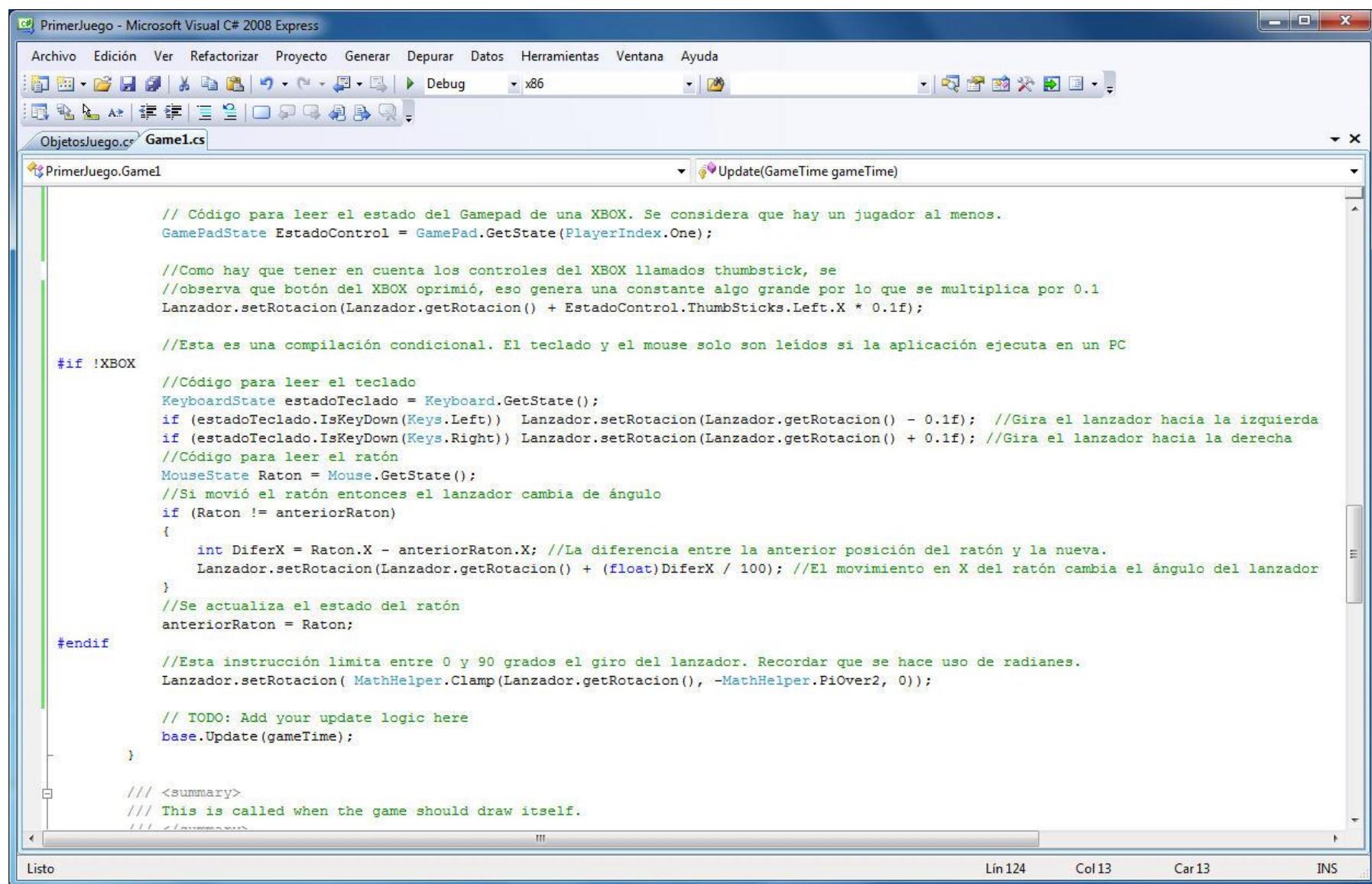
    // TODO: Add your update logic here
    base.Update(gameTime);
}
```

Procedemos entonces a modificar ese método agregando nuestro código tanto para controles XBOX como el teclado y ratón en PC. En una gamepad hay un control llamado thumbstick con el que se controlará el ángulo del lanzador.



**Figura 103: Thumbstick del gamepad**

Hay que agregar una directiva de compilación en Visual C#, para que en caso que la aplicación ejecute en una XBOX no intente leer teclado ni ratón.



```

PrimerJuego - Microsoft Visual C# 2008 Express
Archivo Edición Ver Refactorizar Proyecto Generar Depurar Datos Herramientas Ventana Ayuda
ObjetosJuego.cs Game1.cs
PrimerJuego.Game1 Update(GameTime gameTime)
{
    // Código para leer el estado del Gamepad de una XBOX. Se considera que hay un jugador al menos.
    GamePadState EstadoControl = GamePad.GetState(PlayerIndex.One);

    //Como hay que tener en cuenta los controles del XBOX llamados thumbstick, se
    //observa que botón del XBOX oprimió, eso genera una constante algo grande por lo que se multiplica por 0.1
    Lanzador.setRotacion(Lanzador.getRotacion() + EstadoControl.ThumbSticks.Left.X * 0.1f);

    //Esta es una compilación condicional. El teclado y el mouse solo son leidos si la aplicación ejecuta en un PC
#if !XBOX
    //Código para leer el teclado
    KeyboardState estadoTeclado = Keyboard.GetState();
    if (estadoTeclado.IsKeyDown(Keys.Left)) Lanzador.setRotacion(Lanzador.getRotacion() - 0.1f); //Gira el lanzador hacia la izquierda
    if (estadoTeclado.IsKeyDown(Keys.Right)) Lanzador.setRotacion(Lanzador.getRotacion() + 0.1f); //Gira el lanzador hacia la derecha
    //Código para leer el ratón
    MouseState Raton = Mouse.GetState();
    //Si movió el ratón entonces el lanzador cambia de ángulo
    if (Raton != anteriorRaton)
    {
        int DiferX = Raton.X - anteriorRaton.X; //La diferencia entre la anterior posición del ratón y la nueva.
        Lanzador.setRotacion(Lanzador.getRotacion() + (float)DiferX / 100); //El movimiento en X del ratón cambia el ángulo del lanzador
    }
    //Se actualiza el estado del ratón
    anteriorRaton = Raton;
#endif
    //Esta instrucción limita entre 0 y 90 grados el giro del lanzador. Recordar que se hace uso de radianes.
    Lanzador.setRotacion( MathHelper.Clamp(Lanzador.getRotacion(), -MathHelper.PiOver2, 0));

    // TODO: Add your update logic here
    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>

```

**Figura 104: Código para leer el gamepad, mouse y teclado**

Las instrucciones `#if ... #endif` se conocen como Preprocessor Directives y son necesarias si requiere generar el código dependiendo de parámetros del compilador. En este caso si el ejecutable va para una XBOX o para un PC.

Debemos adicionar la declaración de `anteriorRaton` al inicio de la clase como se ve en la imagen.

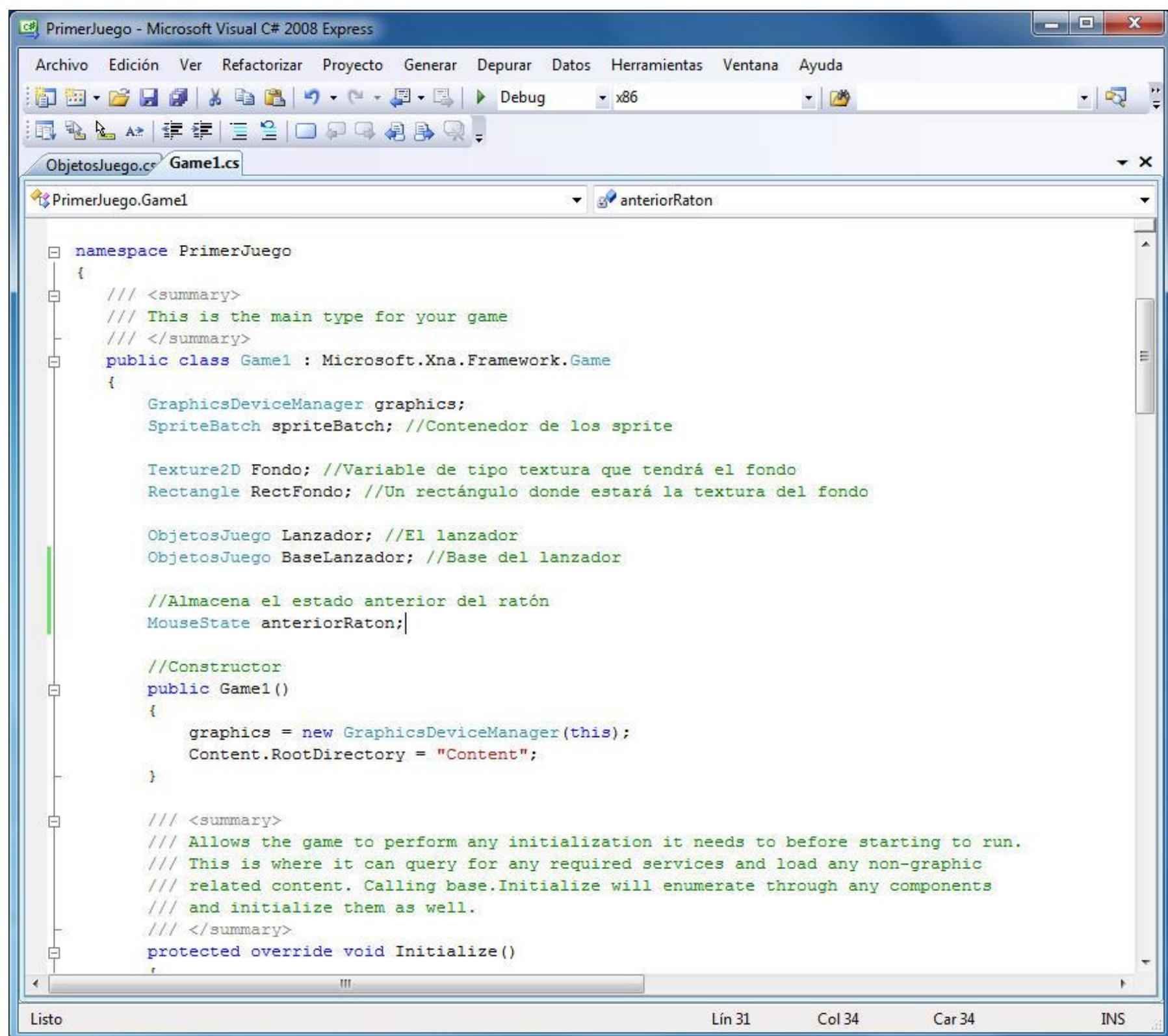


Figura 105: Instancia de estado del ratón

El código completo de Game1.cs se muestra a continuación:

```

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch; //Contenedor de los sprite

        Texture2D Fondo; //Variable de tipo textura que tendrá el fondo
        Rectangle RectFondo; //Un rectángulo donde estará la textura del fondo

        ObjetosJuego Lanzador; //El lanzador
        ObjetosJuego BaseLanzador; //Base del lanzador

        //Almacena el estado anterior del ratón
        MouseState anteriorRaton;

        //Constructor
        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
        }
    }
}

```

```

graphics = new GraphicsDeviceManager(this);
Content.RootDirectory = "Content";
}

/// <summary>
/// Allows the game to perform any initialization it needs to before starting to run.
/// This is where it can query for any required services and load any non-graphic
/// related content. Calling base.Initialize will enumerate through any components
/// and initialize them as well.
/// </summary>
protected override void Initialize()
{
    // TODO: Add your initialization logic here
    base.Initialize();
}

/// <summary>
/// LoadContent will be called once per game and is the place to load
/// all of your content.
/// </summary>
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here

    //Utiliza genéricos <> por eso el uso de <> En C++ se llaman plantillas. Content carga el contenido binario.
    Fondo = Content.Load<Texture2D>("Imagenes\\FondoJuego");

    //El rectángulo en el que estará contenido el fondo
    RectFondo = new Rectangle(0, 0, graphics.GraphicsDevice.Viewport.Width, graphics.GraphicsDevice.Viewport.Height);

    //Cargar e inicializar el Lanzador
    Lanzador = new ObjetosJuego(Content.Load<Texture2D>("Imagenes\\Lanzador")); //No es necesaria la extensión .tga
    Lanzador.setPosicion(new Vector2(160, 530)); //Posición del lanzador

    //Cargar e inicializar la base
    BaseLanzador = new ObjetosJuego(Content.Load<Texture2D>("Imagenes\\Base")); //No es necesaria la extensión .tga
    BaseLanzador.setPosicion(new Vector2(70, 560)); //Posición de la base del lanzador
}

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // Código para leer el estado del Gamepad de una XBOX. Se considera que hay un jugador al menos.
    GamePadState EstadoControl = GamePad.GetState(PlayerIndex.One);

    //Como hay que tener en cuenta los controles del XBOX llamados thumbstick, se
    //observa que botón del XBOX oprimió, eso genera una constante algo grande por lo que se multiplica por 0.1
    Lanzador.setRotacion(Lanzador.getRotacion() + EstadoControl.ThumbSticks.Left.X * 0.1f);

    //Esta es una compilación condicional. El teclado y el mouse solo son leídos si la aplicación ejecuta en un PC
#if !XBOX
    //Código para leer el teclado
    KeyboardState estadoTeclado = Keyboard.GetState();
    if (estadoTeclado.IsKeyDown(Keys.Left)) Lanzador.setRotacion(Lanzador.getRotacion() - 0.1f); //Gira el lanzador hacia la
    izquierda
    if (estadoTeclado.IsKeyDown(Keys.Right)) Lanzador.setRotacion(Lanzador.getRotacion() + 0.1f); //Gira el lanzador hacia la
    derecha
    //Código para leer el ratón
    MouseState Raton = Mouse.GetState();
    //Si movió el ratón entonces el lanzador cambia de ángulo
    if (Raton != anteriorRaton)
    {
        int DiferX = Raton.X - anteriorRaton.X; //La diferencia entre la anterior posición del ratón y la nueva.
        Lanzador.setRotacion(Lanzador.getRotacion() + (float)DiferX / 100); //El movimiento en X del ratón cambia el ángulo del
        lanzador
    }
    //Se actualiza el estado del ratón
    anteriorRaton = Raton;
#endif
    //Esta instrucción limita entre 0 y 90 grados el giro del lanzador. Recordar que se hace uso de radianes.
    Lanzador.setRotacion(MathHelper.Clamp(Lanzador.getRotacion(), -MathHelper.PiOver2, 0));

    // TODO: Add your update logic here
}

```

```

    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here

    //Se inicia el spritebatch (Entre Begin y End se dibuja en cada unidad de tiempo los sprites)
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

    //Dibujar el fondo combinando el color blanco con el fondo
    spriteBatch.Draw(Fondo, RectFondo, Color.White);

    //Dibuja la base del lanzador combinando con blanco
    spriteBatch.Draw(BaseLanzador.getTextura(), BaseLanzador.getPosicion(), Color.White);

    /* Dibuja el lanzador como tal, estos son los parámetros
     * Parámetro 1: El sprite del lanzador
     * Parámetro 2: La posición del lanzador
     * Parámetro 3: null porque es solo una imagen (no una sucesión de estas)
     * Parámetro 4: Color.White, combinación de color
     * Parámetro 5: Angulo de rotación por defecto
     * Parámetro 6: Centro, el centro del sprite para girarlo después
     * Parámetro 7: Escala a dibujar
     * Parametro 8: Dibujar tal como es el sprite (no reflejo vertical ni horizontal)
     * Parámetro 9: Ponerlo en la capa superior */
    spriteBatch.Draw(Lanzador.getTextura(), Lanzador.getPosicion(), null, Color.White, Lanzador.getRotacion(),
Lanzador.getCentro(), 1.0f, SpriteEffects.None, 0);

    //Finaliza el "ciclo" de los sprite
    spriteBatch.End();

    base.Draw(gameTime);
}
}
}

```

## 22. XNA: El protagonista lanza rayos

Nuestro lanzador empezará a disparar rayos. Similar a lo hecho con el lanzador, buscamos o creamos una imagen representativa del rayo y le hacemos su tratamiento respectivo con Paint .NET

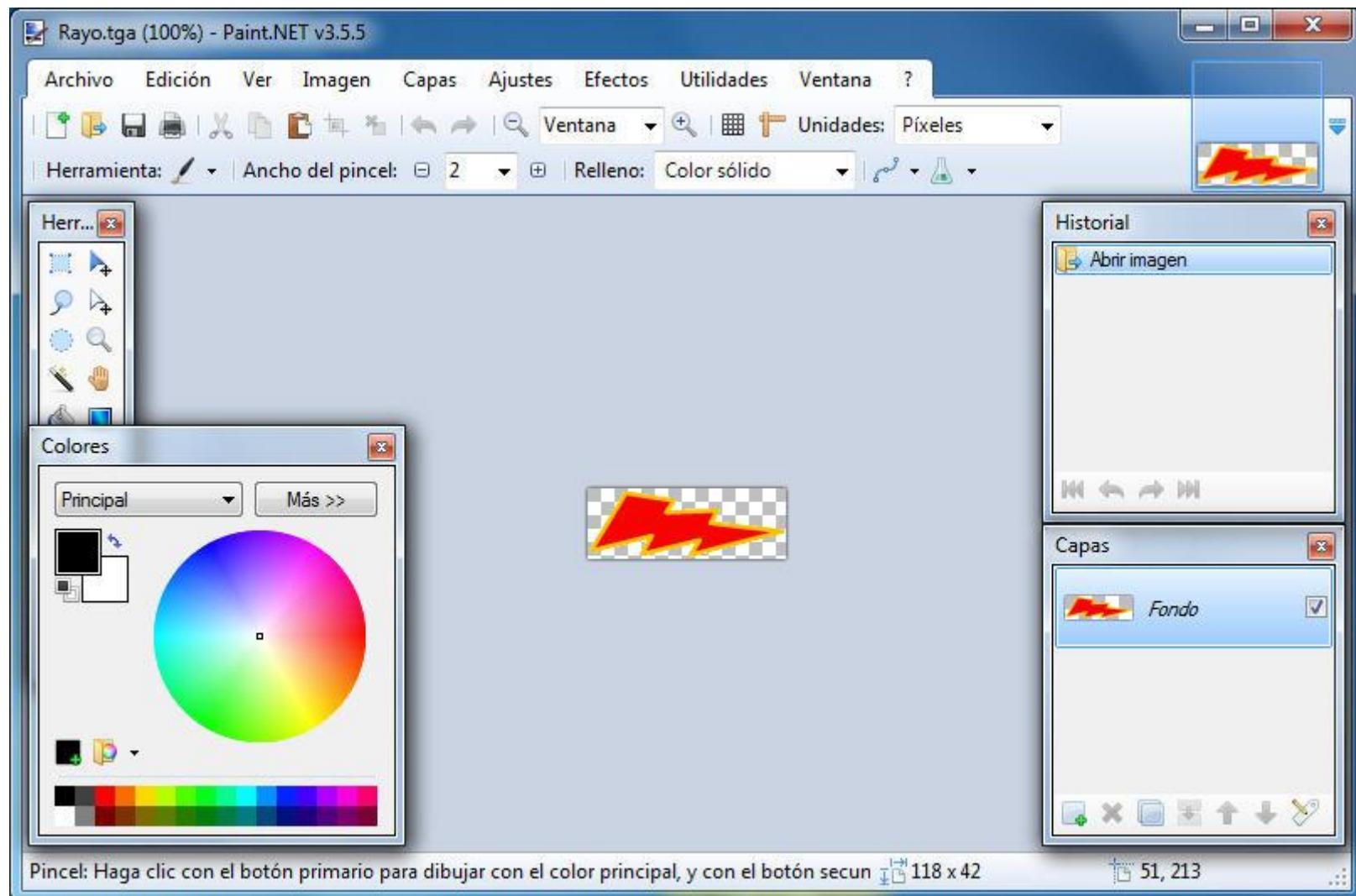


Figura 106: Tratamiento de transparencia en la imagen del rayo

Se añade a la biblioteca de imágenes del proyecto.

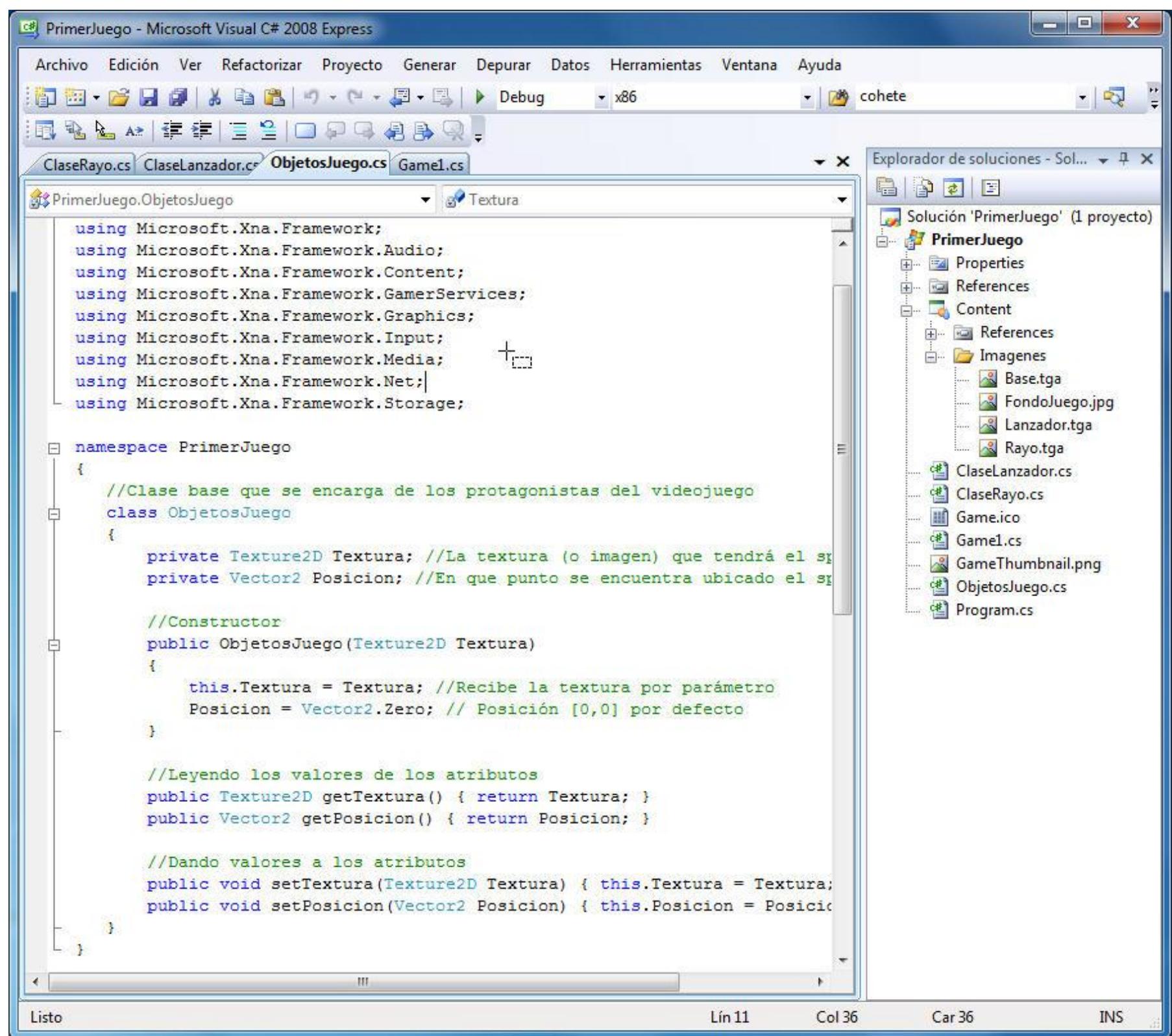


Figura 107: La imagen es entonces puesta en la carpeta de recursos del proyecto

Ya tenemos un nuevo tipo de protagonista en nuestro juego, difiere del lanzador en algunos atributos, una buena idea es hacer que `ObjetosJuego.cs` sea una clase base para los protagonistas y de allí heredar clases hijas: una clase para el lanzador y otra clase para los rayos.

Entonces, `ObjetosJuego.cs` queda de esta manera

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    //Clase base que se encarga de los protagonistas del videojuego
    class ObjetosJuego
    {
        private Texture2D Textura; //La textura (o imagen) que tendrá el sprite
        private Vector2 Posicion; //En que punto se encuentra ubicado el sprite

        //Constructor
        public ObjetosJuego(Texture2D Textura)
        {
            this.Textura = Textura; //Recibe la textura por parámetro
            Posicion = Vector2.Zero; // Posición [0,0] por defecto
        }

        //Leyendo los valores de los atributos
        public Texture2D getTexture() { return Textura; }
        public Vector2 getPosition() { return Posicion; }

        //Dando valores a los atributos
        public void setTexture(Texture2D Textura) { this.Textura = Textura; }
        public void setPosition(Vector2 Posicion) { this.Posicion = Posicion; }
    }
}
```

```
//Dando valores a los atributos
public void setTexture(Texture2D Textura) { this.Textura = Textura; }
public void setPosicion(Vector2 Posicion) { this.Posicion = Posicion; }
}
```

Las clases hijas entonces se particularizan dependiendo del tipo de actor en el juego.

Para el lanzador esta sería la clase ClaseLanzador.cs

```
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    //Clase que administra el lanzador
    class ClaseLanzador : ObjetosJuego
    {
        private Vector2 Centro; //El centro por donde gira el lanzador
        private float Rotacion; //Angulo de rotación del lanzador

        //Constructor
        public ClaseLanzador(Texture2D Textura) : base (Textura)
        {
            Rotacion = (float)0.0; //Rotación 0.0 en radianes por defecto
            Centro = new Vector2(Textura.Width / 2, Textura.Height / 2); //El centro del sprite
        }

        //Leyendo los valores de los atributos
        public Vector2 getCentro() { return Centro; }
        public float getRotacion() { return Rotacion; }

        //Dando valores a los atributos
        public void setCentro(Vector2 Centro) { this.Centro = Centro; }
        public void setRotacion(float Rotacion) { this.Rotacion = Rotacion; }
    }
}
```

Cada rayo tiene una vida limitada: nace en el momento en que es disparado por el lanzador, viaja por la ventana y desaparece cuando llega al borde de esta. Luego es necesario saber cuando el rayo se mantiene activo, es decir, cuando se muestra por pantalla. Para eso debemos tener un atributo a la clase ClaseRayo.cs de tipo booleano que cuando tiene el valor de "true" significa que el rayo esta activo y debe mostrarse, caso contrario, mantenerlo oculto.

No solamente eso, también hay algo importante con los rayos: se desplazan por la pantalla, el desplazamiento tiene una dirección y una velocidad, por lo tanto debemos tener un atributo de velocidad (magnitud y dirección) a la clase ClaseRayo.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    class ClaseRayo : ObjetosJuego
    {
        private Vector2 Centro; //El centro por donde gira el lanzador
        private float Rotacion; //Angulo de rotación del lanzador
        private bool Activo; //Sabe si esta activo o no el rayo
        private Vector2 Velocidad; //Dirección y velocidad del rayo

        //Constructor
        public ClaseRayo(Texture2D Textura) : base (Textura)
        {
            Rotacion = (float)0.0; //Rotación 0.0 en radianes por defecto
            Centro = new Vector2(Textura.Width / 2, Textura.Height / 2); //El centro del sprite

            Activo = false; //Por defecto no se muestra el rayo
            Velocidad = Vector2.Zero; //Por defecto el rayo no se mueve
        }

        //Leyendo los valores de los atributos
        public Vector2 getCentro() { return Centro; }
        public float getRotacion() { return Rotacion; }
    }
}
```

```

public bool getActivo() { return Activo; }
public Vector2 getVelocidad() { return Velocidad; }

//Dando valores a los atributos
public void setCentro(Vector2 Centro) { this.Centro = Centro; }
public void setRotacion(float Rotacion) { this.Rotacion = Rotacion; }
public void setActivo(bool Activo) { this.Activo = Activo; }
public void setVelocidad(Vector2 Velocidad) { this.Velocidad = Velocidad; }
}
}

```

Así debe lucir el IDE

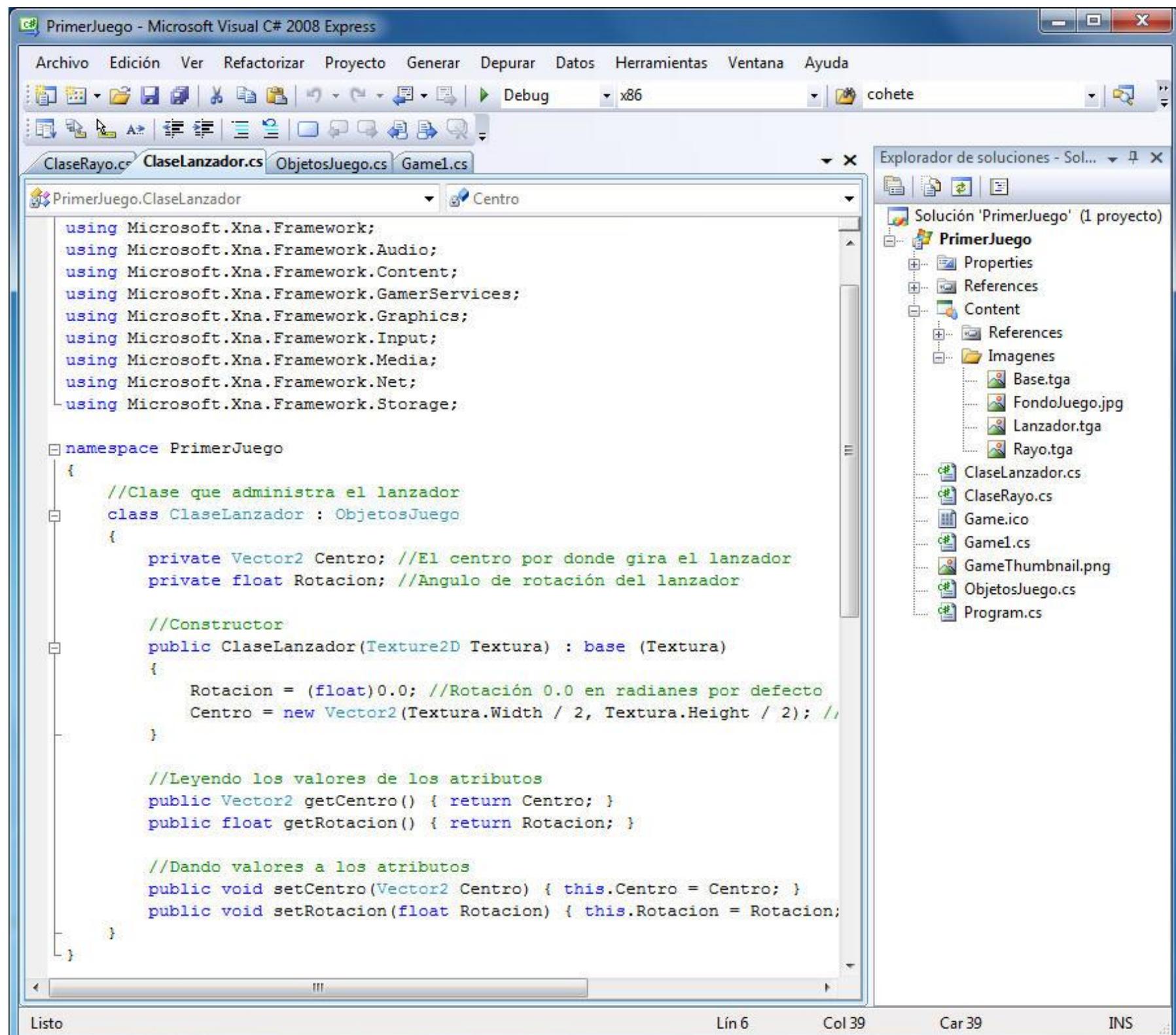


Figura 108: Código para el manejo de los rayos disparados

Como vamos a lanzar varios rayos al tiempo necesitamos una lista que se crea en Game1.cs

```

//Rays
ClaseRayo[] Rayos;
const int MAXRAYOS = 7;

```

Y por supuesto, en el LoadContent se carga la imagen de los rayos

```

//Cargar e inicializar los rayos
Rayos = new ClaseRayo[MAXRAYOS];
for (int Cont = 0; Cont < MAXRAYOS; Cont++)
    Rayos[Cont] = new ClaseRayo(Content.Load<Texture2D>("Imagenes\\Rayo"));

```

Ahora viene la programación del rayo como tal y estas son las cosas que hay que tener en cuenta:

1. El rayo se dispara cuando el usuario presiona la tecla de disparo. Hay que validar que el usuario no mantenga presionada esa tecla constantemente, debe soltarla.
2. Si un rayo está en estado inactivo, entonces es candidato a ser usado cuando el usuario presione la tecla de disparar.
3. El rayo una vez lanzado comienza a desplazarse por la ventana. Luego debemos actualizar su posición constantemente dada la velocidad y el ángulo de salida.
4. El rayo, una vez se salga de la ventana debe inactivarse.

Veamos caso por caso:

**Caso 1:** El rayo se dispara cuando el usuario presiona la tecla de disparo. Hay que validar que el usuario no mantenga presionada esa tecla constantemente, debe soltarla.

Necesitamos guardar el estado anterior del teclado y solo se llamaría al método que dispara el cohete cuando el usuario presione y posteriormente suelte la tecla de disparo.

Se crea un nuevo atributo en la clase Game1

```
//Almacena el estado anterior del teclado
KeyboardState anteriorTeclado;
```

y este es el código que lanza el disparo, el cual se ubica en Update()

```
//Dispara un solo rayo.
//Chequea si el usuario mantiene presionada la tecla de disparo por lo que hace caso omiso a eso,
//el usuario debe presionar y soltar la tecla de disparo para lanzar mas rayos.
if (estadoTeclado.IsKeyDown(Keys.Space) && anteriorTeclado.IsKeyUp(Keys.Space))
    DispararRayo();
```

**Caso 2:** Si un rayo está en estado inactivo, entonces es candidato a ser usado cuando el usuario presione la tecla de disparar.

Hay un método llamado DispararRayo que activa el rayo e inicializa su comportamiento (velocidad, ángulo de salida, ángulo de rotación del sprite)

```
//Método llamado desde Update
public void DispararRayo()
{
    //Busca un rayo inactivo
    foreach (ClaseRayo Rayo in Rayos)
    {
        //Si encuentra un rayo inactivo
        if (Rayo.getActivo() == false)
        {
            //Activa el rayo (para que sea dibujado)
            Rayo.setActivo(true);

            //El rayo inicia en la posición del lanzador
            Rayo.setPosicion(Lanzador.getPosicion());

            //En que dirección y a que velocidad sale el rayo. Hay que tener en cuenta la rotación del lanzador.
            Rayo.setVelocidad(new Vector2((float)Math.Cos(Lanzador.getRotacion()), (float)Math.Sin(Lanzador.getRotacion())) * 4.0f);

            //Rote el sprite del rayo para que coincida con el del lanzador
            Rayo.setRotacion(Lanzador.getRotacion());

            return;
        }
    }
}
```

**Caso 3:** El rayo una vez lanzado comienza a desplazarse por la ventana. Luego debemos actualizar su posición constantemente dada la velocidad y el ángulo de salida con esta instrucción en Update()

```
//Actualiza los rayos
ActualizaRayos();
```

Ese método es:

```
//Método llamado desde Update
public void ActualizaRayos()
{
    //Va de rayo en rayo
    foreach (ClaseRayo Rayo in Rayos)
    {
        if (Rayo.getActivo() == true)
        {
            //Solo es actualizar la velocidad porque el vector se actualiza gracias a esa magnitud
            Rayo.setPosicion(Rayo.getPosicion() + Rayo.getVelocidad());

            //Si el rayo se sale de la pantalla entonces se desactiva
            if (RectFondo.Contains(new Point((int)Rayo.getPosicion().X, (int)Rayo.getPosicion().Y)) == false)
                Rayo.setActivo(false);
        }
    }
}
```

Y ese método resuelve el punto 4. El cohete, una vez se salga de la ventana debe inactivarse.

El código completo de Game1.cs es el siguiente:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
```

```

using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch; //Contenedor de los sprite

        Texture2D Fondo; //Variable de tipo textura que tendrá el fondo
        Rectangle RectFondo; //Un rectángulo donde estará la textura del fondo

        ClaseLanzador Lanzador; //El lanzador
        ClaseLanzador BaseLanzador; //Base del lanzador

        //Almacena el estado anterior del ratón
        MouseState anteriorRaton;

        //Almacena el estado anterior del teclado
        KeyboardState anteriorTeclado;

        //Rayos
        ClaseRayo[] Rayos;
        const int MAXRAYOS = 7;

        //Constructor
        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here
            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // TODO: use this.Content to load your game content here

            //Utiliza genéricos <> por eso el uso de <> En C++ se llaman plantillas. Content carga el contenido binario.
            Fondo = Content.Load<Texture2D>("Imagenes\\FondoJuego");

            //El rectángulo en el que estará contenido el fondo
            RectFondo = new Rectangle(0, 0, graphics.GraphicsDevice.Viewport.Width, graphics.GraphicsDevice.Viewport.Height);

            //Cargar e inicializar el Lanzador
            Lanzador = new ClaseLanzador(Content.Load<Texture2D>("Imagenes\\Lanzador")); //No es necesaria la extensión .tga
            Lanzador.setPosicion(new Vector2(160, 530)); //Posición del lanzador

            //Cargar e inicializar la base
            BaseLanzador = new ClaseLanzador(Content.Load<Texture2D>("Imagenes\\Base")); //No es necesaria la extensión .tga
            BaseLanzador.setPosicion(new Vector2(70, 560)); //Posición de la base del lanzador

            //Cargar e inicializar los rayos
            Rayos = new ClaseRayo[MAXRAYOS];
            for (int Cont = 0; Cont < MAXRAYOS; Cont++)
                Rayos[Cont] = new ClaseRayo(Content.Load<Texture2D>("Imagenes\\Rayo"));

        }

        /// <summary>
        /// UnloadContent will be called once per game and is the place to unload
        /// all content.
        /// </summary>
        protected override void UnloadContent()
        {
            // TODO: Unload any non ContentManager content here
        }

        /// <summary>

```

```

/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // Código para leer el estado del Gamepad de una XBOX. Se considera que hay un jugador al menos.
    GamePadState EstadoControl = GamePad.GetState(PlayerIndex.One);

    //Como hay que tener en cuenta los controles del XBOX llamados thumbstick, se
    //observa que botón del XBOX oprimió, eso genera una constante algo grande por lo que se multiplica por 0.1
    Lanzador.setRotacion(Lanzador.getRotacion() + EstadoControl.ThumbSticks.Left.X * 0.1f);

    //Esta es una compilación condicional. El teclado y el mouse solo son leídos si la aplicación ejecuta en un PC
#if !XBOX
    //Código para leer el teclado
    KeyboardState estadoTeclado = Keyboard.GetState();
    if (estadoTeclado.IsKeyDown(Keys.Left)) Lanzador.setRotacion(Lanzador.getRotacion() - 0.1f); //Gira el lanzador hacia la
    izquierda
    if (estadoTeclado.IsKeyDown(Keys.Right)) Lanzador.setRotacion(Lanzador.getRotacion() + 0.1f); //Gira el lanzador hacia la
    derecha

    //Código para leer el ratón
    MouseState Raton = Mouse.GetState();

    //Si movió el ratón entonces el lanzador cambia de ángulo
    if (Raton != anteriorRaton)
    {
        int DiferX = Raton.X - anteriorRaton.X; //La diferencia entre la anterior posición del ratón y la nueva.
        Lanzador.setRotacion(Lanzador.getRotacion() + (float)DiferX / 100); //El movimiento en X del ratón cambia el ángulo del
        lanzador
    }

    //Se actualiza el estado del ratón
    anteriorRaton = Raton;

    //Dispara un solo rayo.
    //Chequea si el usuario mantiene presionada la tecla de disparo por lo que hace caso omiso a eso,
    //el usuario debe presionar y soltar la tecla de disparo para lanzar mas rayos.
    if (estadoTeclado.IsKeyDown(Keys.Space) && anteriorTeclado.IsKeyUp(Keys.Space))
        DispararRayo();

    anteriorTeclado = estadoTeclado;
#endif

    //Esta instrucción limita entre 0 y 90 grados el giro del lanzador. Recordar que se hace uso de radianes.
    Lanzador.setRotacion( MathHelper.Clamp(Lanzador.getRotacion(), -MathHelper.PiOver2, 0));

    //Actualiza los rayos
    ActualizaRayos();

    // TODO: Add your update logic here
    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here

    //Se inicia el spritebatch (Entre Begin y End se dibuja en cada unidad de tiempo los sprites)
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

    //Dibujar el fondo combinando el color blanco con el fondo
    spriteBatch.Draw(Fondo, RectFondo, Color.White);

    //Dibuja la base del lanzador combinando con blanco
    spriteBatch.Draw(BaseLanzador.getTextura(), BaseLanzador.getPosicion(), Color.White);

    /* Dibuja el lanzador como tal, estos son los parámetros
     * Parámetro 1: El sprite del lanzador
     * Parámetro 2: La posición del lanzador
     * Parámetro 3: null porque es solo una imagen (no una sucesión de estas)
     * Parámetro 4: Color.White, combinación de color
     * Parámetro 5: Angulo de rotación por defecto
     * Parámetro 6: Centro, el centro del sprite para girarlo después
     * Parámetro 7: Escala a dibujar
     * Parametro 8: Dibujar tal como es el sprite (no reflejo vertical ni horizontal)
     * Parámetro 9: Ponerlo en la capa superior */
    spriteBatch.Draw(Lanzador.getTextura(), Lanzador.getPosicion(), null, Color.White, Lanzador.getRotacion(),
Lanzador.getCentro(), 1f, SpriteEffects.None, 0);

    //Dibuja cada rayo
    foreach (ClaseRayo Rayo in Rayos)

```

```

if (Rayo.getActivo() == true)
    spriteBatch.Draw(Rayo.getTextura(), Rayo.getPosicion(), null, Color.White, Rayo.getRotacion(), Rayo.getCentro(), 1f,
SpriteEffects.None, 0);

//Finaliza el "ciclo" de los sprite
spriteBatch.End();

base.Draw(gameTime);
}

//Método llamado desde Update
public void DispararRayo()
{
    //Busca un rayo inactivo
    foreach (ClaseRayo Rayo in Rayos)
    {
        //Si encuentra un rayo inactivo
        if (Rayo.getActivo() == false)
        {
            //Activa el rayo (para que sea dibujado)
            Rayo.setActivo(true);

            //El rayo inicia en la posición del lanzador
            Rayo.setPosicion(Lanzador.getPosicion());

            //En que dirección y a que velocidad sale el rayo. Hay que tener en cuenta la rotación del lanzador.
            Rayo.setVelocidad(new Vector2((float)Math.Cos(Lanzador.getRotacion()), (float)Math.Sin(Lanzador.getRotacion())) * 4.0f);

            //Rote el sprite del rayo para que coincida con el del lanzador
            Rayo.setRotacion(Lanzador.getRotacion());

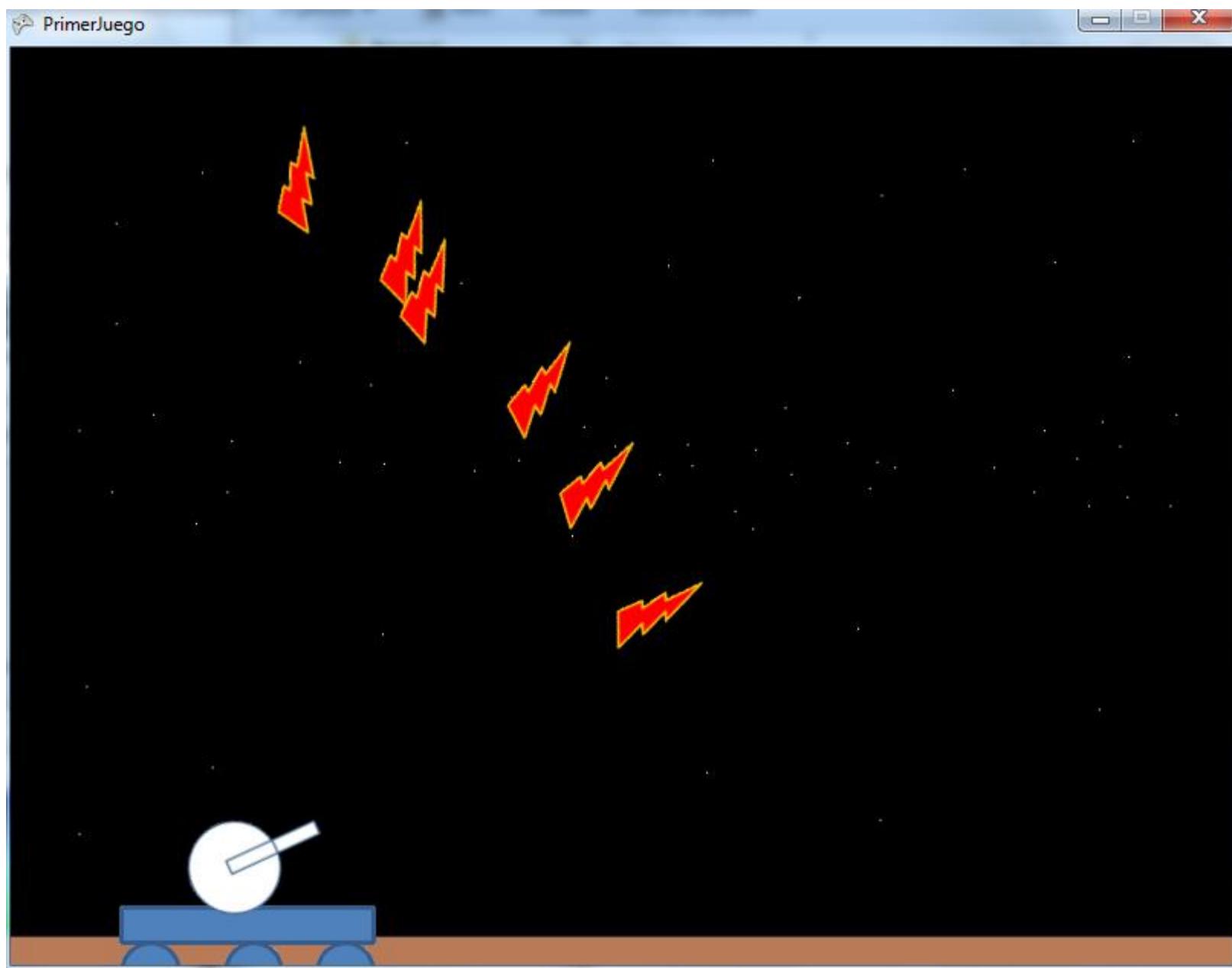
            return;
        }
    }
}

//Método llamado desde Update
public void ActualizaRayos()
{
    //Va de rayo en rayo
    foreach (ClaseRayo Rayo in Rayos)
        if (Rayo.getActivo() == true)
        {
            //Solo es actualizar la velocidad porque el vector se actualiza gracias a esa magnitud
            Rayo.setPosicion(Rayo.getPosicion() + Rayo.getVelocidad());

            //Si el rayo se sale de la pantalla entonces se desactiva
            if (RectFondo.Contains(new Point((int)Rayo.getPosicion().X, (int)Rayo.getPosicion().Y)) == false)
                Rayo.setActivo(false);
        }
}
}

```

Este es el resultado, cada vez que se presiona la tecla espaciadora se lanza un rayo hasta MAXRAYOS



**Figura 109: Se dispara un rayo cada vez que el usuario presiona la tecla espaciadora**

## 23. XNA: El "enemigo" a batir

El objetivo del juego es destruir chatarra espacial, ese es el "enemigo" a batir. Similar a lo hecho con el lanzador, buscamos una imagen representativa de chatarra espacial en Internet y le hacemos su tratamiento respectivo con Paint .NET

**Paso 1:** Procesamos la imagen que represente la chatarra espacial usando Paint .Net

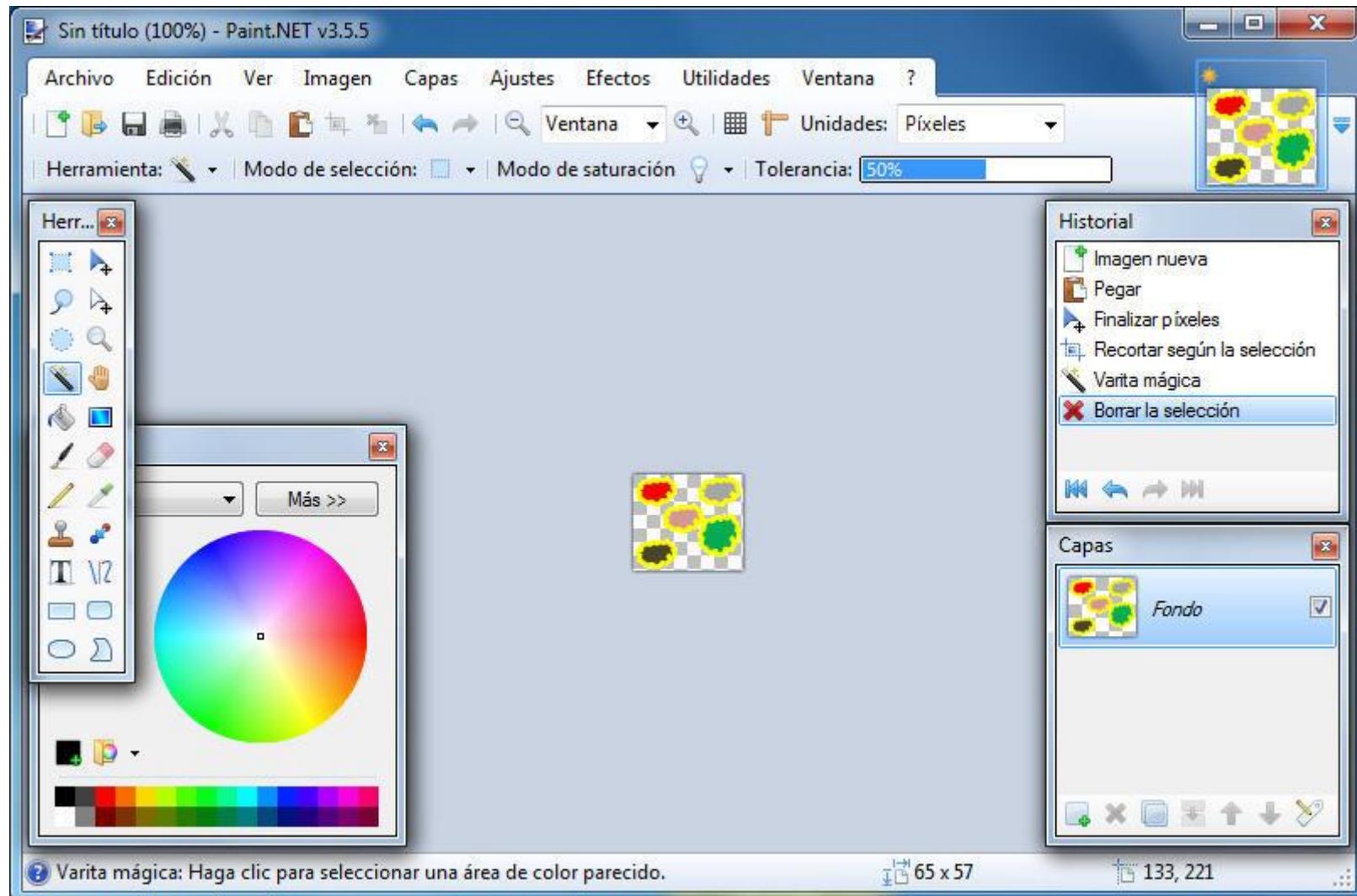
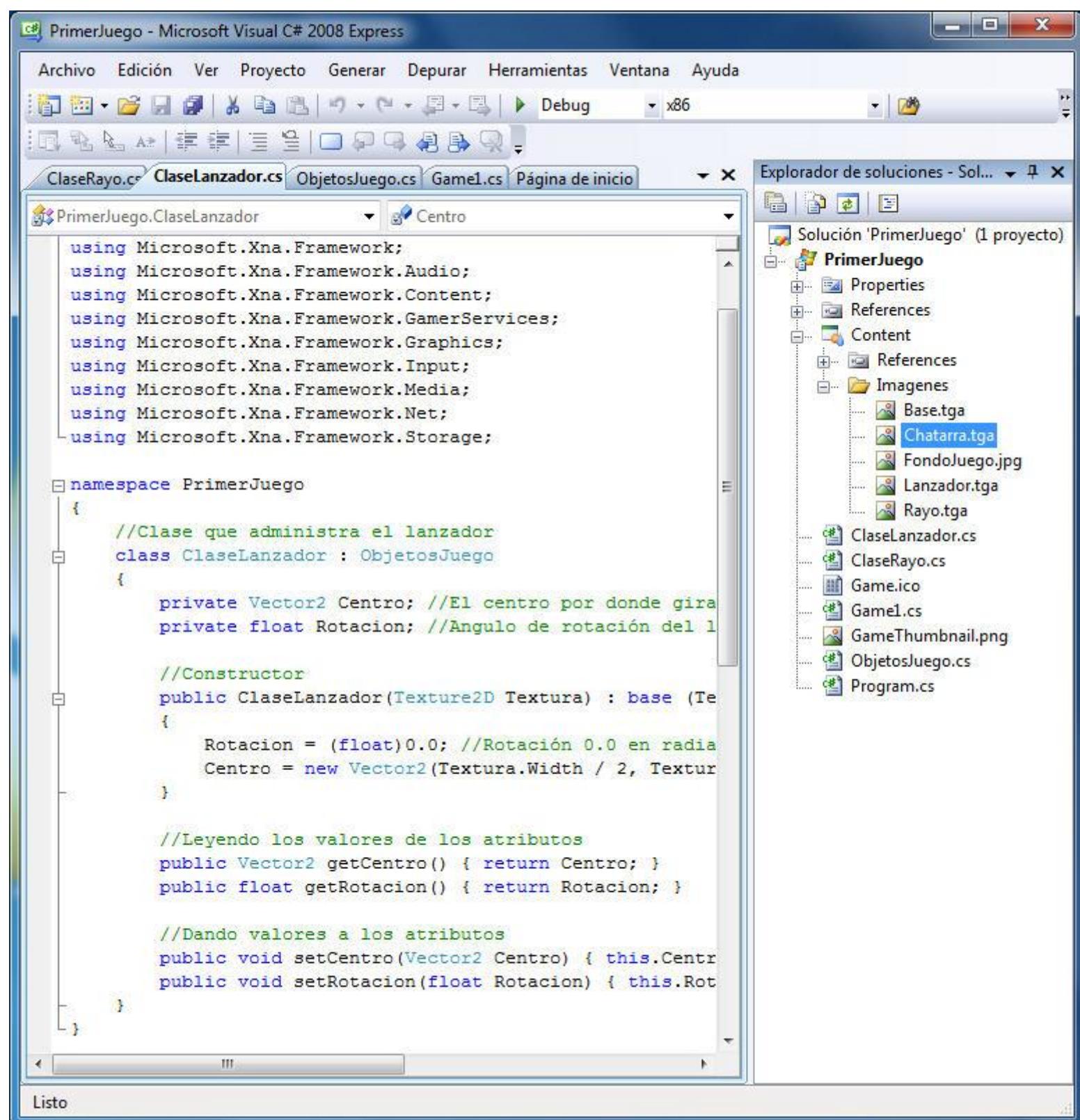


Figura 110: Tratamiento a la imagen que representa al enemigo en el juego.

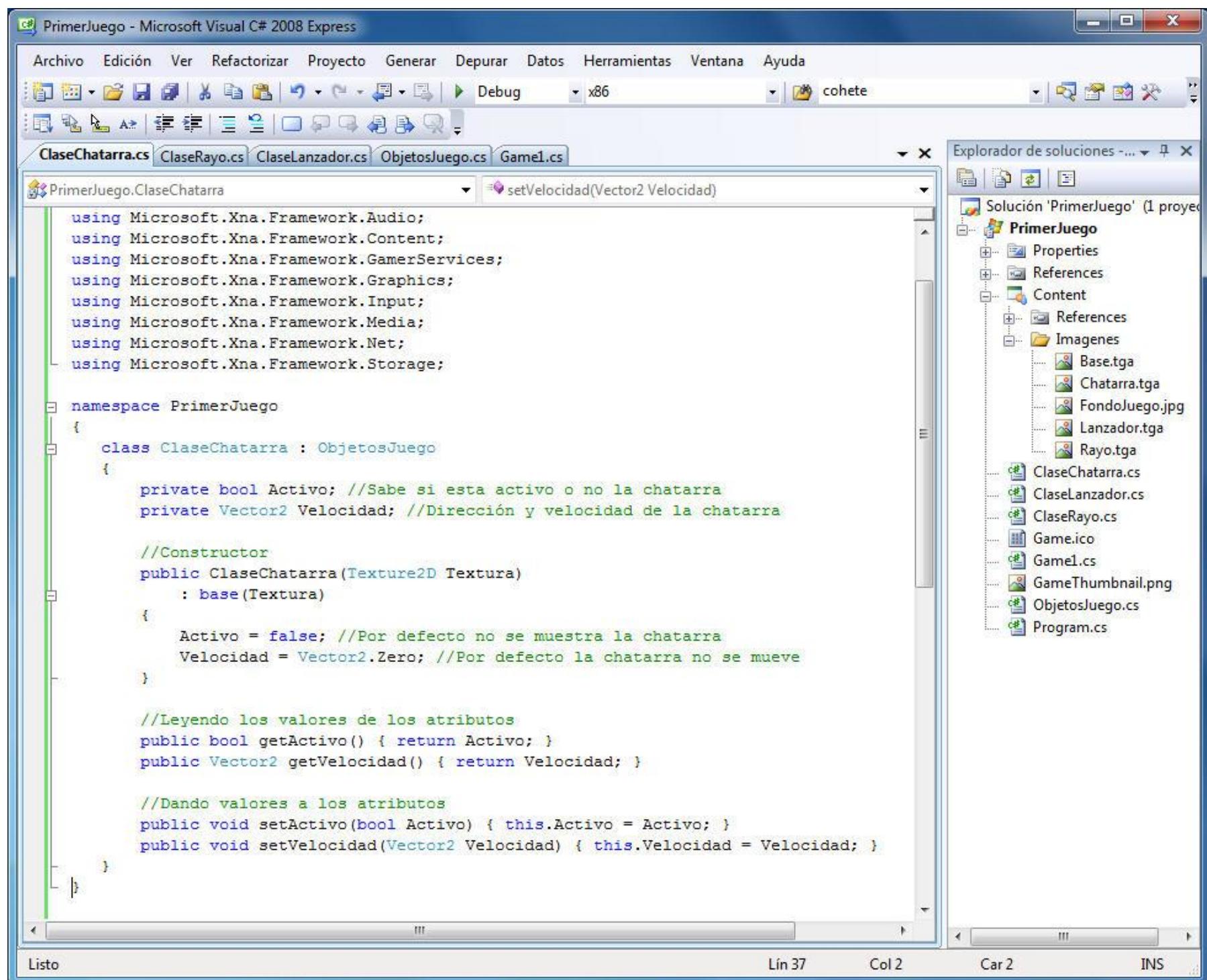
**Paso 2:** Se añade a la biblioteca de imágenes del proyecto.



**Figura 111: Agrega la imagen del enemigo en los recursos del proyecto**

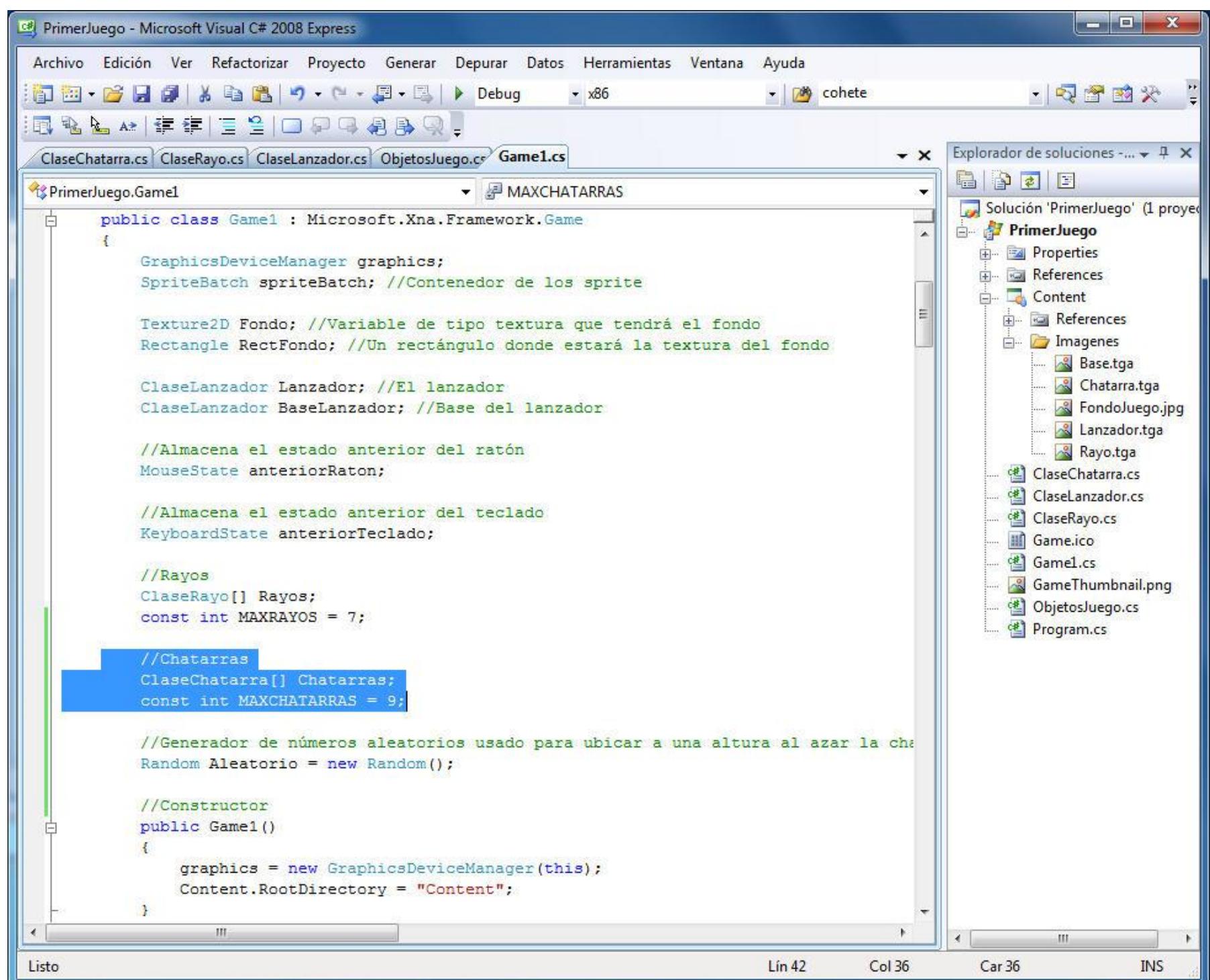
**Paso 3:** La clase que se encargará de la chatarra espacial (enemigo). Cada enemigo tiene una vida limitada: nace en la parte derecha de la ventana, viaja por la ventana y desaparece cuando llega al borde izquierdo de esta. Luego es necesario saber cuando el enemigo se mantiene activo, es decir, cuando se muestra por pantalla. Para eso hacemos uso del atributo de la clase ClaseChatarra de tipo booleano que cuando tiene el valor de "true" significa que el enemigo está activo y debe mostrarse, caso contrario, mantenerlo oculto.

No solamente eso, también hay algo importante con los enemigos: se desplazan por la pantalla, el desplazamiento tiene una dirección y una velocidad, por lo tanto hacemos uso del atributo de velocidad (magnitud y dirección) de la clase ClaseChatarra.



**Figura 112: Código adicionado para el tratamiento del enemigo**

**Paso 4:** Y se cargan las imágenes de ese enemigo en el arreglo unidimensional de objetos.



**Figura 113: Se hace uso de arreglos unidimensionales de objetos para el manejo del enemigo**

Paso 5: Ahora viene la programación del enemigo como tal y estas son las cosas que hay que tener en cuenta:

1. El enemigo aparece a la derecha de la ventana en una posición Y aleatoria.
  2. El enemigo viaja a una velocidad inicial aleatoria.
  3. El enemigo se desplaza de derecha a izquierda, desaparece cuando llegue al borde izquierdo de la ventana.
- Para lo aleatorio necesitamos la clase Random();

```
//Generador de números aleatorios usado para ubicar a una altura al azar la chatarra
Random Aleatorio = new Random();
```

Como el enemigo sale en una posición Y aleatoria, debemos saber entre que rango es el aleatorio (una interpolación lineal entre un Ymin y un Ymax), lo mismo pasa con la velocidad (entre un Vmin y Vmax). Usamos estas variables en el método que actualiza la chatarra.

```
float AlturaMax = 0.1f; //No pegado del techo
float AlturaMin = 0.5f; //La mitad de la pantalla
float VelocidadMax = 5.0f; //Máxima velocidad
float VelocidadMin = 1.0f; //Mínima velocidad
```

El uso de estas variables se da en estas instrucciones:

MathHelper.Lerp es una interpolación lineal entre dos valores. Mas información aqui: <http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.mathhelper.lerp.aspx>

El código que actualiza los enemigos es similar al de los rayos: posición del enemigo mientras este se mantenga activo, este es el método:

```
//Método llamado desde Update
public void ActualizaEnemigos()
{
    float AlturaMax = 0.1f; //No pegado del techo
    float AlturaMin = 0.5f; //La mitad de la pantalla
    float VelocidadMax = 5.0f; //Máxima velocidad
    float VelocidadMin = 1.0f; //Mínima velocidad

    foreach (ClaseChatarra Chatarra in Chatarras)
    {
        if (Chatarra.getActivo() == true)
        {
            //El enemigo avanza a determinada velocidad
            Chatarra.setPosicion(Chatarra.getPosicion() + Chatarra.getVelocidad());
        }
    }
}
```

```

if (RectFondo.Contains(new Point((int) Chatarra.getPosicion().X, (int) Chatarra.getPosicion().Y)) == false)
    Chatarra.setActivo(false);
}
else
{
    //Activa el enemigo
    Chatarra.setActivo(true);

    //Interpolación. Posición eje Y aleatoria.
    Chatarra.setPosicion(new Vector2(RectFondo.Right, MathHelper.Lerp((float)RectFondo.Height * AlturaMin,
    (float)RectFondo.Height * AlturaMax, (float) Aleatorio.NextDouble())));
}

//Velocidad del enemigo. Aleatoria.
Chatarra.setVelocidad(new Vector2(MathHelper.Lerp(-VelocidadMin, -VelocidadMax, (float) Aleatorio.NextDouble()), 0));
}
}
}

```

No olvidar dibujar los enemigos en Draw()

```

//Dibuja cada enemigo
foreach (ClaseChatarra Chatarra in Chatarras)
    if (Chatarra.getActivo() == true)
        spriteBatch.Draw(Chatarra.getTextura(), Chatarra.getPosicion(), Color.White);

```

Y llamar la rutina de actualización en Update()

```

//Actualiza los enemigos
ActualizaEnemigos();

```

El código completo de la clase Game1.cs es el siguiente:

```

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch; //Contenedor de los sprite

        Texture2D Fondo; //Variable de tipo textura que tendrá el fondo
        Rectangle RectFondo; //Un rectángulo donde estará la textura del fondo

        ClaseLanzador Lanzador; //El lanzador
        ClaseLanzador BaseLanzador; //Base del lanzador

        //Almacena el estado anterior del ratón
        MouseState anteriorRaton;

        //Almacena el estado anterior del teclado
        KeyboardState anteriorTeclado;

        //Rayos
        ClaseRayo[] Rayos;
        const int MAXRAYOS = 7;

        //Chatarras
        ClaseChatarra[] Chatarras;
        const int MAXCHATARRAS = 9;

        //Generador de números aleatorios usado para ubicar a una altura al azar la chatarra
        Random Aleatorio = new Random();

        //Constructor
        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
    }
}

```

```

/// and initialize them as well.
/// </summary>
protected override void Initialize()
{
    // TODO: Add your initialization logic here
    base.Initialize();
}

/// <summary>
/// LoadContent will be called once per game and is the place to load
/// all of your content.
/// </summary>
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here

    //Utiliza genéricos <> por eso el uso de <> En C++ se llaman plantillas. Content carga el contenido binario.
    Fondo = Content.Load<Texture2D>("Imagenes\\FondoJuego");

    //El rectángulo en el que estará contenido el fondo
    RectFondo = new Rectangle(0, 0, graphics.GraphicsDevice.Viewport.Width, graphics.GraphicsDevice.Viewport.Height);

    //Cargar e inicializar el Lanzador
    Lanzador = new ClaseLanzador(Content.Load<Texture2D>("Imagenes\\Lanzador")); //No es necesaria la extensión .tga
    Lanzador.setPosicion(new Vector2(160, 530)); //Posición del lanzador

    //Cargar e inicializar la base
    BaseLanzador = new ClaseLanzador(Content.Load<Texture2D>("Imagenes\\Base")); //No es necesaria la extensión .tga
    BaseLanzador.setPosicion(new Vector2(70, 560)); //Posición de la base del lanzador

    //Cargar e inicializar los rayos
    Rayos = new ClaseRayo[MAXRAYOS];
    for (int Cont = 0; Cont < MAXRAYOS; Cont++)
        Rayos[Cont] = new ClaseRayo(Content.Load<Texture2D>("Imagenes\\Rayo"));

    //Cargar e inicializar los chatarras
    Chatarras = new ClaseChatarra[MAXCHATARRAS];
    for (int Cont = 0; Cont < MAXCHATARRAS; Cont++)
        Chatarras[Cont] = new ClaseChatarra(Content.Load<Texture2D>("Imagenes\\Chatarra"));

}

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // Código para leer el estado del Gamepad de una XBOX. Se considera que hay un jugador al menos.
    GamePadState EstadoControl = GamePad.GetState(PlayerIndex.One);

    //Como hay que tener en cuenta los controles del XBOX llamados thumbstick, se
    //observa que botón del XBOX oprimió, eso genera una constante algo grande por lo que se multiplica por 0.1
    Lanzador.setRotacion(Lanzador.getRotacion() + EstadoControl.ThumbSticks.Left.X * 0.1f);

    //Esta es una compilación condicional. El teclado y el mouse solo son leídos si la aplicación ejecuta en un PC
#if !XBOX
    //Código para leer el teclado
    KeyboardState estadoTeclado = Keyboard.GetState();
    if (estadoTeclado.IsKeyDown(Keys.Left)) Lanzador.setRotacion(Lanzador.getRotacion() - 0.1f); //Gira el lanzador hacia la
    izquierda
    if (estadoTeclado.IsKeyDown(Keys.Right)) Lanzador.setRotacion(Lanzador.getRotacion() + 0.1f); //Gira el lanzador hacia la
    derecha

    //Código para leer el ratón
    MouseState Raton = Mouse.GetState();

    //Si movió el ratón entonces el lanzador cambia de ángulo
    if (Raton != anteriorRaton)
    {
        int DiferX = Raton.X - anteriorRaton.X; //La diferencia entre la anterior posición del ratón y la nueva.
        Lanzador.setRotacion(Lanzador.getRotacion() + (float)DiferX / 100); //El movimiento en X del ratón cambia el ángulo del
        lanzador
    }

    //Se actualiza el estado del ratón
    anteriorRaton = Raton;
}

```

```

//Dispara un solo rayo.
//Chequea si el usuario mantiene presionada la tecla de disparo por lo que hace caso omiso a eso,
//el usuario debe presionar y soltar la tecla de disparo para lanzar mas rayos.
if (estadoTeclado.IsKeyDown(Keys.Space) && anteriorTeclado.IsKeyUp(Keys.Space))
    DispararRayo();

anteriorTeclado = estadoTeclado;
#endif

//Esta instrucción limita entre 0 y 90 grados el giro del lanzador. Recordar que se hace uso de radianes.
Lanzador.setRotacion( MathHelper.Clamp(Lanzador.getRotacion(), -MathHelper.PiOver2, 0));

//Actualiza los rayos
ActualizaRayos();

//Actualiza los enemigos
ActualizaEnemigos();

// TODO: Add your update logic here
base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here

    //Se inicia el spritebatch (Entre Begin y End se dibuja en cada unidad de tiempo los sprites)
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

    //Dibujar el fondo combinando el color blanco con el fondo
    spriteBatch.Draw(Fondo, RectFondo, Color.White);

    //Dibuja la base del lanzador combinando con blanco
    spriteBatch.Draw(BaseLanzador.getTextura(), BaseLanzador.getPosicion(), Color.White);

    /* Dibuja el lanzador como tal, estos son los parámetros
     * Parámetro 1: El sprite del lanzador
     * Parámetro 2: La posición del lanzador
     * Parámetro 3: null porque es solo una imagen (no una sucesión de estas)
     * Parámetro 4: Color.White, combinación de color
     * Parámetro 5: Angulo de rotación por defecto
     * Parámetro 6: Centro, el centro del sprite para girarlo después
     * Parámetro 7: Escala a dibujar
     * Parametro 8: Dibujar tal como es el sprite (no reflejo vertical ni horizontal)
     * Parámetro 9: Ponerlo en la capa superior */
    spriteBatch.Draw(Lanzador.getTextura(), Lanzador.getPosicion(), null, Color.White, Lanzador.getRotacion(),
Lanzador.getCentro(), 1f, SpriteEffects.None, 0);

    //Dibuja cada rayo
    foreach (ClaseRayo Rayo in Rayos)
        if (Rayo.getActivo() == true)
            spriteBatch.Draw(Rayo.getTextura(), Rayo.getPosicion(), null, Color.White, Rayo.getRotacion(), Rayo.getCentro(), 1f,
SpriteEffects.None, 0);

    //Dibuja cada enemigo
    foreach (ClaseChatarra Chatarra in Chatarras)
        if (Chatarra.getActivo() == true)
            spriteBatch.Draw(Chatarra.getTextura(), Chatarra.getPosicion(), Color.White);

    //Finaliza el "ciclo" de los sprite
    spriteBatch.End();

    base.Draw(gameTime);
}

//Metodo llamado desde Update
public void DispararRayo()
{
    //Busca un rayo inactivo
    foreach (ClaseRayo Rayo in Rayos)
    {
        //Si encuentra un rayo inactivo
        if (Rayo.getActivo() == false)
        {
            //Activa el rayo (para que sea dibujado)
            Rayo.setActivo(true);

            //El rayo inicia en la posición del lanzador
            Rayo.setPosicion(Lanzador.getPosicion());

            //En que dirección y a que velocidad sale el rayo. Hay que tener en cuenta la rotación del lanzador.
            Rayo.setVelocidad(new Vector2((float)Math.Cos(Lanzador.getRotacion()), (float)Math.Sin(Lanzador.getRotacion())) * 4.0f);

            //Rote el sprite del rayo para que coincida con el del lanzador
            Rayo.setRotacion(Lanzador.getRotacion());
        }
    }
}

```

```

        return;
    }
}

//Método llamado desde Update
public void ActualizaRayos()
{
    //Va de rayo en rayo
    foreach (ClaseRayo Rayo in Rayos)
    {
        //Solo es actualizar la velocidad porque el vector se actualiza gracias a esa magnitud
        Rayo.setPosicion(Rayo.getPosicion() + Rayo.getVelocidad());

        //Si el rayo se sale de la pantalla entonces se desactiva
        if (RectFondo.Contains(new Point((int)Rayo.getPosicion().X, (int)Rayo.getPosicion().Y)) == false)
            Rayo.setActivo(false);
    }
}

//Método llamado desde Update
public void ActualizaEnemigos()
{
    float AlturaMax = 0.1f; //No pegado del techo
    float AlturaMin = 0.5f; //La mitad de la pantalla
    float VelocidadMax = 5.0f; //Máxima velocidad
    float VelocidadMin = 1.0f; //Mínima velocidad

    foreach (ClaseChatarra Chatarra in Chatarras)
    {
        if (Chatarra.getActivo() == true)
        {
            //El enemigo avanza a determinada velocidad
            Chatarra.setPosicion(Chatarra.getPosicion() + Chatarra.getVelocidad());

            //Si el enemigo se sale de la pantalla
            if (RectFondo.Contains(new Point((int) Chatarra.getPosicion().X, (int) Chatarra.getPosicion().Y)) == false)
                Chatarra.setActivo(false);
        }
        else
        {
            //Activa el enemigo
            Chatarra.setActivo(true);

            //Interpolación. Posición eje Y aleatoria.
            Chatarra.setPosicion(new Vector2(RectFondo.Right, MathHelper.Lerp((float)RectFondo.Height * AlturaMin,
(flat)RectFondo.Height * AlturaMax, (float) Aleatorio.NextDouble())));
        }
    }
}
}

```

Este es el resultado:

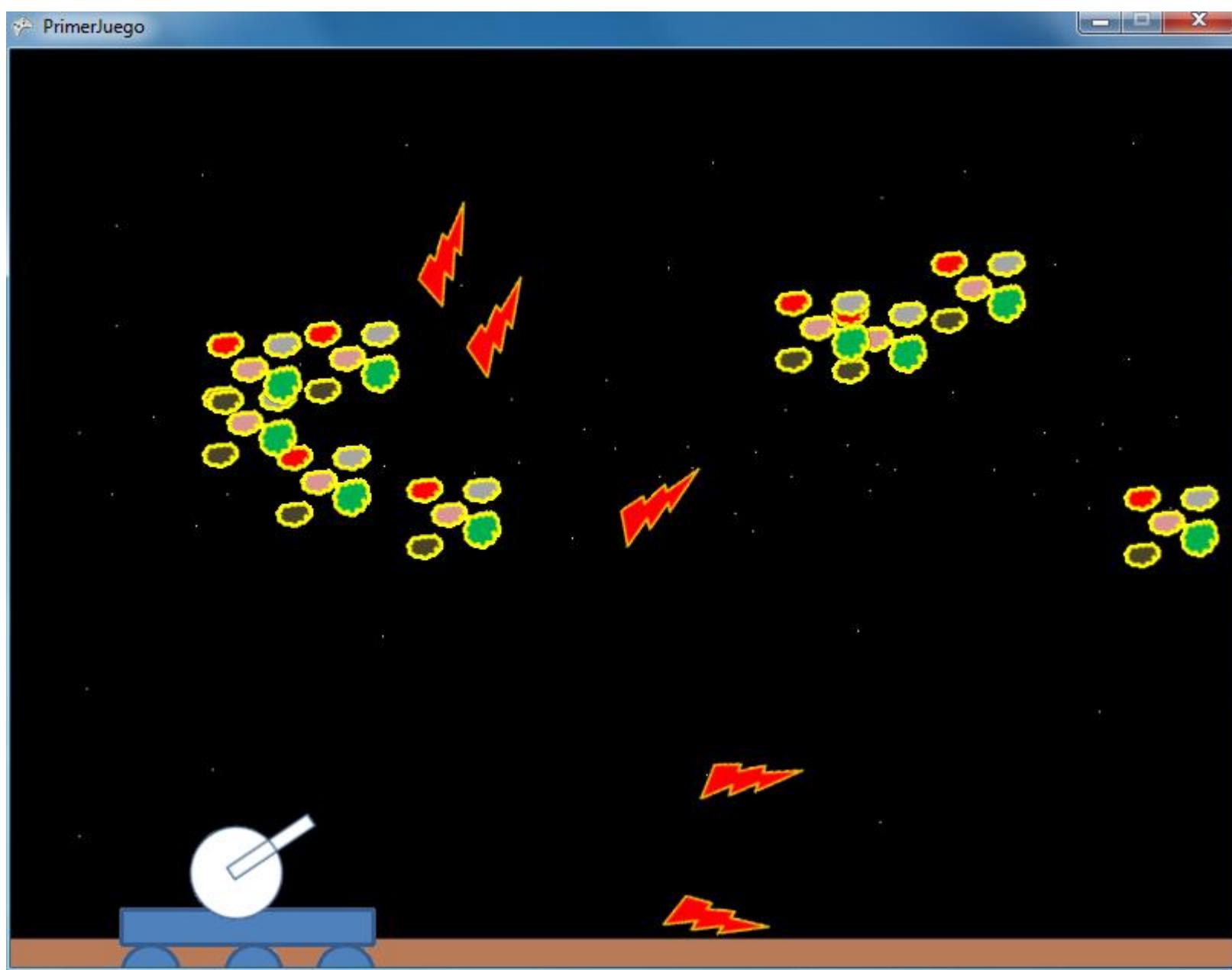


Figura 114: El juego ya tiene adicionado el objeto enemigo

## 24. XNA: Colisiones. Haciendo que el rayo golpee al enemigo

Hasta ahora si disparamos los cohetes, estos siguen, pueden pasar por encima del enemigo y no sucede nada. El juego no tiene emoción. Debemos añadirle que cuando un cohete acierte al enemigo, ambos desaparezcan. Eso se le conoce en el medio con el nombre de "colisión". Pero ¿Cómo se hace la colisión?. Un primer acercamiento es comparar las coordenadas del cohete, con las coordenadas del enemigo, si coinciden, hay una colisión. Sin embargo, lograr tal coincidencia es muy difícil durante el juego. La comparación es entonces por áreas rectangulares, se compara si el área rectangular del cohete toca el área rectangular del enemigo. Eso es mucho mas posible. XNA nos ofrece la función para comparar si dos rectángulos se interceptan.

El método que actualiza los cohetes en Game1.cs tiene ahora este código (no se modifica ningún otro código):

```
//Método llamado desde Update
public void ActualizaRayos()
{
    //Va de rayo en rayo
    foreach (ClaseRayo Rayo in Rayos)
        if (Rayo.getActivo() == true)
    {
        //Solo es actualizar la velocidad porque el vector se actualiza gracias a esa magnitud
        Rayo.setPosicion(Rayo.getPosicion() + Rayo.getVelocidad());

        //Chequea la colisión

        //Deduce el rectángulo del cohete y debe tener en cuenta el tamaño con que se esté dibujando en pantalla en Draw()
        Rectangle RectRayo = new Rectangle((int)Rayo.getPosicion().X, (int)Rayo.getPosicion().Y, (int)(Rayo.getTextura().Width * 0.2), (int)(Rayo.getTextura().Height * 0.2));

        //Chequea de enemigo en enemigo si el cohete lo colisiona
        foreach (ClaseChatarra Chatarra in Chatarras)
        {
            //Deduce el rectángulo del enemigo y debe tener en cuenta el tamaño con que se esté dibujando en pantalla en Draw()
            Rectangle RectEnemigo = new Rectangle((int)Chatarra.getPosicion().X, (int)Chatarra.getPosicion().Y, (int)(Chatarra.getTextura().Width), (int)(Chatarra.getTextura().Height));

            //Si ambos rectángulos se intersectan
            if (RectRayo.Intersects(RectEnemigo) == true)
            {
                Chatarra.setActivo(false); //inactiva el enemigo
                Rayo.setActivo(false); //inactiva el rayo
                break;
            }
        }

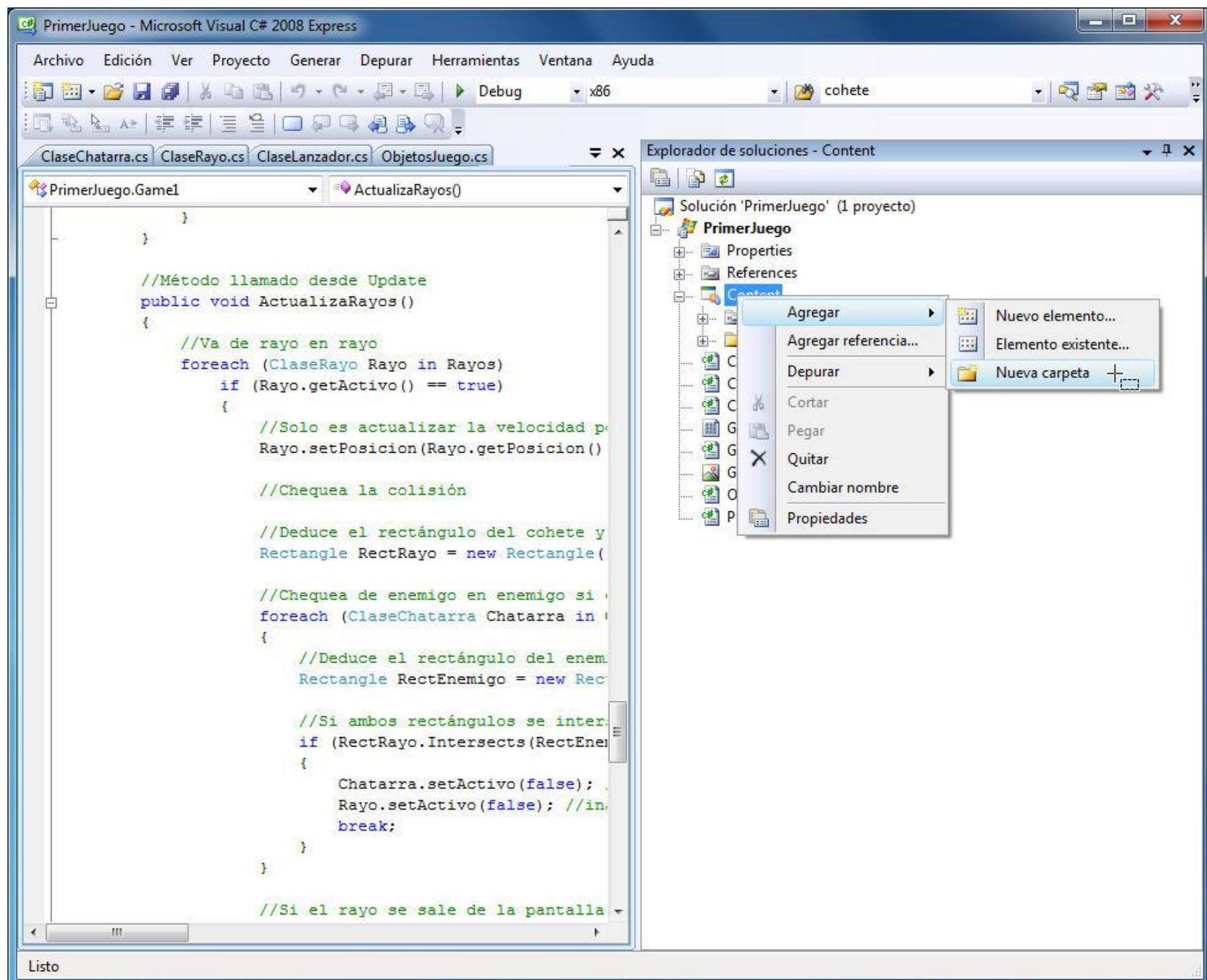
        //Si el rayo se sale de la pantalla entonces se desactiva
        if (RectFondo.Contains(new Point((int)Rayo.getPosicion().X, (int)Rayo.getPosicion().Y)) == false)
            Rayo.setActivo(false);
    }
}
```

## 25. XNA: Mostrando el puntaje

Necesitamos mostrar al jugador cuantos satélites chatarra espacial va destruyendo a medida que juega. Para eso debemos primero tener una variable que cuente cuantos satélites destruye (se incrementa de uno en uno cuando el cohete intercepta al enemigo), y además hay que mostrar ese puntaje en pantalla.

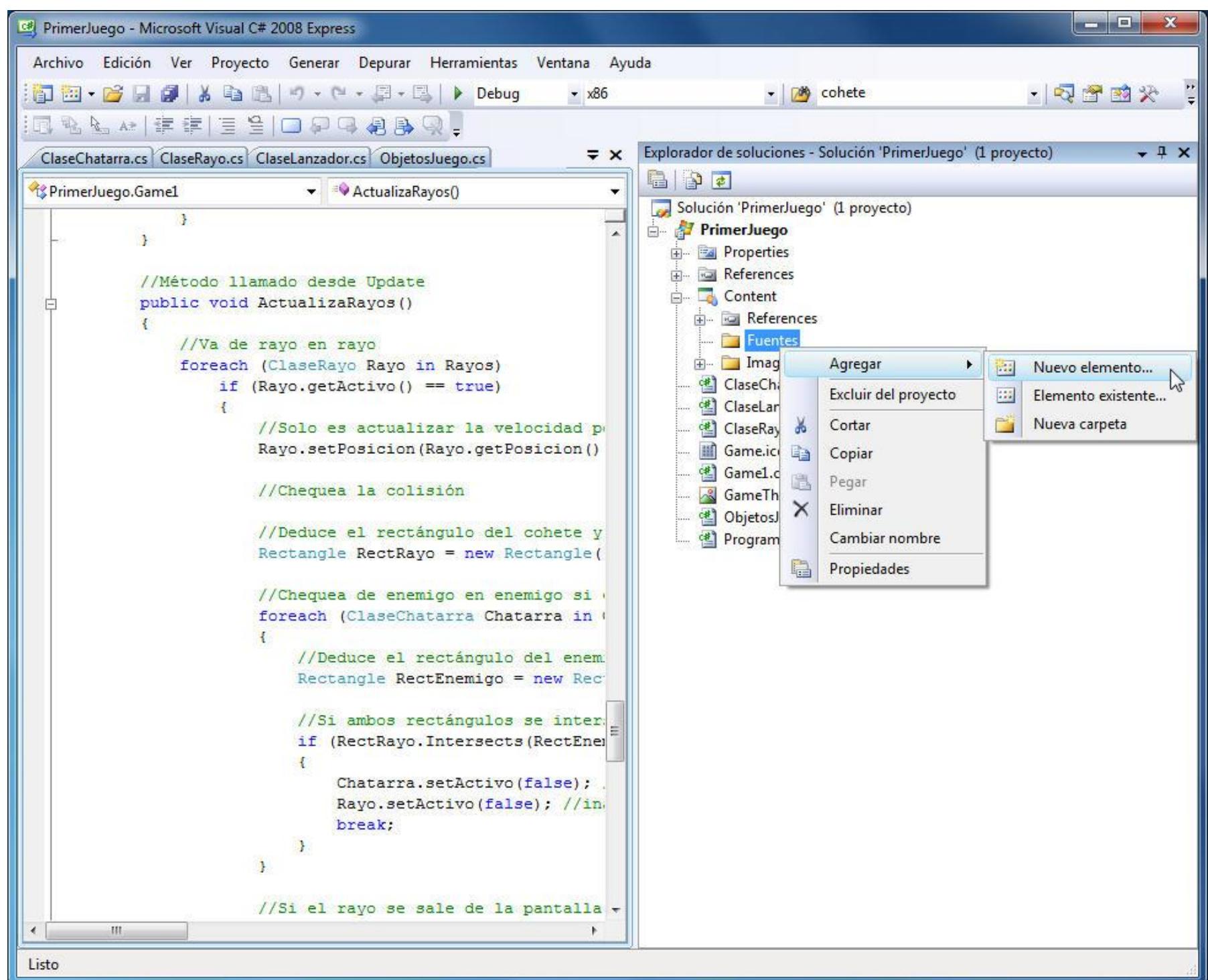
Paso 1: ¿Qué tipo de letra (fuente) usará para desplegar el puntaje?

De clic en Content en Explorador de soluciones y allí clic botón derecho, seleccione Agregar -> Nueva carpeta



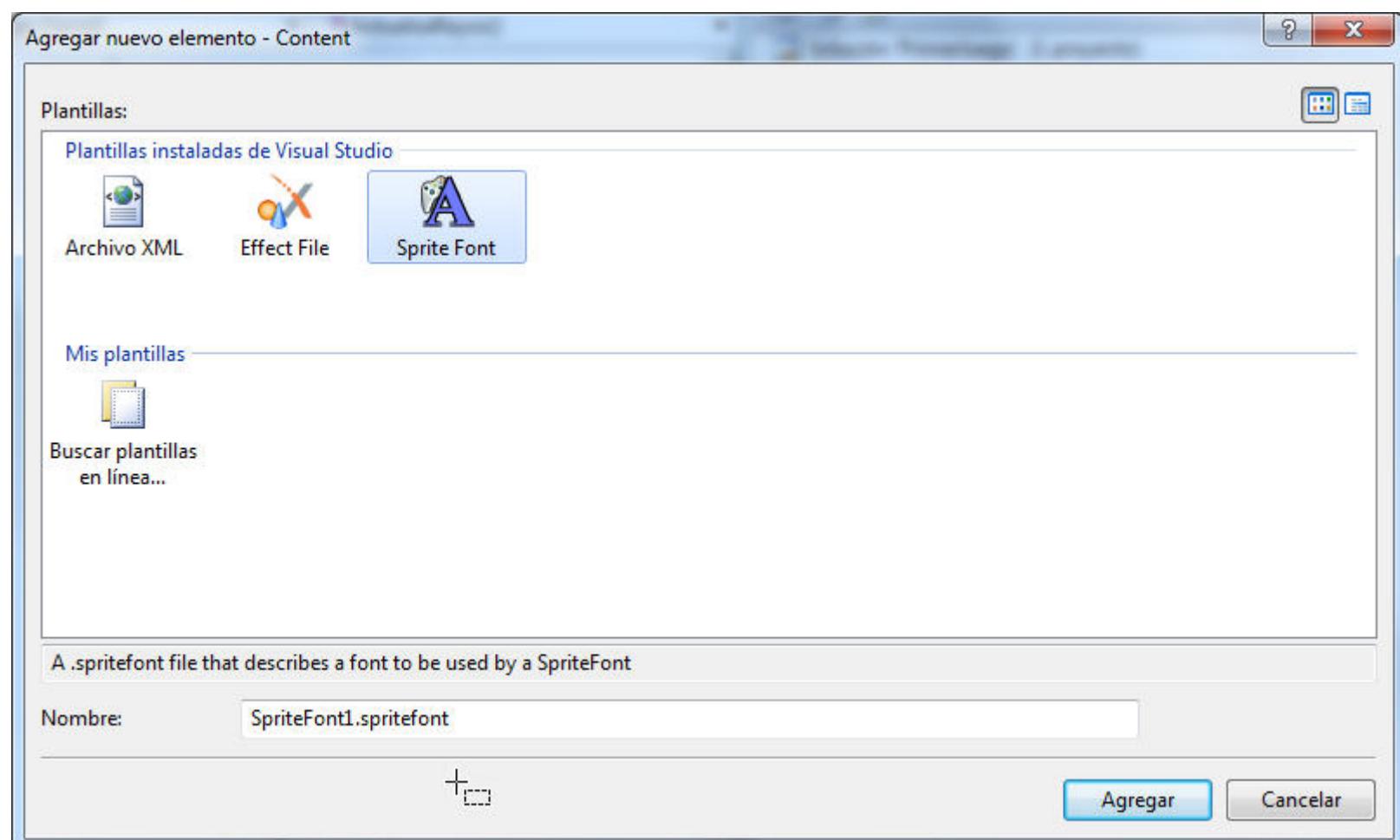
**Figura 115: Se agrega una nueva carpeta a recursos del proyecto**

Esa carpeta llamémosla "Fuentes"



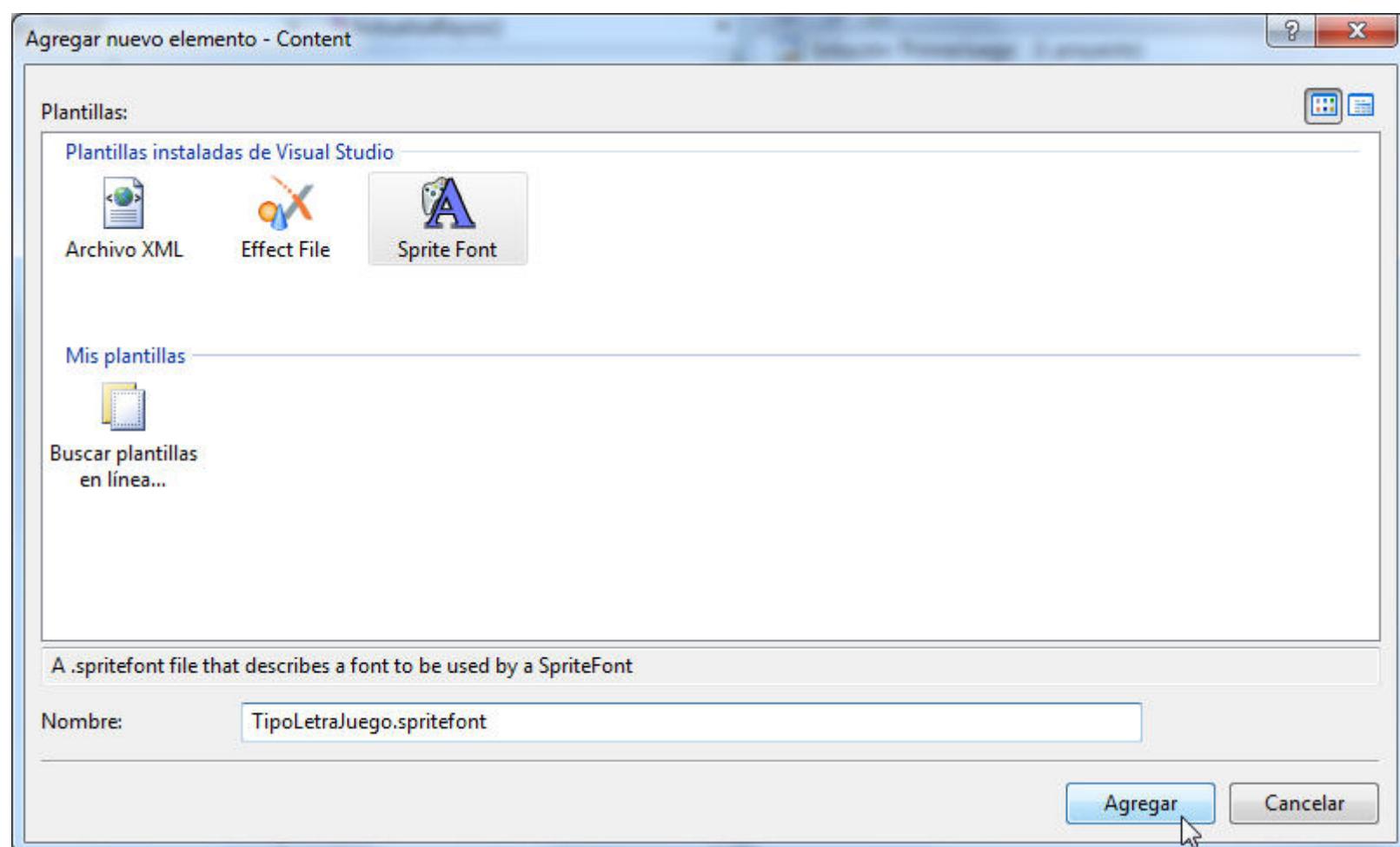
**Figura 116:** Se agrega un nuevo elemento de tipo recurso

Agreguemos un Nuevo elemento a esa carpeta Fuentes



**Figura 117:** Agregando un Sprite Font

El elemento a agregar es un Sprite Font al cual le ponemos el nombre TipoLetraJuego.spritefont (conserva la extensión)



**Figura 118: Dándole un nombre al Sprite Font**

Nos muestra una página XML, la cual personalizamos a nuestro gusto para el tipo de letra

The screenshot shows the Microsoft Visual Studio 2008 Express interface. The title bar reads "PrimerJuego - Microsoft Visual C# 2008 Express". The menu bar includes Archivo, Edición, Ver, Proyecto, Generar, Depurar, XML, Herramientas, Ventana, Ayuda. The toolbar has various icons for file operations. The status bar at the bottom shows Líne1 Col1 Car1 INS.

The main window displays the XML content of the file "TiposLetraJuego.spritefont". The code defines a font description with properties like FontName (Kootenay), Size (14), Spacing (0), and UseKerning (true). The Solution Explorer on the right shows the project structure:

- Solución 'PrimerJuego' (1 proyecto)
  - PrimerJuego
    - Properties
    - References
    - Content
      - References
      - Fuentes
        - TiposLetraJuego.spritefont
      - Imagenes
      - ClaseChatarra.cs
      - ClaseLanzador.cs
      - ClaseRayo.cs
      - Game.ico
      - Game1.cs
      - GameThumbnail.png
      - ObjetosJuego.cs
      - Program.cs

**Figura 119: Configurando el recurso "Tipode Fuente"**

En **<FontName>** hemos cambiado por Verdana y en **<Size>** ponemos 18

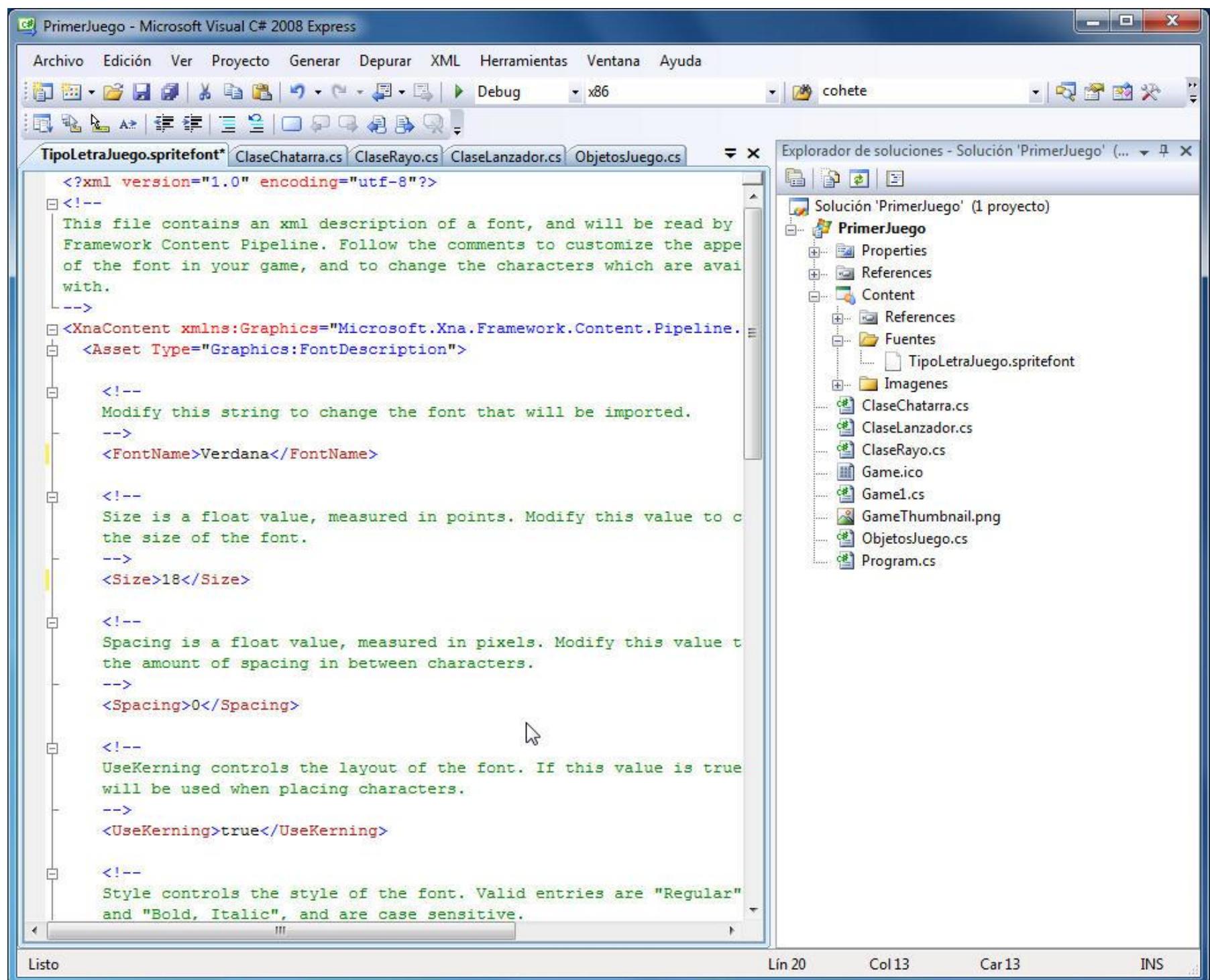


Figura 120: Recurso "Tipo de Fuente" configurado

Los siguientes pasos se hacen sobre la clase Game1.cs

**Paso 2:** Declarar la variable que lleva el puntaje y la variable "tipo de letra" que despliega el valor de puntaje en pantalla

```
//Puntaje
int Puntaje = 0; //Lleva cuantos enemigos han sido destruidos
SpriteFont Fuente; //Variable tipo de letra para mostrar el puntaje
```

**Paso 3:** Cargar la fuente en el método LoadContent

```
//Cargar la fuente
Fuente = Content.Load<SpriteFont>("Fuentes\\TipoLetraJuego");
```

**Paso 4:** Actualizar el puntaje cada vez que se acierte a un enemigo en el método public void ActualizaRayos()

```
Puntaje++;
```

**Paso 5:** Mostrar el puntaje por pantalla en el método Draw

```
//Dibuja el puntaje
spriteBatch.DrawString(Fuente, "Destruido: " + Puntaje.ToString(), new Vector2(80, 60), Color.White);
```

El código completo de Game1.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
```

```

namespace PrimerJuego
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch; //Contenedor de los sprite

        Texture2D Fondo; //Variable de tipo textura que tendrá el fondo
        Rectangle RectFondo; //Un rectángulo donde estará la textura del fondo

        ClaseLanzador Lanzador; //El lanzador
        ClaseLanzador BaseLanzador; //Base del lanzador

        //Almacena el estado anterior del ratón
        MouseState anteriorRaton;

        //Almacena el estado anterior del teclado
        KeyboardState anteriorTeclado;

        //Rayos
        ClaseRayo[] Rayos;
        const int MAXRAYOS = 7;

        //Chatarras
        ClaseChatarra[] Chatarras;
        const int MAXCHATARRAS = 9;

        //Generador de números aleatorios usado para ubicar a una altura al azar la chatarra
        Random Aleatorio = new Random();

        //Puntaje
        int Puntaje = 0; //Lleva cuantos enemigos han sido destruidos
        SpriteFont Fuente; //Variable tipo de letra para mostrar el puntaje

        //Constructor
        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here
            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // TODO: use this.Content to load your game content here

            //Utiliza genéricos <> por eso el uso de <> En C++ se llaman plantillas. Content carga el contenido binario.
            Fondo = Content.Load<Texture2D>("Imagenes\\FondoJuego");

            //El rectángulo en el que estará contenido el fondo
            RectFondo = new Rectangle(0, 0, graphics.GraphicsDevice.Viewport.Width, graphics.GraphicsDevice.Viewport.Height);

            //Cargar e inicializar el Lanzador
            Lanzador = new ClaseLanzador(Content.Load<Texture2D>("Imagenes\\Lanzador")); //No es necesaria la extensión .tga
            Lanzador.setPosicion(new Vector2(160, 530)); //Posición del lanzador

            //Cargar e inicializar la base
            BaseLanzador = new ClaseLanzador(Content.Load<Texture2D>("Imagenes\\Base")); //No es necesaria la extensión .tga
            BaseLanzador.setPosicion(new Vector2(70, 560)); //Posición de la base del lanzador

            //Cargar e inicializar los rayos
            Rayos = new ClaseRayo[MAXRAYOS];
            for (int Cont = 0; Cont < MAXRAYOS; Cont++)
                Rayos[Cont] = new ClaseRayo(Content.Load<Texture2D>("Imagenes\\Rayo"));

            //Cargar e inicializar los rayos
            Chatarras = new ClaseChatarra[MAXCHATARRAS];
            for (int Cont = 0; Cont < MAXCHATARRAS; Cont++)
                Chatarras[Cont] = new ClaseChatarra(Content.Load<Texture2D>("Imagenes\\Chatarra"));

            //Cargar la fuente
        }
    }
}

```

```

Fuente = Content.Load<SpriteFont>("Fuentes\\TipoLetraJuego");
}

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // Código para leer el estado del Gamepad de una XBOX. Se considera que hay un jugador al menos.
    GamePadState EstadoControl = GamePad.GetState(PlayerIndex.One);

    //Como hay que tener en cuenta los controles del XBOX llamados thumbstick, se
    //observa que botón del XBOX oprimió, eso genera una constante algo grande por lo que se multiplica por 0.1
    Lanzador.setRotacion(Lanzador.getRotacion() + EstadoControl.ThumbSticks.Left.X * 0.1f);

    //Esta es una compilación condicional. El teclado y el mouse solo son leídos si la aplicación ejecuta en un PC
#if !XBOX
    //Código para leer el teclado
    KeyboardState estadoTeclado = Keyboard.GetState();
    if (estadoTeclado.IsKeyDown(Keys.Left)) Lanzador.setRotacion(Lanzador.getRotacion() - 0.1f); //Gira el lanzador hacia la
    izquierda
    if (estadoTeclado.IsKeyDown(Keys.Right)) Lanzador.setRotacion(Lanzador.getRotacion() + 0.1f); //Gira el lanzador hacia la
    derecha

    //Código para leer el ratón
    MouseState Raton = Mouse.GetState();

    //Si movió el ratón entonces el lanzador cambia de ángulo
    if (Raton != anteriorRaton)
    {
        int DiferX = Raton.X - anteriorRaton.X; //La diferencia entre la anterior posición del ratón y la nueva.
        Lanzador.setRotacion(Lanzador.getRotacion() + (float)DiferX / 100); //El movimiento en X del ratón cambia el ángulo del
        lanzador
    }

    //Se actualiza el estado del ratón
    anteriorRaton = Raton;

    //Dispara un solo rayo.
    //Chequea si el usuario mantiene presionada la tecla de disparo por lo que hace caso omiso a eso,
    //el usuario debe presionar y soltar la tecla de disparo para lanzar mas rayos.
    if (estadoTeclado.IsKeyDown(Keys.Space) && anteriorTeclado.IsKeyUp(Keys.Space))
        DispararRayo();

    anteriorTeclado = estadoTeclado;
#endif

    //Esta instrucción limita entre 0 y 90 grados el giro del lanzador. Recordar que se hace uso de radianes.
    Lanzador.setRotacion(MathHelper.Clamp(Lanzador.getRotacion(), -MathHelper.PiOver2, 0));

    //Actualiza los rayos
    ActualizaRayos();

    //Actualiza los enemigos
    ActualizaEnemigos();

    // TODO: Add your update logic here
    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here

    //Se inicia el spritebatch (Entre Begin y End se dibuja en cada unidad de tiempo los sprites)
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

    //Dibujar el fondo combinando el color blanco con el fondo
    spriteBatch.Draw(Fondo, RectFondo, Color.White);

    //Dibuja la base del lanzador combinando con blanco
    spriteBatch.Draw(BaseLanzador.getTextura(), BaseLanzador.getPosicion(), Color.White);
}

```

```

/* Dibuja el lanzador como tal, estos son los parámetros
 * Parámetro 1: El sprite del lanzador
 * Parámetro 2: La posición del lanzador
 * Parámetro 3: null porque es solo una imagen (no una sucesión de estas)
 * Parámetro 4: Color.White, combinación de color
 * Parámetro 5: Angulo de rotación por defecto
 * Parámetro 6: Centro, el centro del sprite para girarlo después
 * Parámetro 7: Escala a dibujar
 * Parametro 8: Dibujar tal como es el sprite (no reflejo vertical ni horizontal)
 * Parámetro 9: Ponerlo en la capa superior */
spriteBatch.Draw(Lanzador.getTextura(), Lanzador.getPosicion(), null, Color.White, Lanzador.getRotacion(),
Lanzador.getCentro(), 1f, SpriteEffects.None, 0);

//Dibuja cada rayo
foreach (ClaseRayo Rayo in Rayos)
    if (Rayo.getActivo() == true)
        spriteBatch.Draw(Rayo.getTextura(), Rayo.getPosicion(), null, Color.White, Rayo.getRotacion(), Rayo.getCentro(), 1f,
SpriteEffects.None, 0);

//Dibuja cada enemigo
foreach (ClaseChatarra Chatarra in Chatarras)
    if (Chatarra.getActivo() == true)
        spriteBatch.Draw(Chatarra.getTextura(), Chatarra.getPosicion(), Color.White);

//Dibuja el puntaje
spriteBatch.DrawString(Fuente, "Destruido: " + Puntaje.ToString(), new Vector2(80, 60), Color.White);

//Finaliza el "ciclo" de los sprite
spriteBatch.End();

base.Draw(gameTime);
}

//Metodo llamado desde Update
public void DispararRayo()
{
    //Busca un rayo inactivo
    foreach (ClaseRayo Rayo in Rayos)
    {
        //Si encuentra un rayo inactivo
        if (Rayo.getActivo() == false)
        {
            //Activa el rayo (para que sea dibujado)
            Rayo.setActivo(true);

            //El rayo inicia en la posición del lanzador
            Rayo.setPosicion(Lanzador.getPosicion());

            //En que dirección y a que velocidad sale el rayo. Hay que tener en cuenta la rotación del lanzador.
            Rayo.setVelocidad(new Vector2((float)Math.Cos(Lanzador.getRotacion()), (float)Math.Sin(Lanzador.getRotacion())) * 4.0f);

            //Rote el sprite del rayo para que coincida con el del lanzador
            Rayo.setRotacion(Lanzador.getRotacion());

            return;
        }
    }
}

//Método llamado desde Update
public void ActualizaRayos()
{
    //Va de rayo en rayo
    foreach (ClaseRayo Rayo in Rayos)
        if (Rayo.getActivo() == true)
        {
            //Solo es actualizar la velocidad porque el vector se actualiza gracias a esa magnitud
            Rayo.setPosicion(Rayo.getPosicion() + Rayo.getVelocidad());

            //Chequea la colisión

            //Deduce el rectángulo del cohete y debe tener en cuenta el tamaño con que se esté dibujando en pantalla en Draw()
            Rectangle RectRayo = new Rectangle((int)Rayo.getPosicion().X, (int)Rayo.getPosicion().Y, (int)(Rayo.getTextura().Width * 0.2), (int)(Rayo.getTextura().Height * 0.2));

            //Chequea de enemigo en enemigo si el cohete lo colisiona
            foreach (ClaseChatarra Chatarra in Chatarras)
            {
                //Deduce el rectángulo del enemigo y debe tener en cuenta el tamaño con que se esté dibujando en pantalla en Draw()
                Rectangle RectEnemigo = new Rectangle((int)Chatarra.getPosicion().X, (int)Chatarra.getPosicion().Y, (int)(Chatarra.getTextura().Width), (int)(Chatarra.getTextura().Height));

                //Si ambos rectángulos se intersectan
                if (RectRayo.Intersects(RectEnemigo) == true)
                {
                    Chatarra.setActivo(false); //inactiva el enemigo
                    Rayo.setActivo(false); //inactiva el rayo
                    Puntaje++;
                    break;
                }
            }
        }
}

```

```

//Si el rayo se sale de la pantalla entonces se desactiva
if (RectFondo.Contains(new Point((int)Rayo.getPosicion().X, (int)Rayo.getPosicion().Y)) == false)
    Rayo.setActivo(false);
}

//Método llamado desde Update
public void ActualizaEnemigos()
{
    float AlturaMax = 0.1f; //No pegado del techo
    float AlturaMin = 0.5f; //La mitad de la pantalla
    float VelocidadMax = 5.0f; //Máxima velocidad
    float VelocidadMin = 1.0f; //Mínima velocidad

    foreach (ClaseChatarra Chatarra in Chatarras)
    {
        if (Chatarra.getActivo() == true)
        {
            //El enemigo avanza a determinada velocidad
            Chatarra.setPosicion(Chatarra.getPosicion() + Chatarra.getVelocidad());

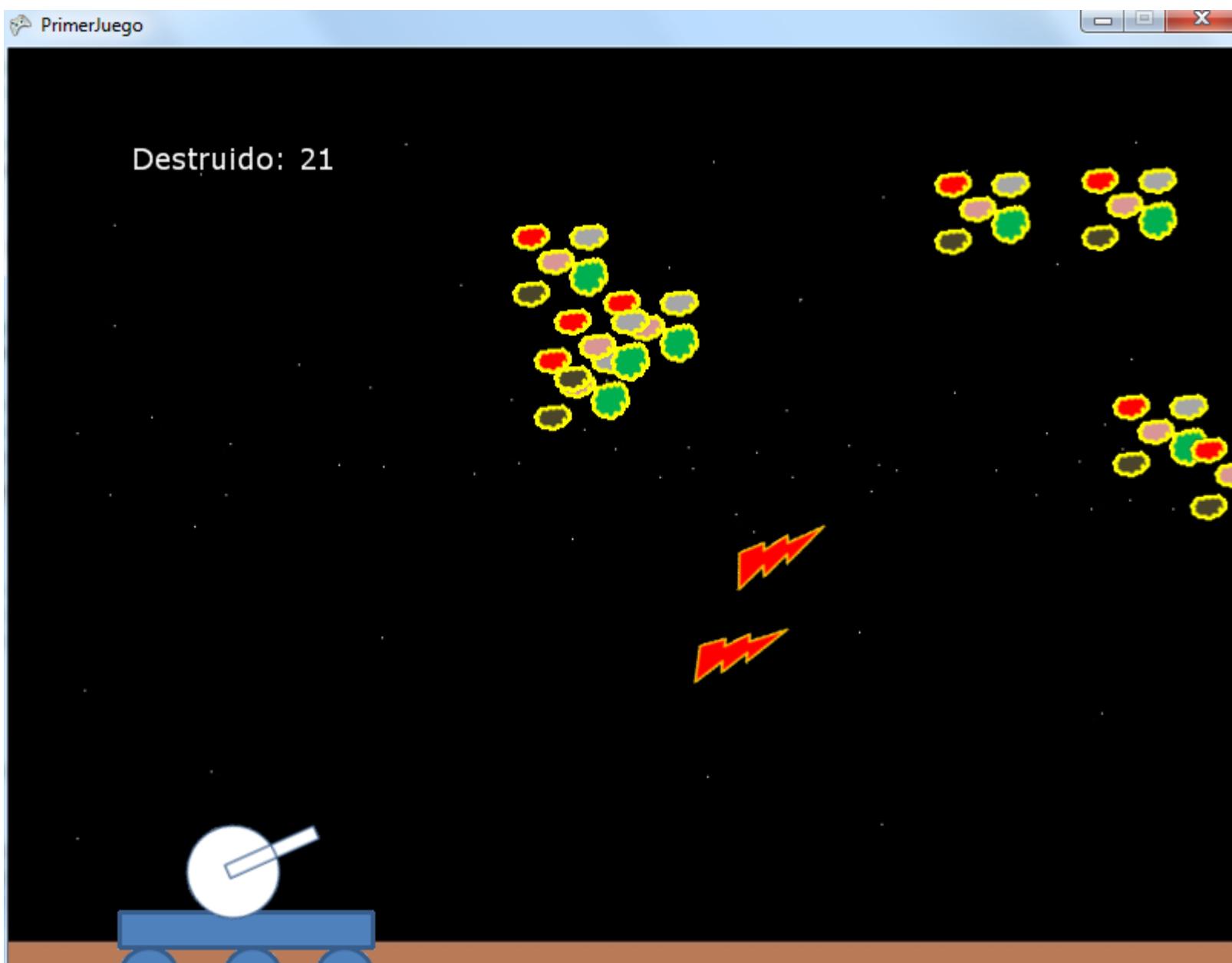
            //Si el enemigo se sale de la pantalla
            if (RectFondo.Contains(new Point((int) Chatarra.getPosicion().X, (int) Chatarra.getPosicion().Y)) == false)
                Chatarra.setActivo(false);
        }
        else
        {
            //Activa el enemigo
            Chatarra.setActivo(true);

            //Interpolación. Posición eje Y aleatoria.
            Chatarra.setPosicion(new Vector2(RectFondo.Right, MathHelper.Lerp((float)RectFondo.Height * AlturaMin,
(flat)RectFondo.Height * AlturaMax, (float) Aleatorio.NextDouble())));

            //Velocidad del enemigo. Aleatoria.
            Chatarra.setVelocidad(new Vector2(MathHelper.Lerp(-VelocidadMin, -VelocidadMax, (float) Aleatorio.NextDouble()), 0));
        }
    }
}

```

Este es el resultado:



**Figura 121:** Mostrando el puntaje en el informe

## 26. XNA: Sonidos en el juego

Hasta ahora nuestro juego es mudo. Es momento de añadirle mas emoción y realismo, y el camino para eso es agregándole sonidos. Estos son los pasos:

**Paso 1:** Buscar en Internet sonidos o generar sus propios sonidos y guardarlos en archivos con extensión .wav



Figura 122: Archivos de sonido

**Paso 2:** Ejecutar el Microsoft Cross-Platform Audio Creation Tool 3 (XACT3) y crear un nuevo proyecto

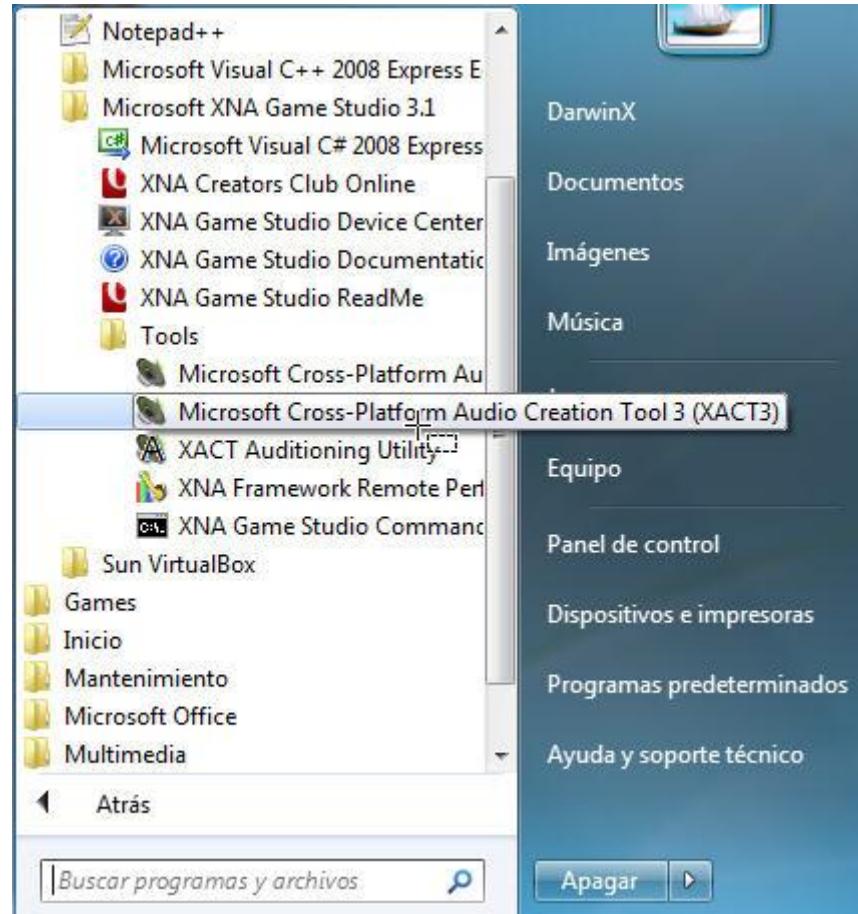
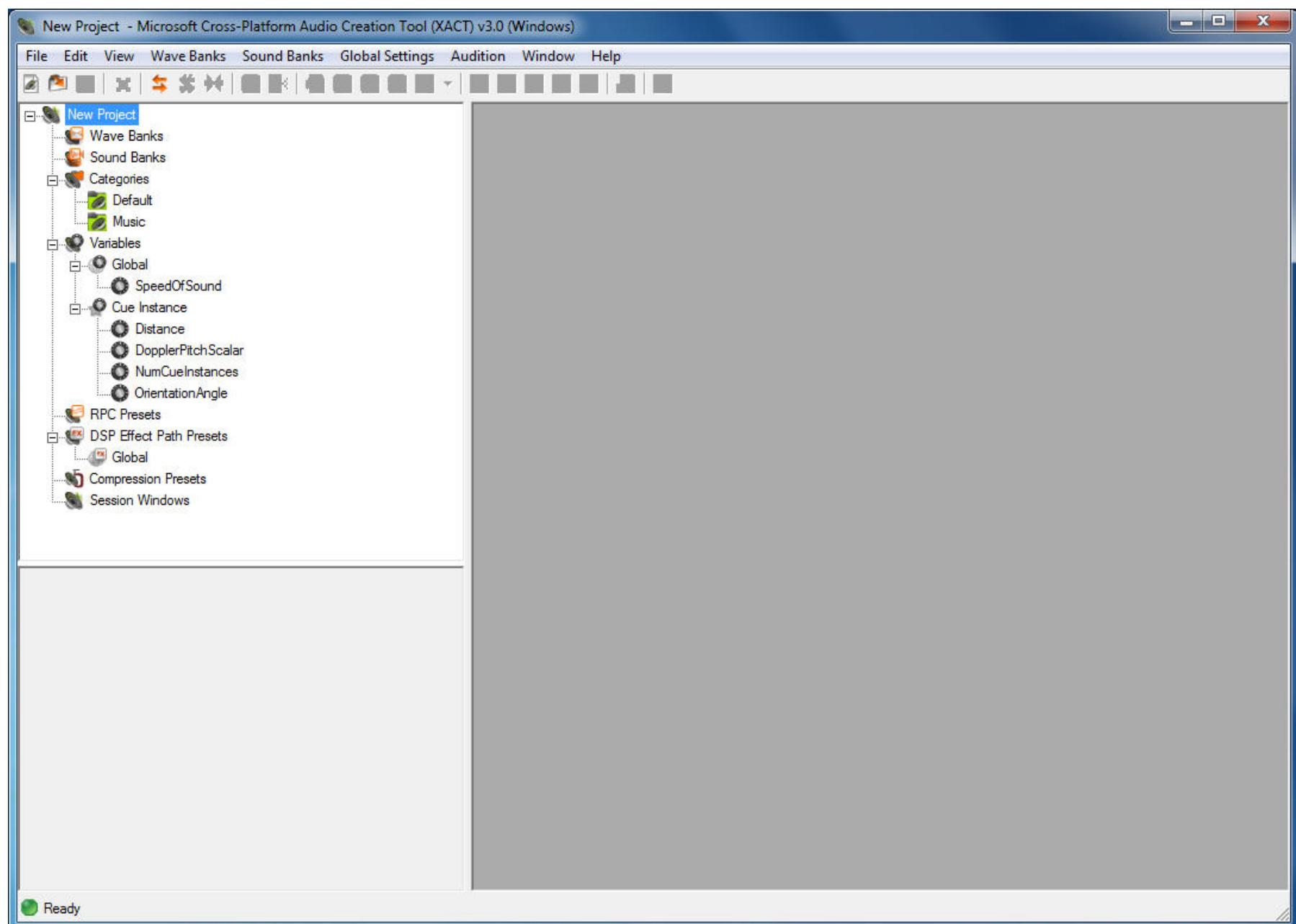
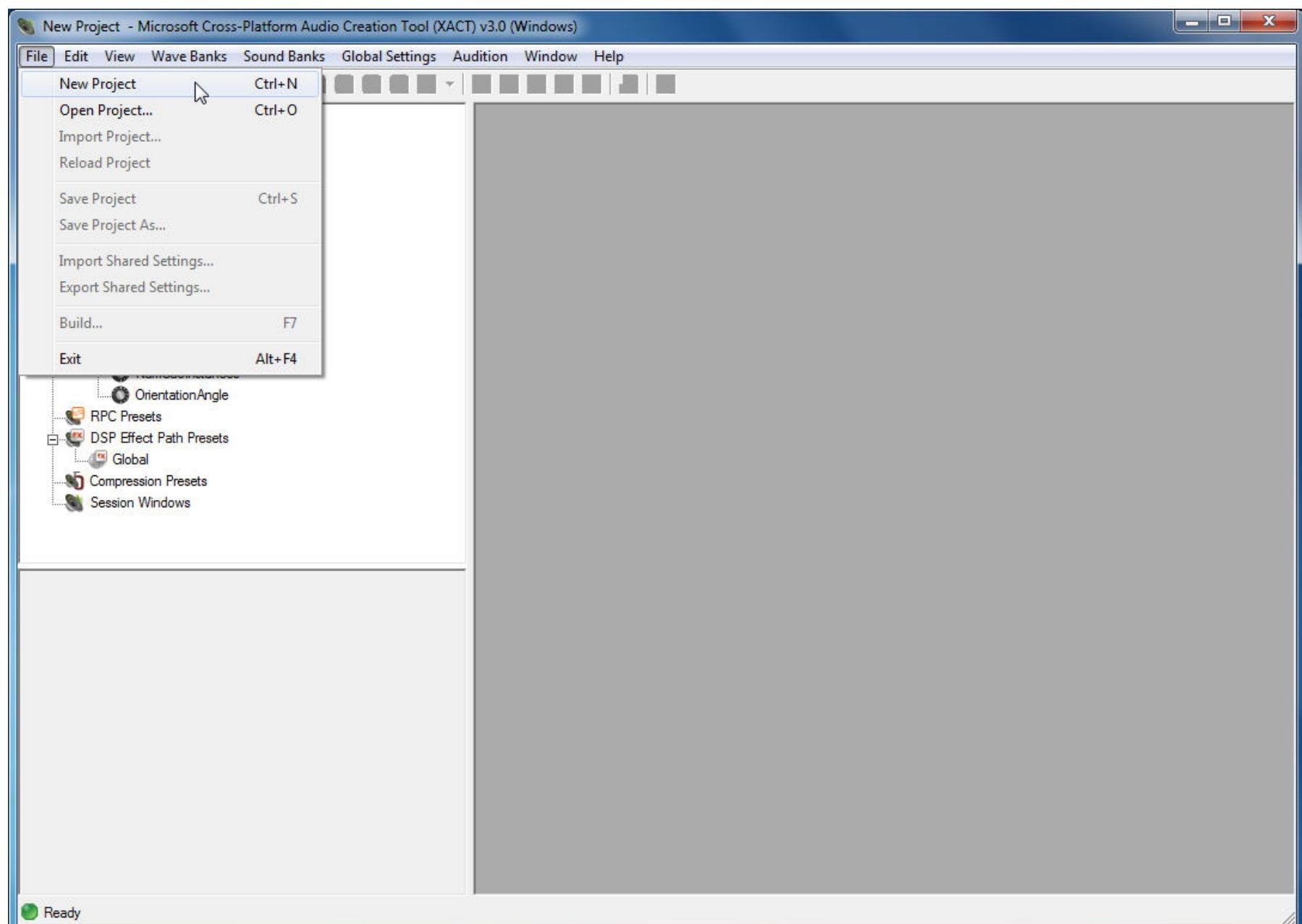


Figura 123: Ejecutando el Microsoft Cross-Platform Audio Creation Tool (XACT3)



**Figura 124: Ejecutando el Microsoft Cross-Platform Audio Creation Tool (XACT3)**



**Figura 125: Se crea un nuevo proyecto**

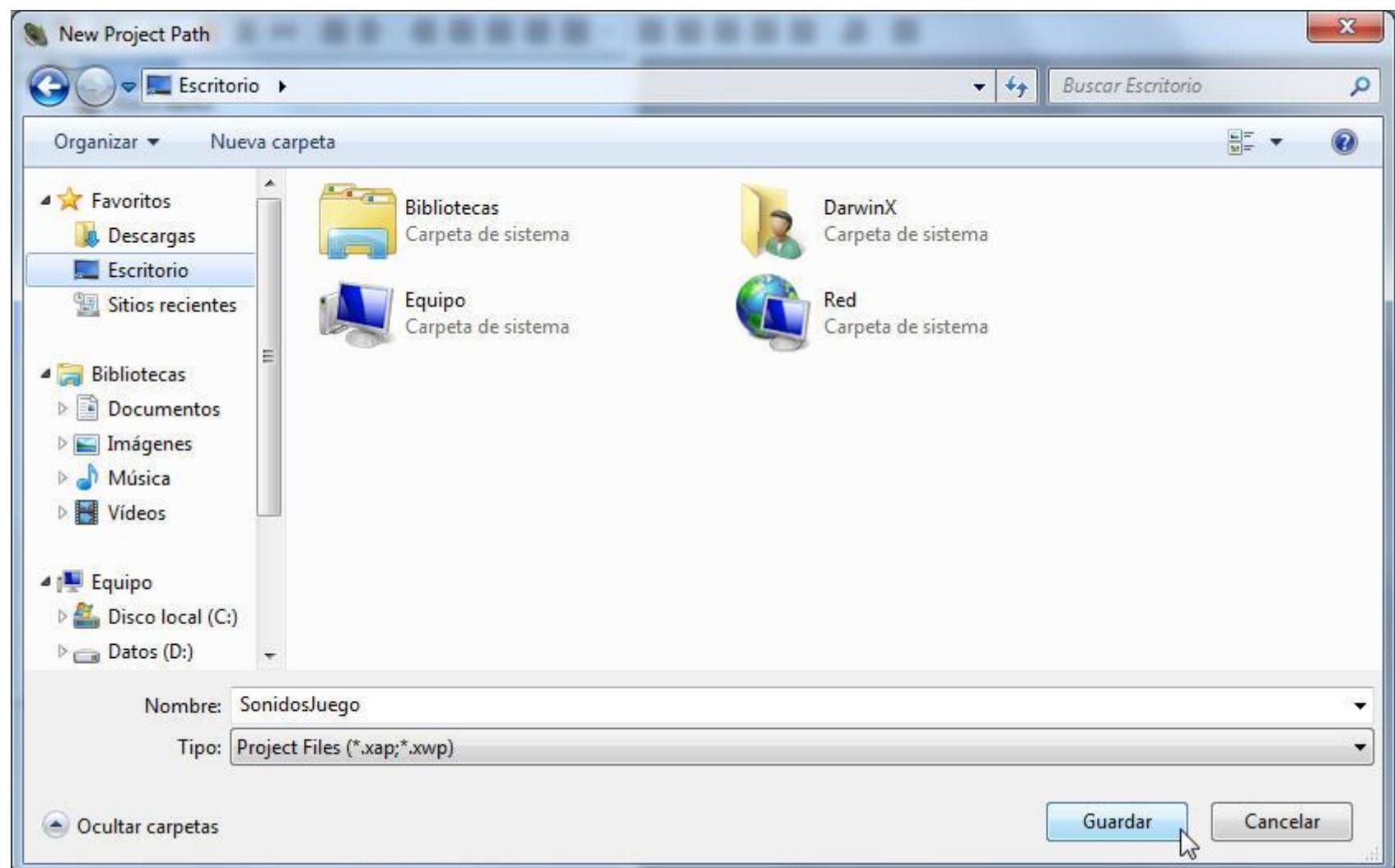


Figura 126: Se guarda el proyecto de sonido

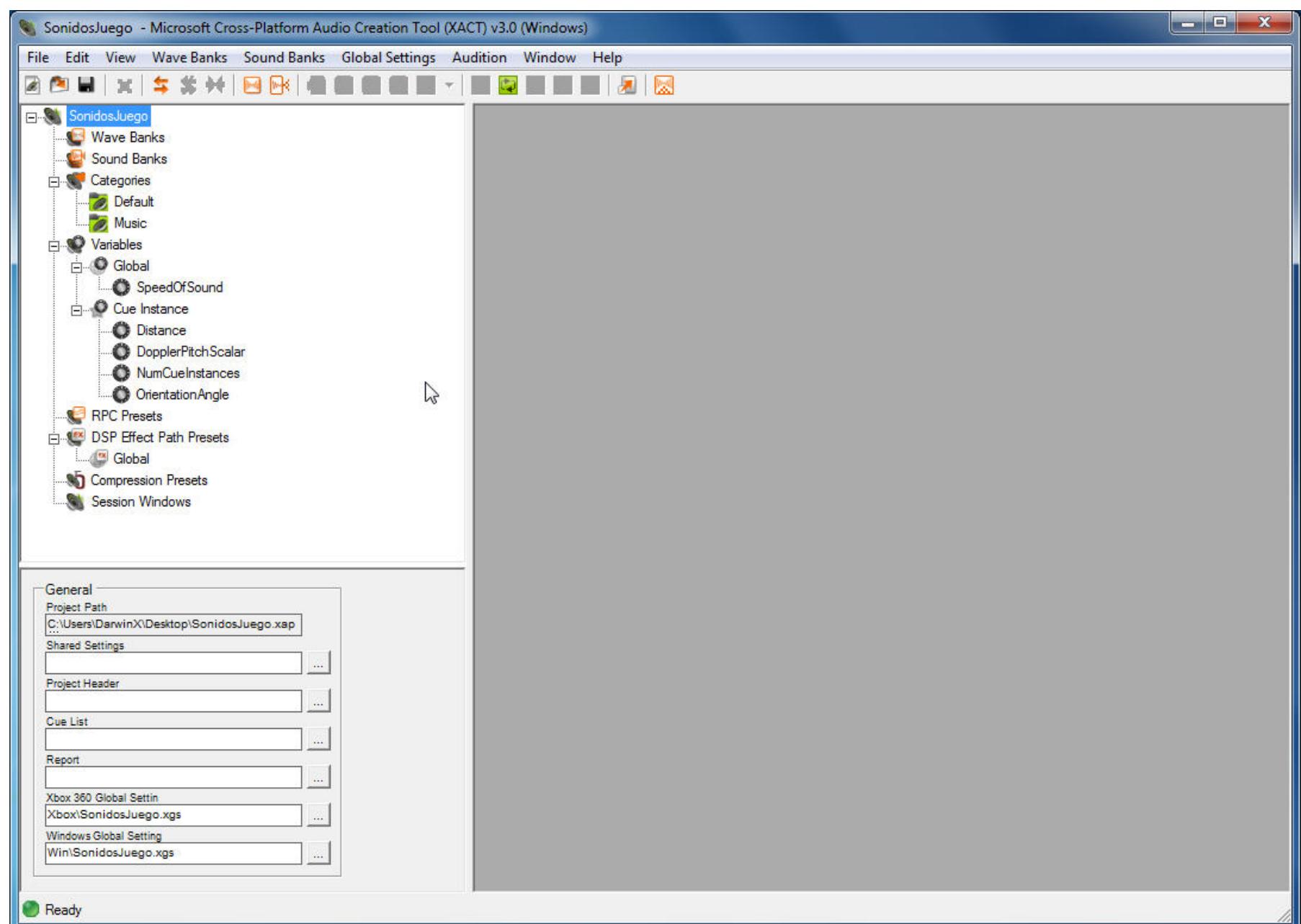
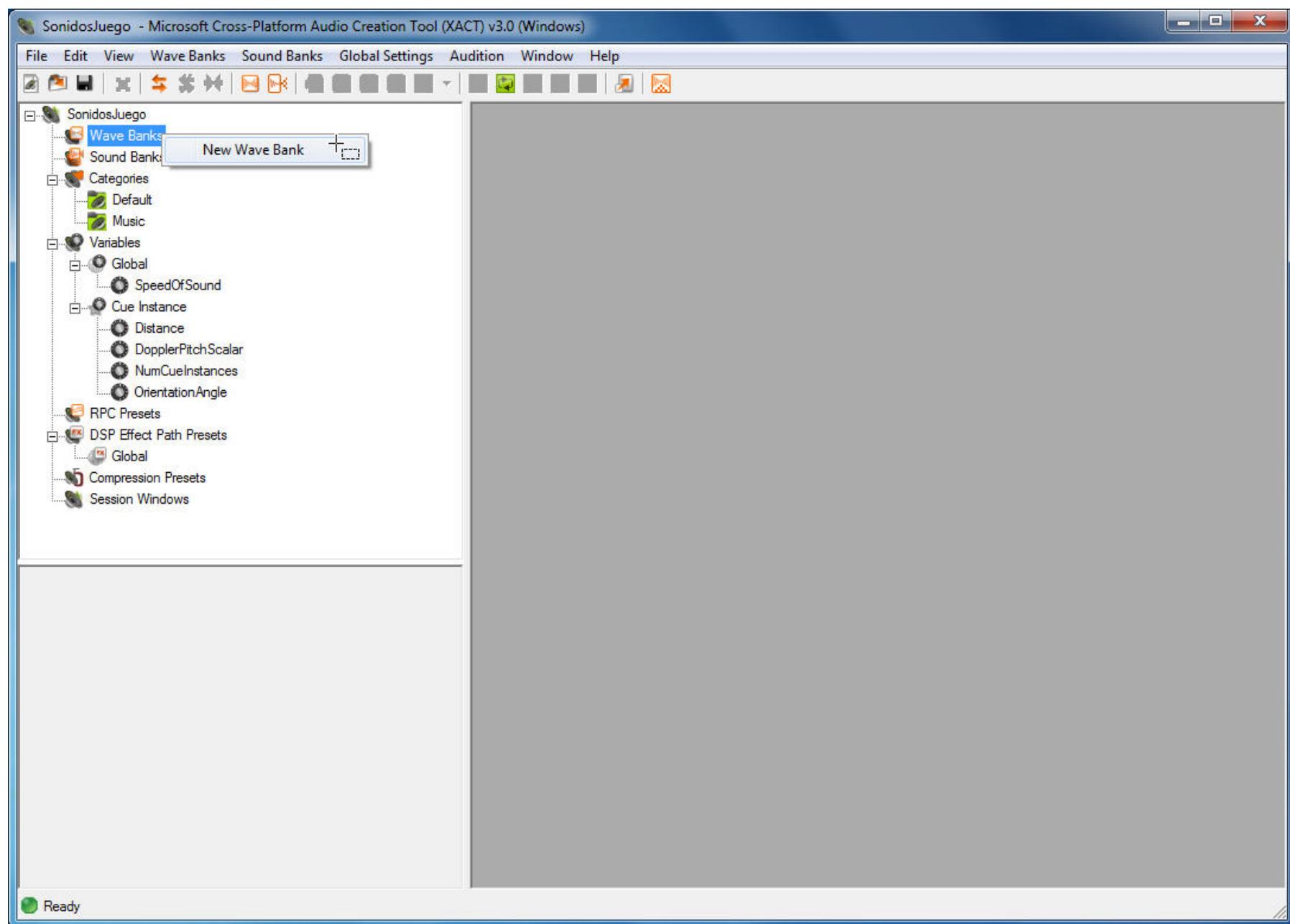
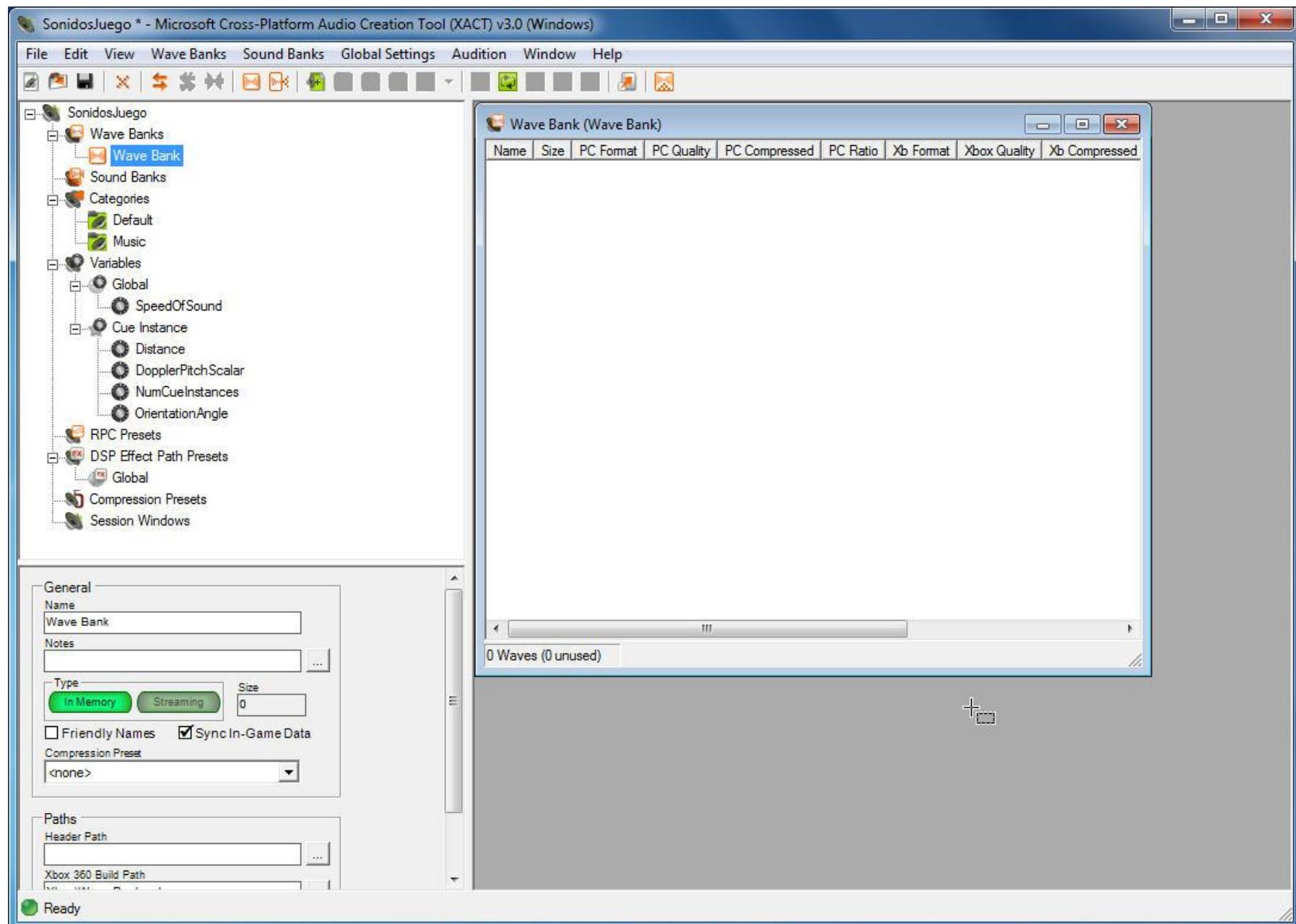


Figura 127: Preparando el proyecto de sonidos

Paso 3: Se crea un nuevo "Wave Bank"

**Figura 128: Crea un nuevo "Wave Bank"****Figura 129: "Wave Bank" generado**

**Paso 4:** Arrastre los archivos .wav a la ventana de Wave Bank

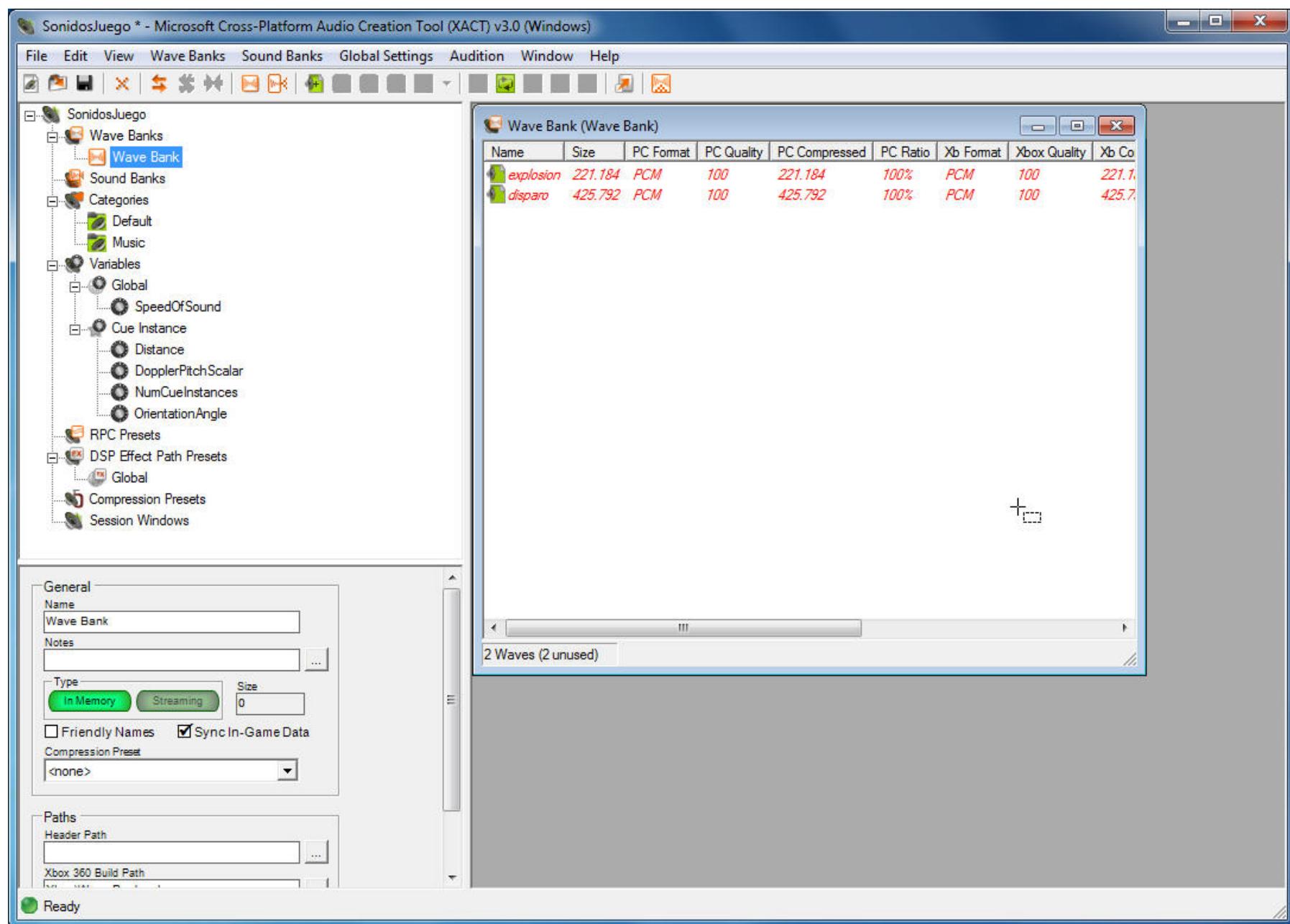


Figura 130: Se agregan los archivos de sonido .wav al "Wave Bank"

#### Paso 5: Creamos un nuevo "Sound Bank"

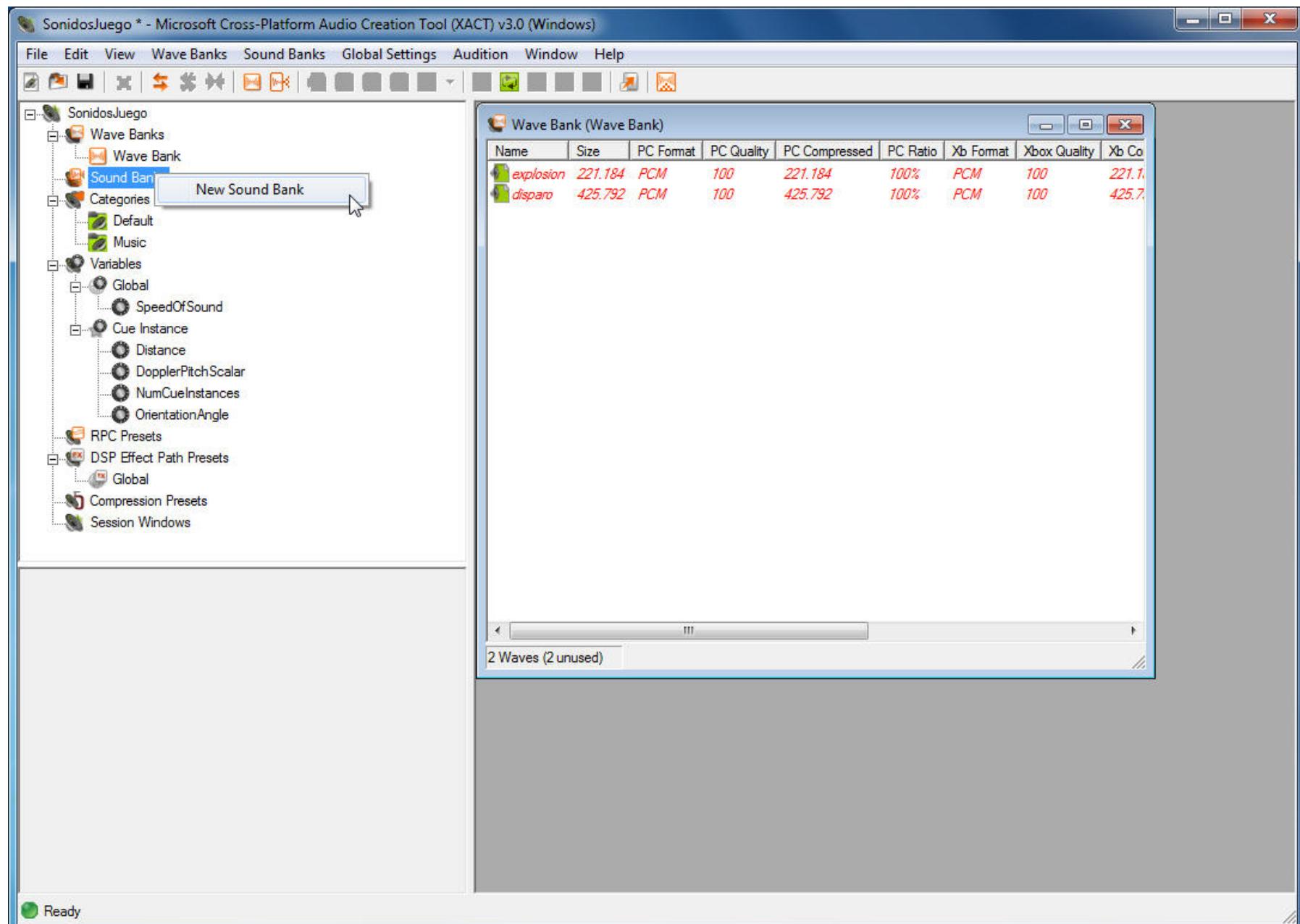


Figura 131: Se genera un nuevo "Sound Bank"

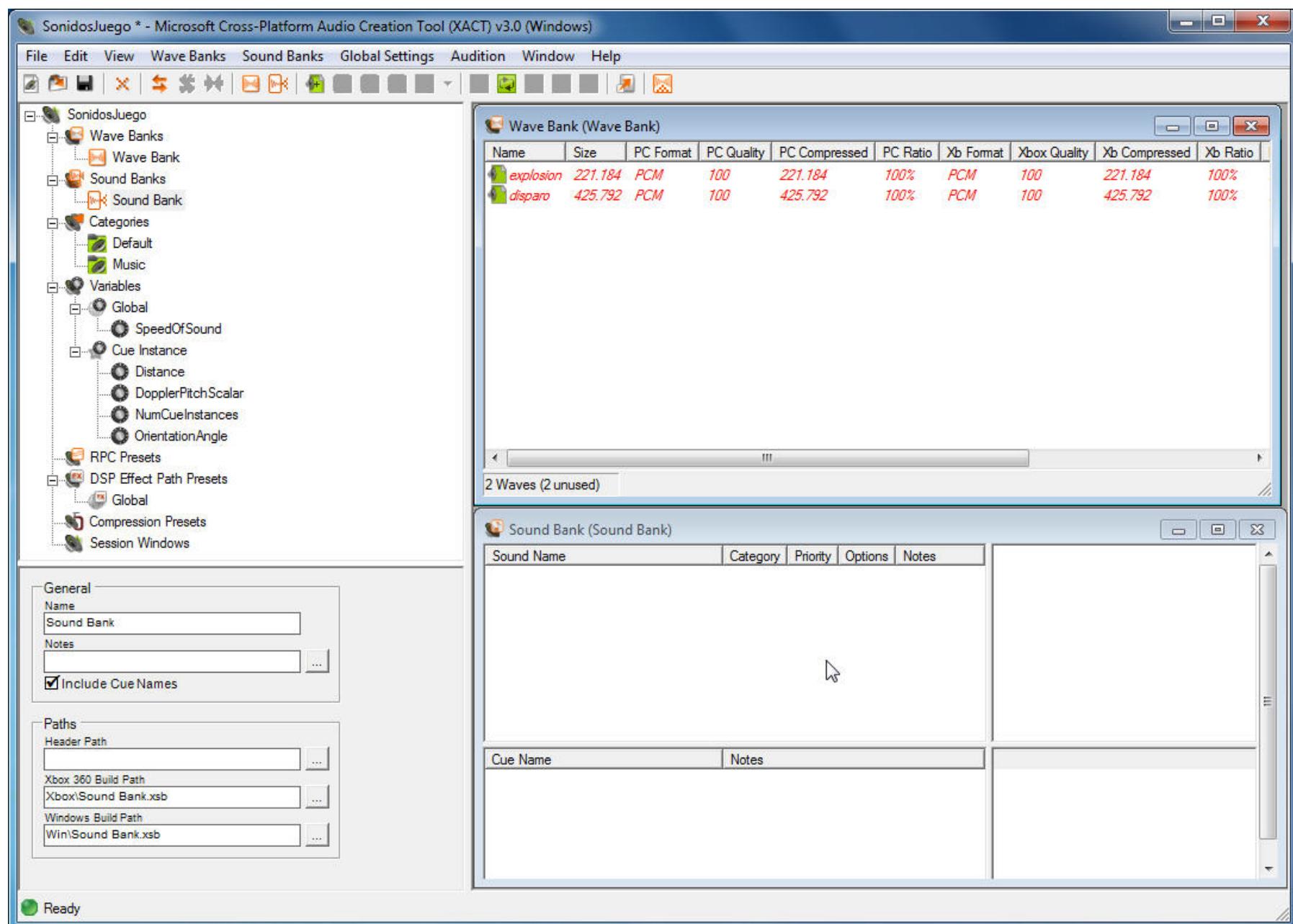


Figura 132: "Sound Bank" generado

#### Paso 6: Arrastramos de "Wave Bank" a "Sound Bank"

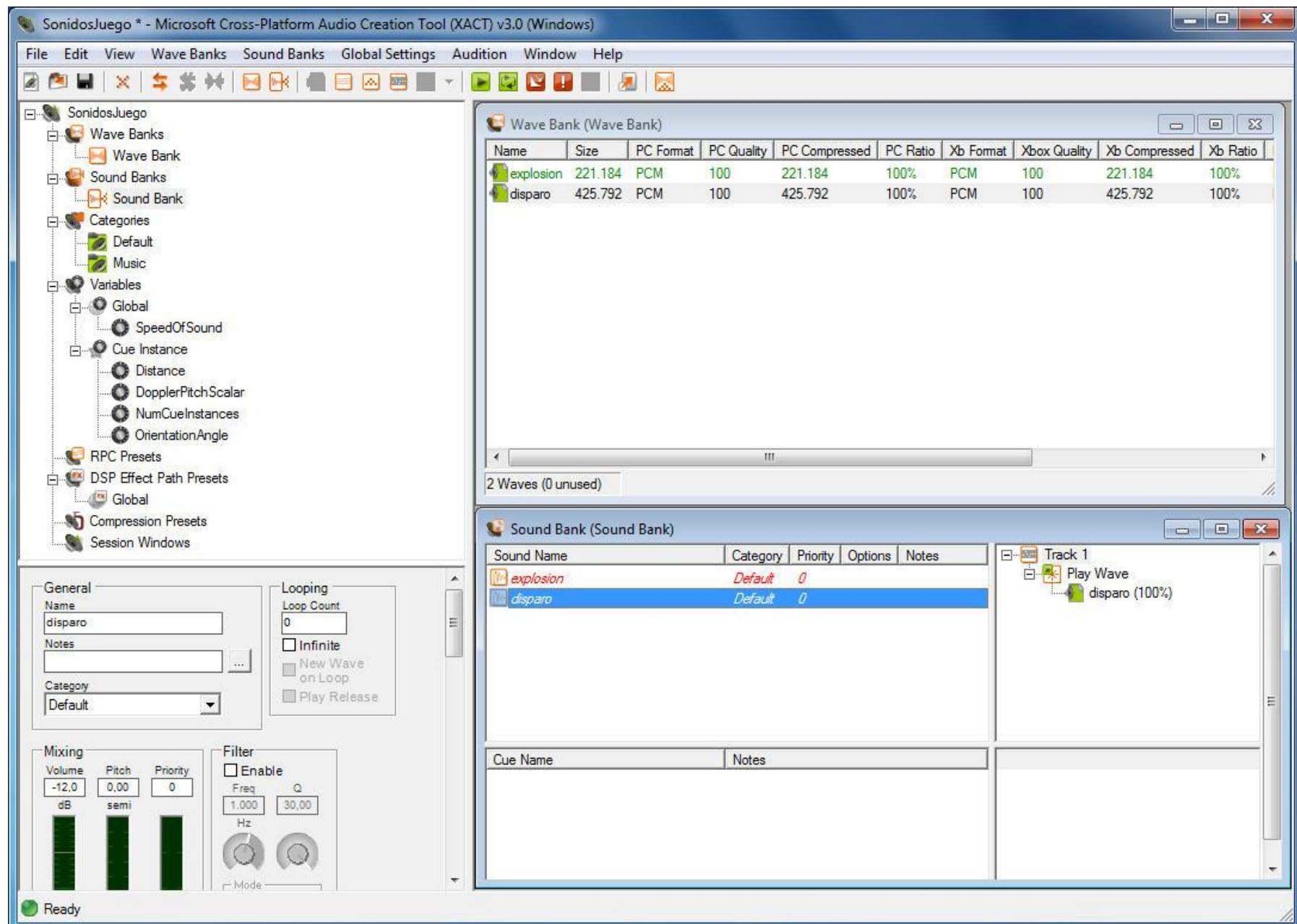


Figura 133: Se arrastra de "Wave Bank" a "Sound Bank"

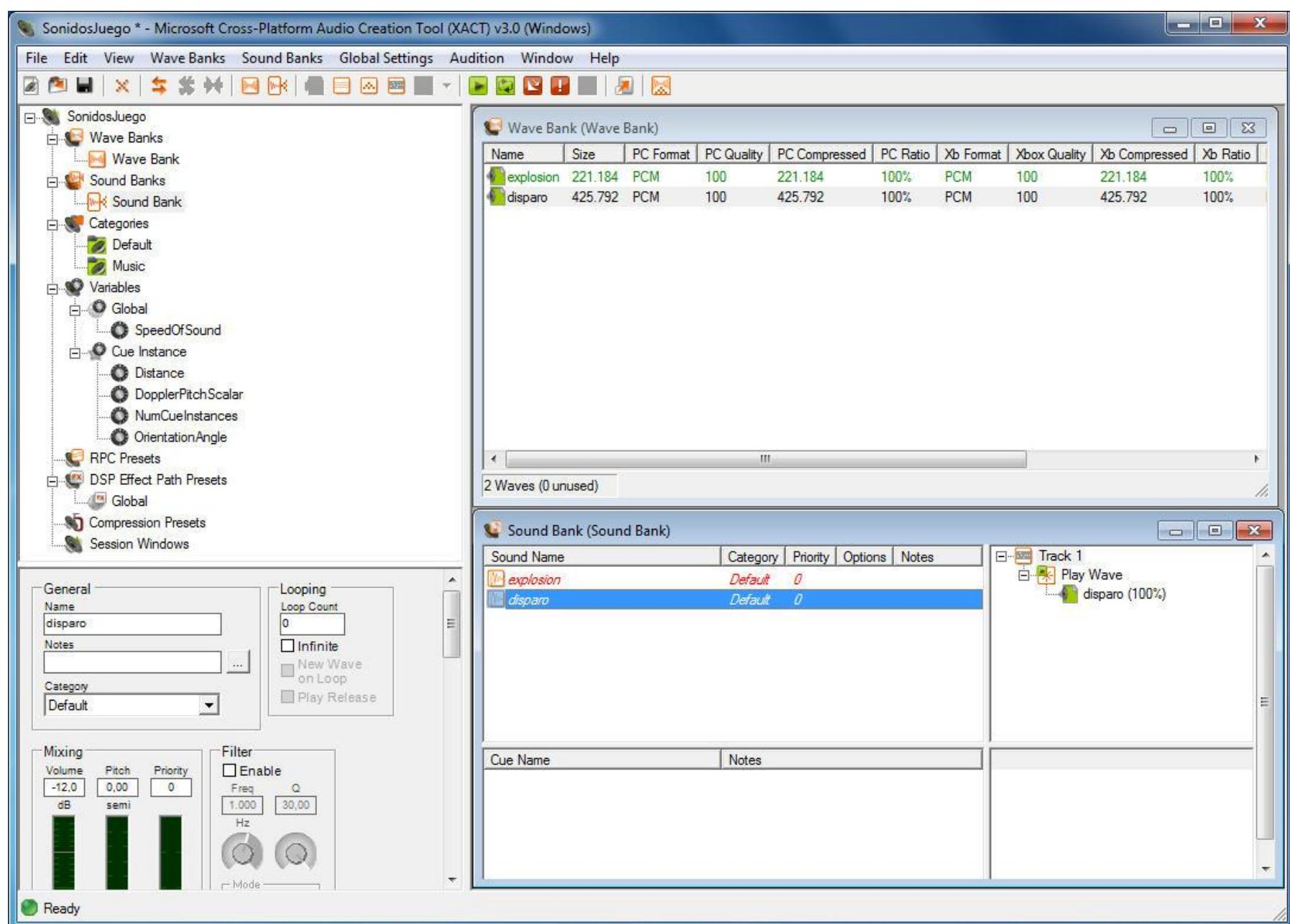


Figura 134: Resultado del arrastre

**Paso 7:** Y arrastramos de "Sound Bank" a "Cue Name" y grabamos el proyecto.

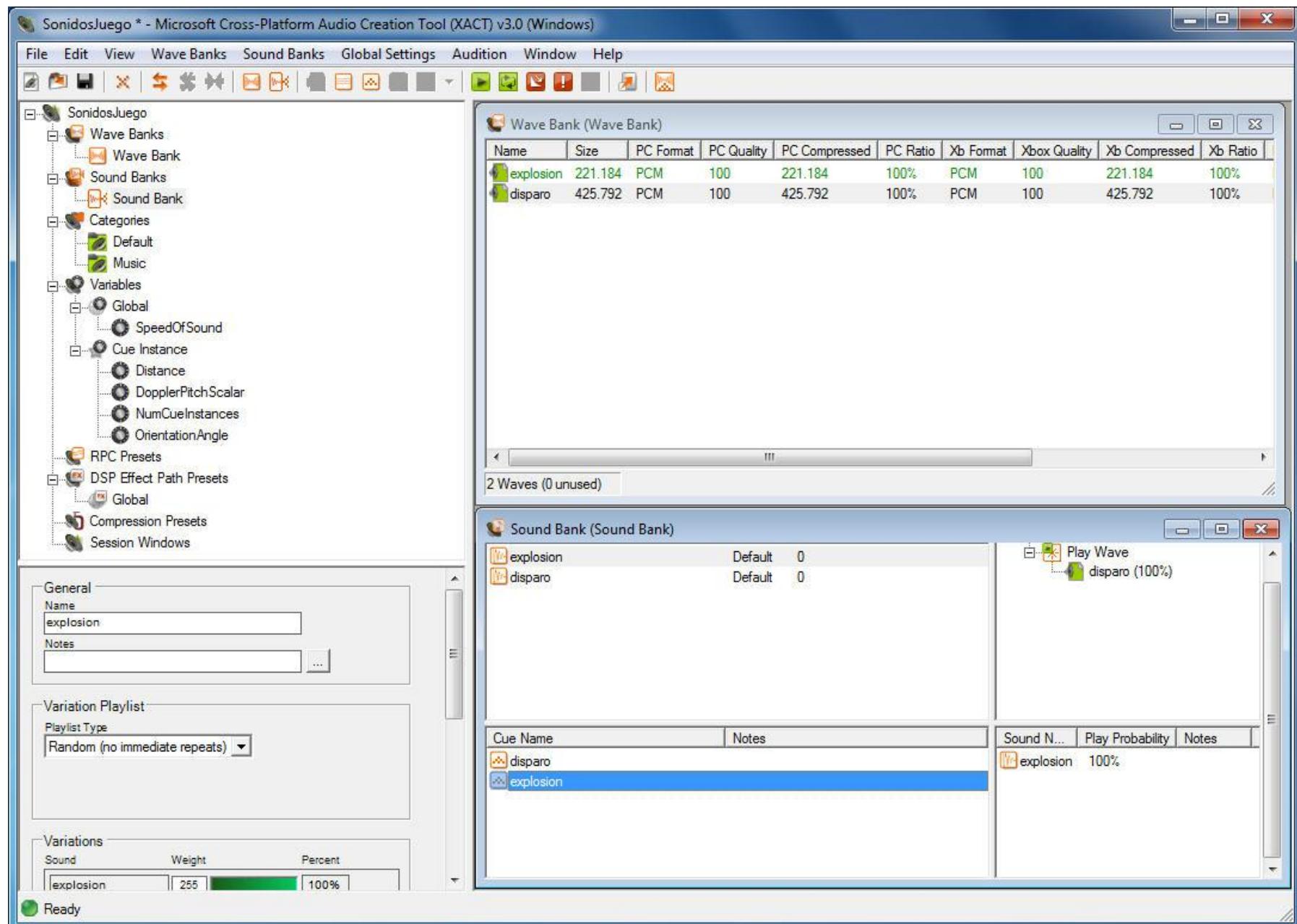


Figura 135: Arrastre de "Sound Bank" a "Cue Name"

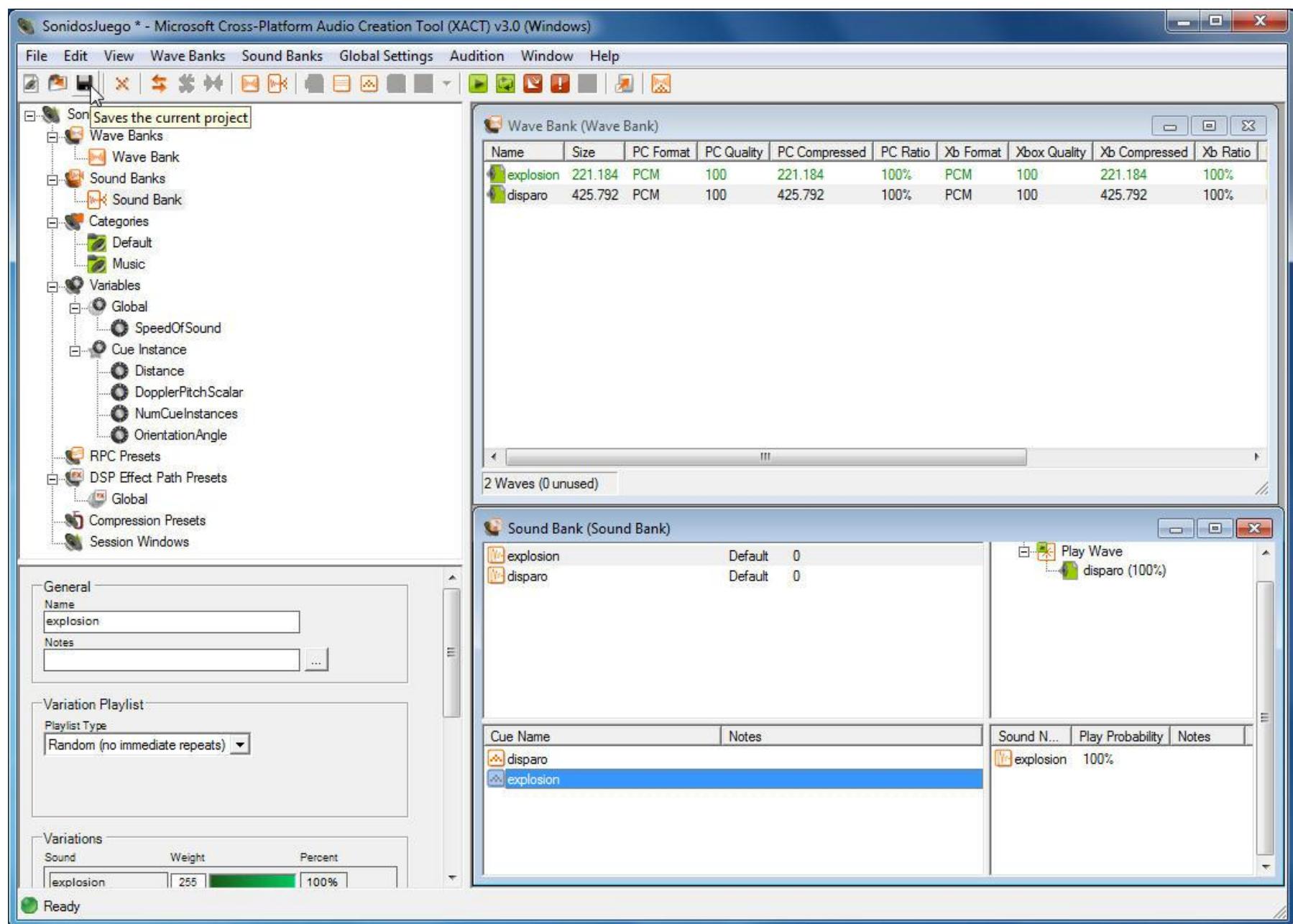


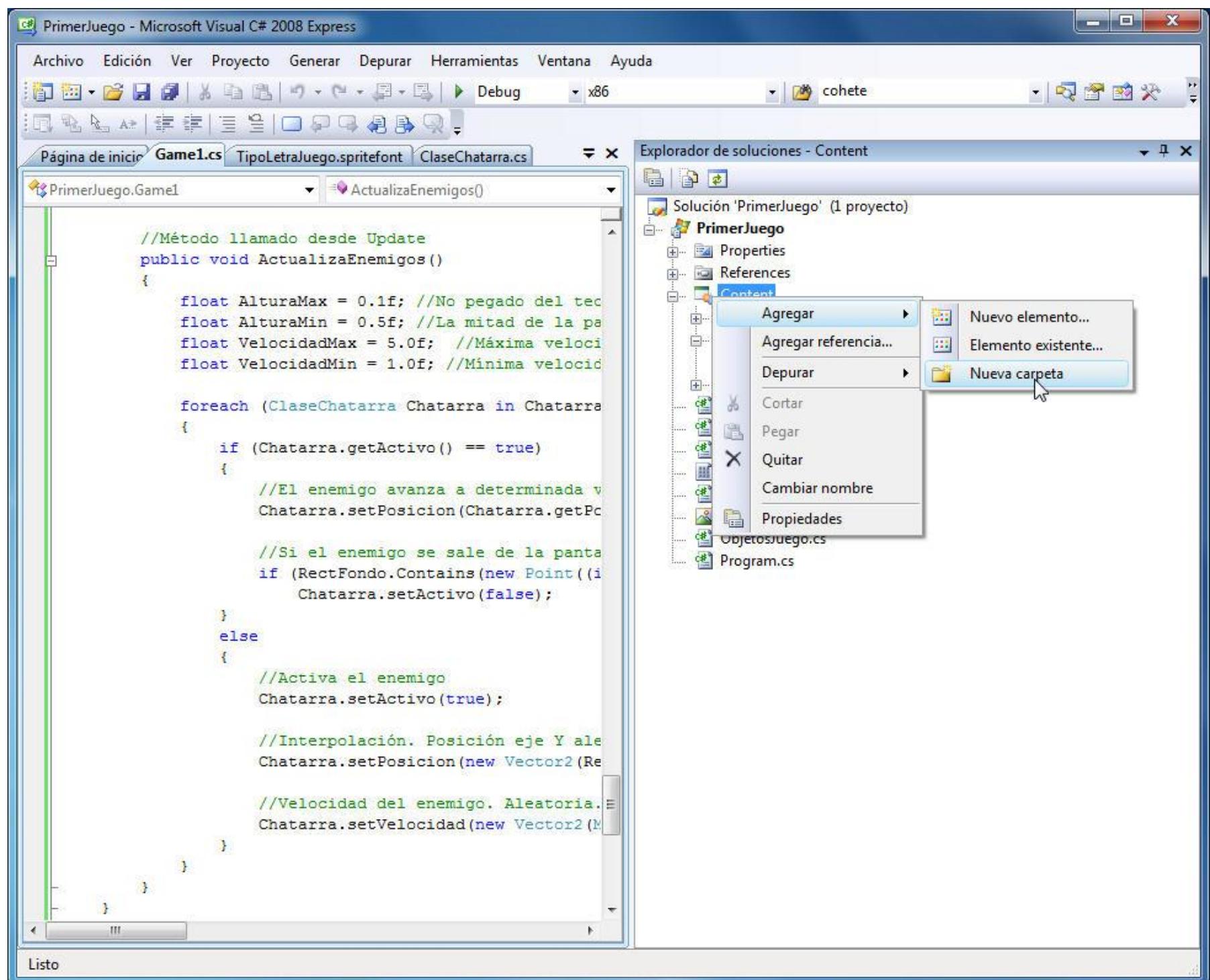
Figura 136: Se graba finalmente el proyecto de sonido

**Paso 8:** Esto es lo que se obtiene

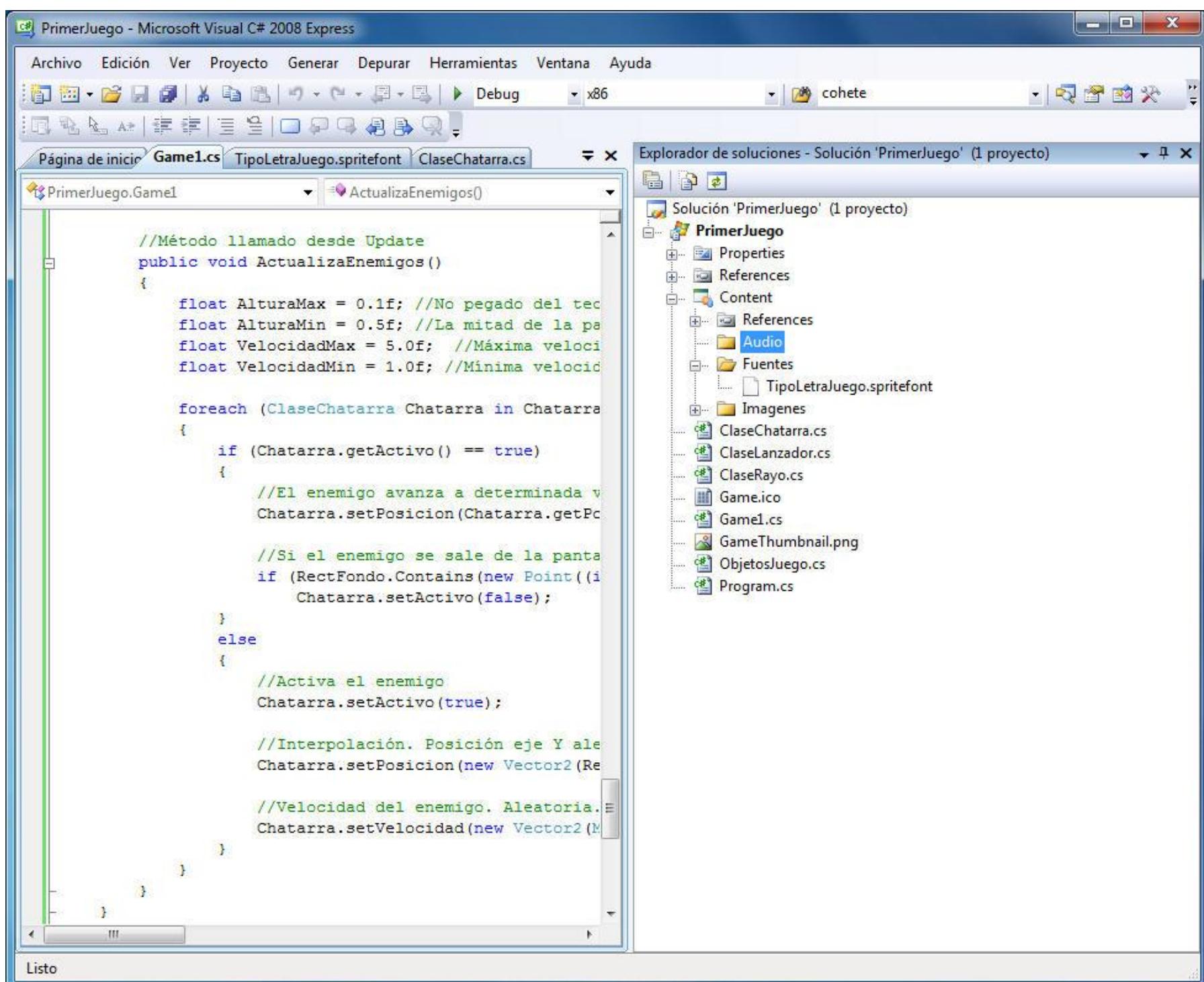


Figura 137: Resultado obtenido del proyecto de sonido

**Paso 9:** En "Content" damos Agregar "Nueva carpeta" y la llamamos "Audio"

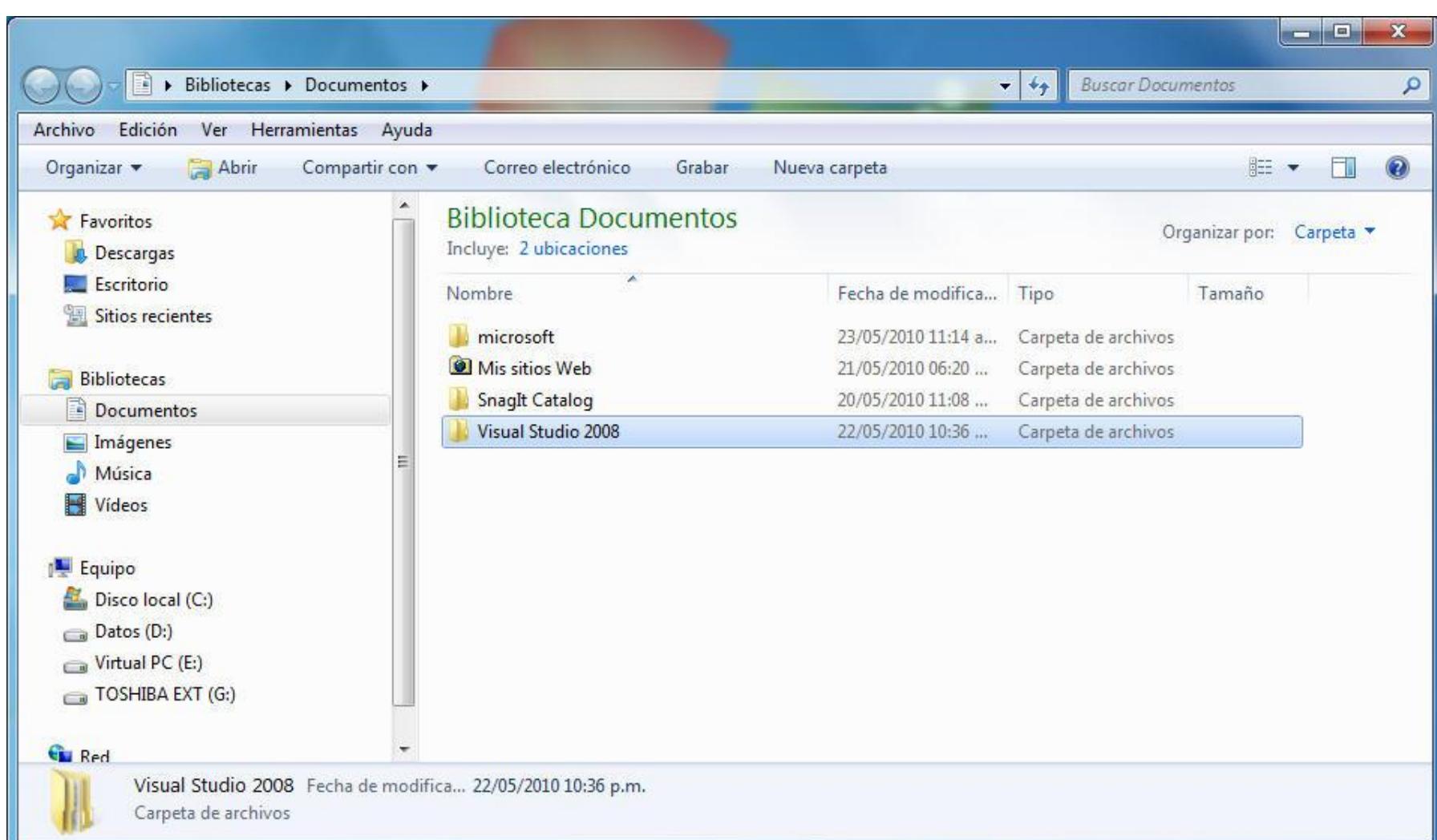


**Figura 138:** Se crea una nueva carpeta en el área de recursos del proyecto de videojuego

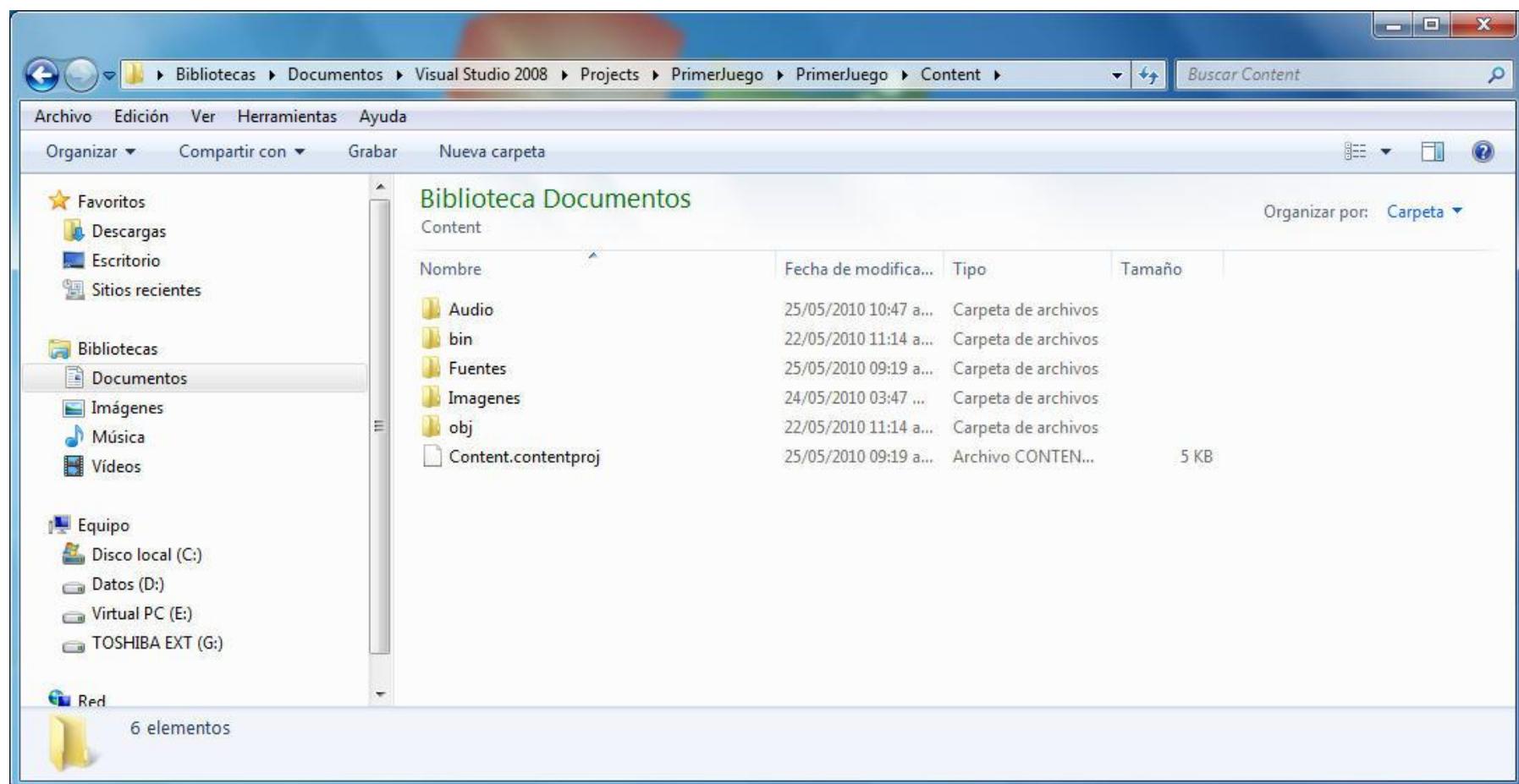


**Figura 139: Carpeta para recurso audio generada**

**Paso 10:** Vamos a la carpeta donde tenemos nuestro proyecto de Visual C# y navegamos por las carpetas hasta llegar a la de Content, allí debe estar la de Audio vacía

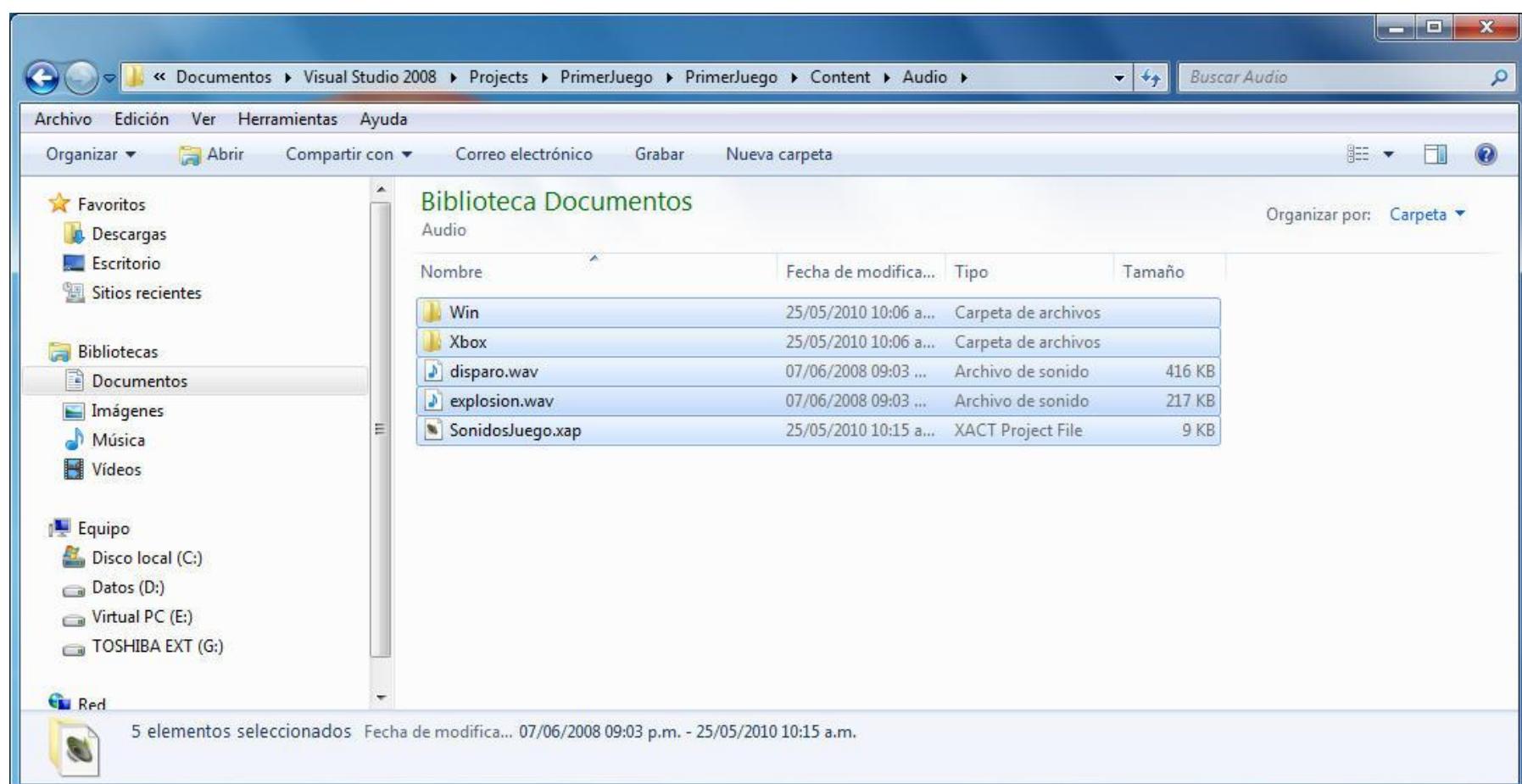


**Figura 140: Buscamos el proyecto de videojuego en el disco duro**



**Figura 141: Ingresa a la sub-carpeta Content y allí se ubica la carpeta Audio**

**Paso 11:** Arrastramos del escritorio todo el proyecto de sonido (junto con los sonidos) a esa carpeta de Audio



**Figura 142: Se toma todo el proyecto de audio y se arrastra a la carpeta señalada anteriormente**

**Paso 12:** A la carpeta de Audio le agregamos "Elemento existente..."

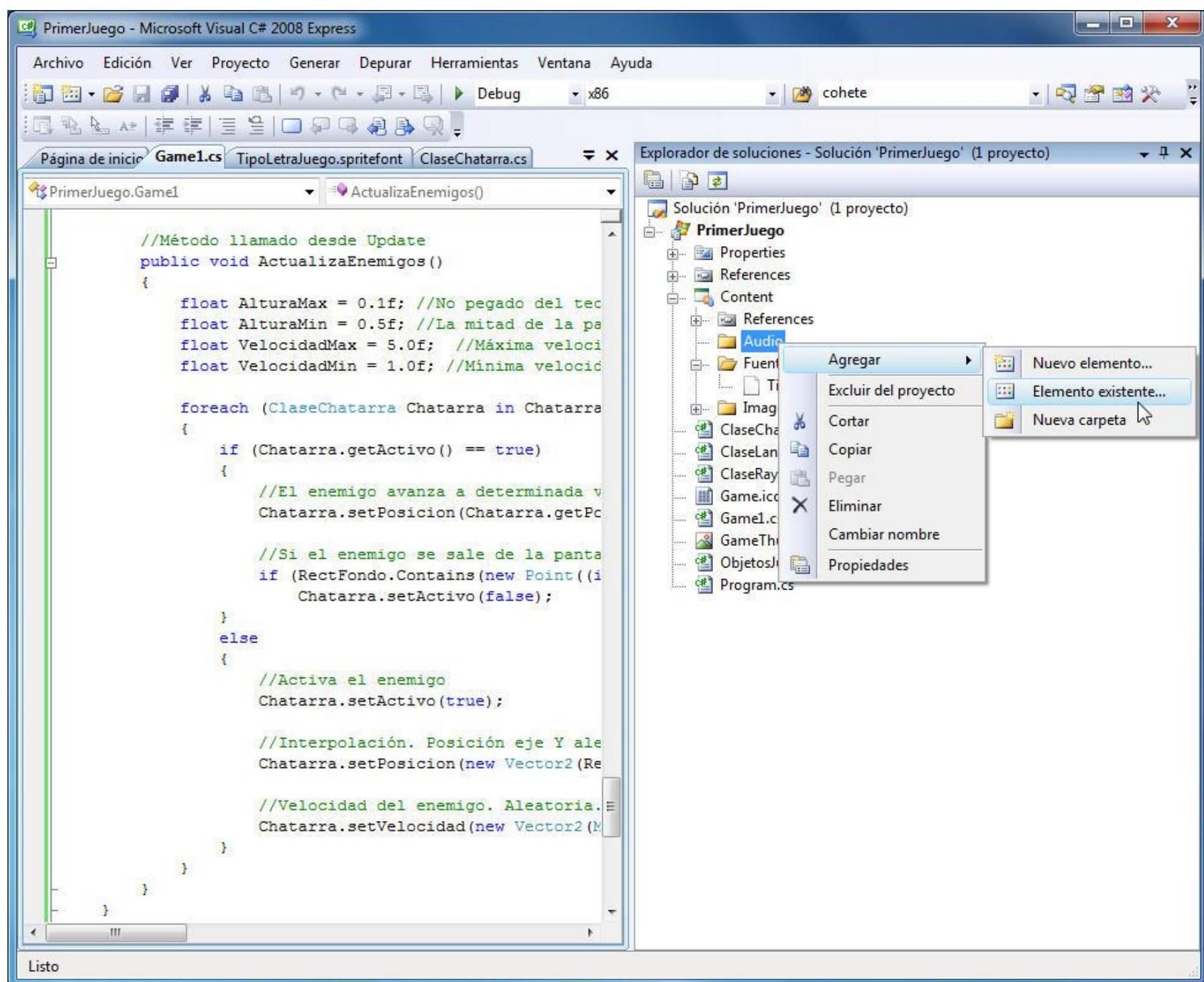


Figura 143: Se agrega elemento existente

**Paso 13:** Y escogemos a SonidosJuego.xap

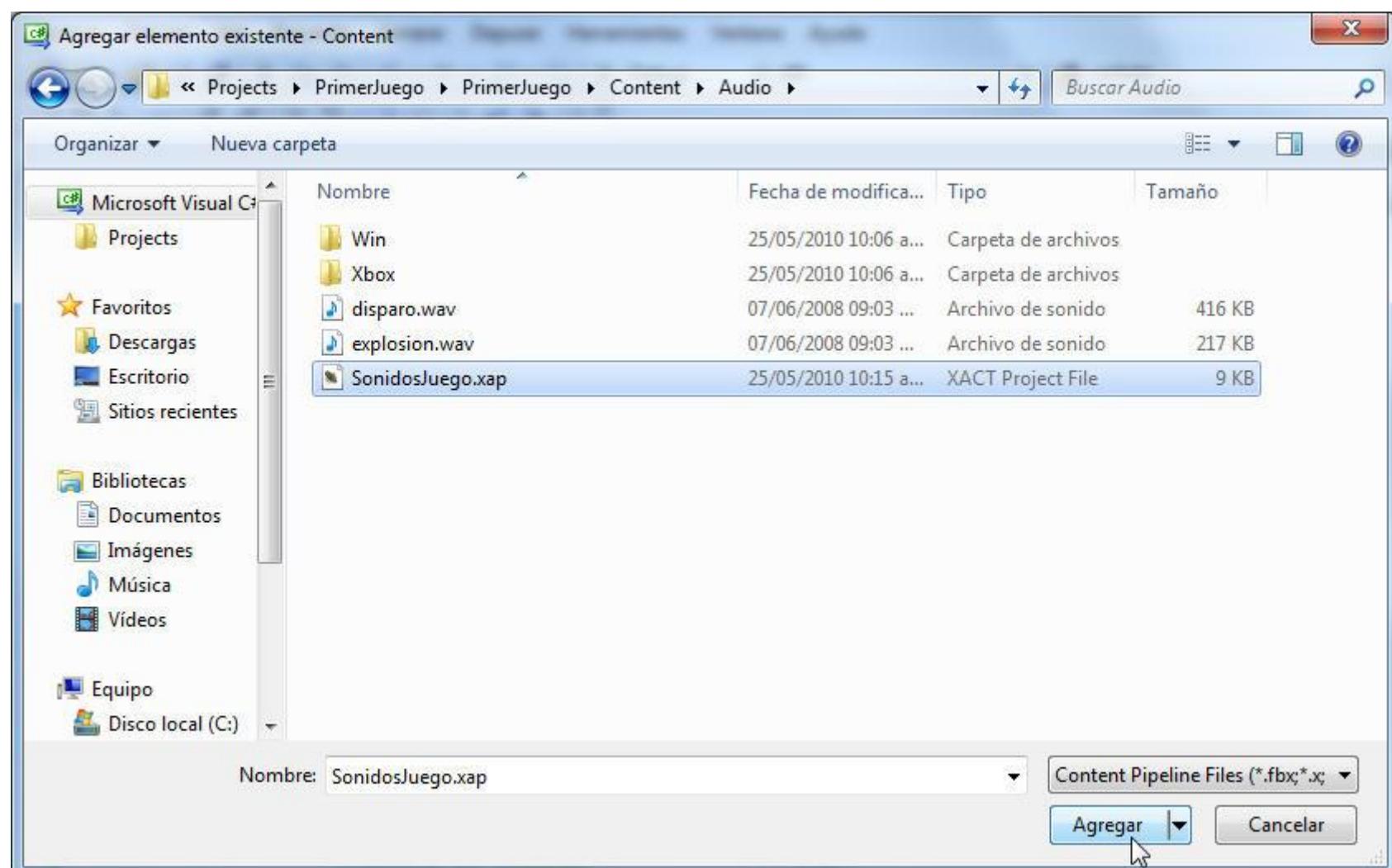
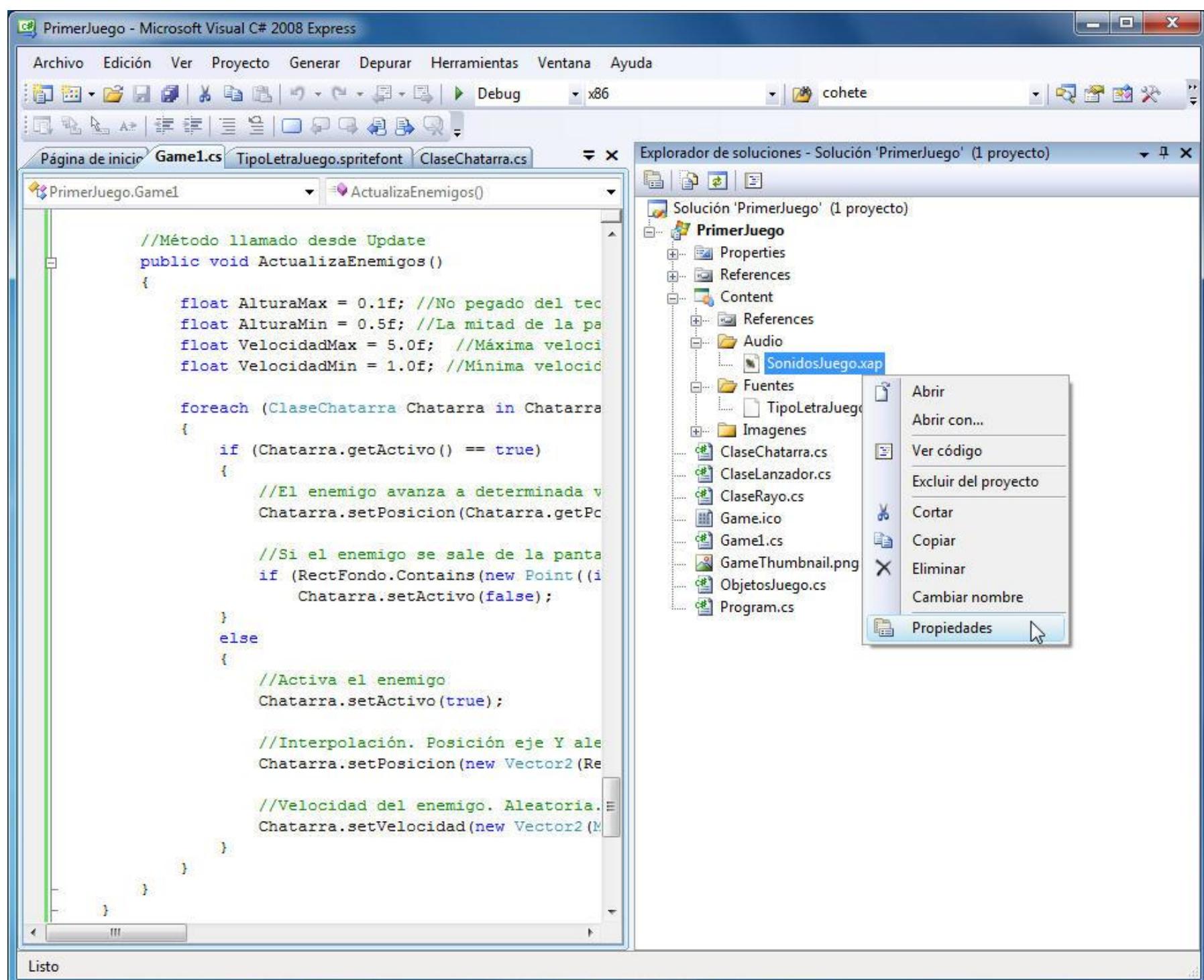


Figura 144: Se busca SonidosJuego.xap

**Paso 14:** Ya queda en nuestro proyecto, ahora veamos las propiedades de SonidosJuego.xap



**Figura 145:** Importante es cambiar el tipo de recurso que es SonidsJuego.xap

**Paso 15:** Cambiamos la propiedad "Content Processor" por Sound Effect - XNA Framework

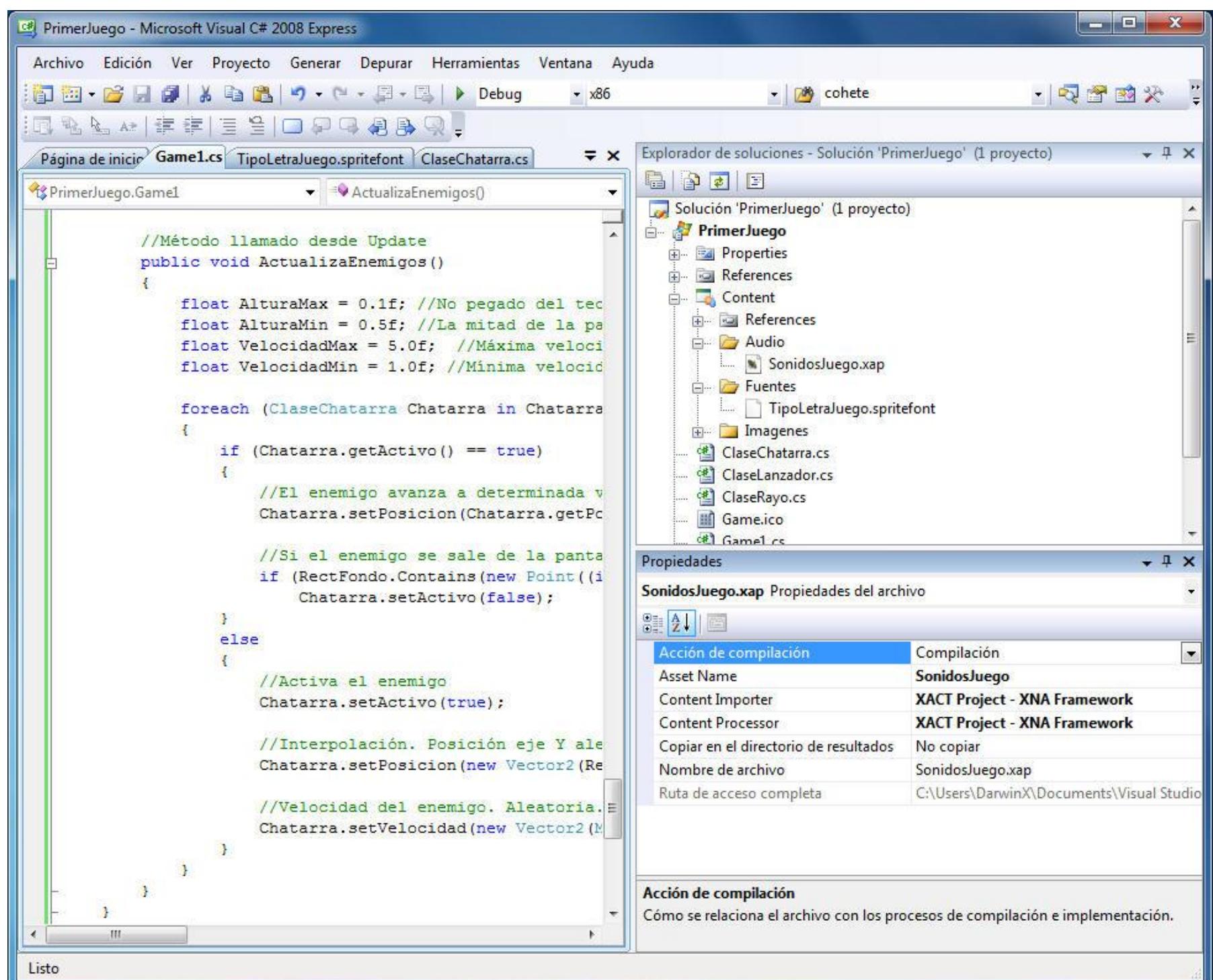


Figura 146: Originalmente es XACT Project como Content Processor

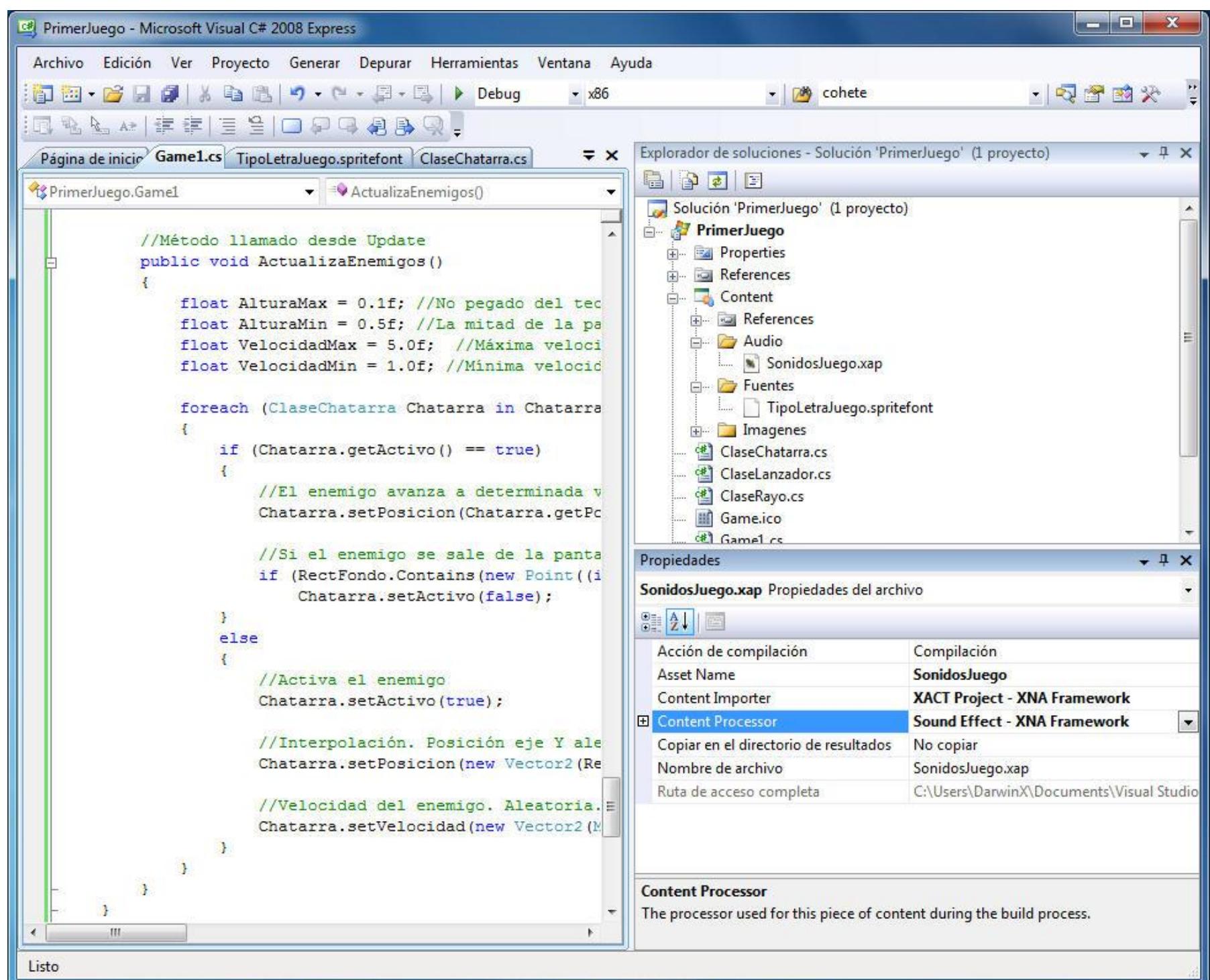


Figura 147: Se cambia a Sound Effect

**Paso 16:** Agregar el código requerido para el manejo de sonidos. Atributos de la clase Game1.cs

```
//Audio
AudioEngine MotorAudio;
SoundBank BancoSonido;
WaveBank BancoAudio;
```

Cargando los sonidos en LoadContent()

```
//Cargar el sonido (ojo con la extensión .xgs)
MotorAudio = new AudioEngine("Content\\Audio\\SonidosJuego.xgs"); //xna game studio
BancoSonido = new SoundBank(MotorAudio, "Content\\Audio\\Sound Bank.xsb"); //xna sound bank
BancoAudio = new WaveBank(MotorAudio, "Content\\Audio\\Wave Bank.xwb"); //xna wave bank
```

Ejecutando los sonidos cuando se dispara o cuando hay una colisión (Hay que tener en cuenta las mayúsculas y minúsculas en el nombre del sonido).

```
//Sonido de la explosión
BancoSonido.PlayCue("explosion");
```

Y

```
//Sonido del disparo
BancoSonido.PlayCue("disparo");
```

Este es el código completo

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
```

```

namespace PrimerJuego
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch; //Contenedor de los sprite

        Texture2D Fondo; //Variable de tipo textura que tendrá el fondo
        Rectangle RectFondo; //Un rectángulo donde estará la textura del fondo

        ClaseLanzador Lanzador; //El lanzador
        ClaseLanzador BaseLanzador; //Base del lanzador

        //Almacena el estado anterior del ratón
        MouseState anteriorRaton;

        //Almacena el estado anterior del teclado
        KeyboardState anteriorTeclado;

        //Rayos
        ClaseRayo[] Rayos;
        const int MAXRAYOS = 7;

        //Chatarras
        ClaseChatarra[] Chatarras;
        const int MAXCHATARRAS = 9;

        //Generador de números aleatorios usado para ubicar a una altura al azar la chatarra
        Random Aleatorio = new Random();

        //Puntaje
        int Puntaje = 0; //Lleva cuantos enemigos han sido destruidos
        SpriteFont Fuente; //Variable tipo de letra para mostrar el puntaje

        //Audio
        AudioEngine MotorAudio;
        SoundBank BancoSonido;
        WaveBank BancoAudio;

        //Constructor
        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here
            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // TODO: use this.Content to load your game content here

            //Utiliza genéricos <> por eso el uso de <> En C++ se llaman plantillas. Content carga el contenido binario.
            Fondo = Content.Load<Texture2D>("Imagenes\\FondoJuego");

            //El rectángulo en el que estará contenido el fondo
            RectFondo = new Rectangle(0, 0, graphics.GraphicsDevice.Viewport.Width, graphics.GraphicsDevice.Viewport.Height);

            //Cargar e inicializar el Lanzador
            Lanzador = new ClaseLanzador(Content.Load<Texture2D>("Imagenes\\Lanzador")); //No es necesaria la extensión .tga
            Lanzador.setPosicion(new Vector2(160, 530)); //Posición del lanzador

            //Cargar e inicializar la base
            BaseLanzador = new ClaseLanzador(Content.Load<Texture2D>("Imagenes\\Base")); //No es necesaria la extensión .tga
            BaseLanzador.setPosicion(new Vector2(70, 560)); //Posición de la base del lanzador

            //Cargar e inicializar los rayos
            Rayos = new ClaseRayo[MAXRAYOS];
            for (int Cont = 0; Cont < MAXRAYOS; Cont++)
                Rayos[Cont] = new ClaseRayo(Content.Load<Texture2D>("Imagenes\\Rayo"));

            //Cargar e inicializar los rayos
        }
    }
}

```

```

Chatarras = new ClaseChatarra[MAXCHATARRAS];
for (int Cont = 0; Cont < MAXCHATARRAS; Cont++)
    Chatarras[Cont] = new ClaseChatarra(Content.Load<Texture2D>("Imagenes\\Chatarra"));

//Cargar la fuente
Fuente = Content.Load<SpriteFont>("Fuentes\\TipoLetraJuego");

//Cargar el sonido (ojo con la extension .xgs)
MotorAudio = new AudioEngine("Content\\Audio\\SonidosJuego.xgs"); //xna game studio
BancoSonido = new SoundBank(MotorAudio, "Content\\Audio\\Sound Bank.xsb"); //xna sound bank
BancoAudio = new WaveBank(MotorAudio, "Content\\Audio\\Wave Bank.xwb"); //xna wave bank
}

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // Código para leer el estado del Gamepad de una XBOX. Se considera que hay un jugador al menos.
    GamePadState EstadoControl = GamePad.GetState(PlayerIndex.One);

    //Como hay que tener en cuenta los controles del XBOX llamados thumbstick, se
    //observa que botón del XBOX oprimió, eso genera una constante algo grande por lo que se multiplica por 0.1
    Lanzador.setRotacion(Lanzador.getRotacion() + EstadoControl.ThumbSticks.Left.X * 0.1f);

    //Esta es una compilación condicional. El teclado y el mouse solo son leídos si la aplicación ejecuta en un PC
#ifndef !XBOX
    //Código para leer el teclado
    KeyboardState estadoTeclado = Keyboard.GetState();
    if (estadoTeclado.IsKeyDown(Keys.Left)) Lanzador.setRotacion(Lanzador.getRotacion() - 0.1f); //Gira el lanzador hacia la
    izquierda
    if (estadoTeclado.IsKeyDown(Keys.Right)) Lanzador.setRotacion(Lanzador.getRotacion() + 0.1f); //Gira el lanzador hacia la
    derecha

    //Código para leer el ratón
    MouseState Raton = Mouse.GetState();

    //Si movió el ratón entonces el lanzador cambia de ángulo
    if (Raton != anteriorRaton)
    {
        int DiferX = Raton.X - anteriorRaton.X; //La diferencia entre la anterior posición del ratón y la nueva.
        Lanzador.setRotacion(Lanzador.getRotacion() + (float)DiferX / 100); //El movimiento en X del ratón cambia el ángulo del
        lanzador
    }

    //Se actualiza el estado del ratón
    anteriorRaton = Raton;

    //Dispara un solo rayo.
    //Chequea si el usuario mantiene presionada la tecla de disparo por lo que hace caso omiso a eso,
    //el usuario debe presionar y soltar la tecla de disparo para lanzar mas rayos.
    if (estadoTeclado.IsKeyDown(Keys.Space) && anteriorTeclado.IsKeyUp(Keys.Space))
        DispararRayo();

    anteriorTeclado = estadoTeclado;
#endif

    //Esta instrucción limita entre 0 y 90 grados el giro del lanzador. Recordar que se hace uso de radianes.
    Lanzador.setRotacion(MathHelper.Clamp(Lanzador.getRotacion(), -MathHelper.PiOver2, 0));

    //Actualiza los rayos
    ActualizaRayos();

    //Actualiza los enemigos
    ActualizaEnemigos();

    // TODO: Add your update logic here
    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
}

```

```

// TODO: Add your drawing code here

//Se inicia el spritebatch (Entre Begin y End se dibuja en cada unidad de tiempo los sprites)
spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

//Dibujar el fondo combinando el color blanco con el fondo
spriteBatch.Draw(Fondo, RectFondo, Color.White);

//Dibuja la base del lanzador combinando con blanco
spriteBatch.Draw(BaseLanzador.getTextura(), BaseLanzador.getPosicion(), Color.White);

/* Dibuja el lanzador como tal, estos son los parámetros
 * Parámetro 1: El sprite del lanzador
 * Parámetro 2: La posición del lanzador
 * Parámetro 3: null porque es solo una imagen (no una sucesión de estas)
 * Parámetro 4: Color.White, combinación de color
 * Parámetro 5: Angulo de rotación por defecto
 * Parámetro 6: Centro, el centro del sprite para girarlo después
 * Parámetro 7: Escala a dibujar
 * Parametro 8: Dibujar tal como es el sprite (no reflejo vertical ni horizontal)
 * Parámetro 9: Ponerlo en la capa superior */
spriteBatch.Draw(Lanzador.getTextura(), Lanzador.getPosicion(), null, Color.White, Lanzador.getRotacion(),
Lanzador.getCentro(), 1f, SpriteEffects.None, 0);

//Dibuja cada rayo
foreach (ClaseRayo Rayo in Rayos)
    if (Rayo.getActivo() == true)
        spriteBatch.Draw(Rayo.getTextura(), Rayo.getPosicion(), null, Color.White, Rayo.getRotacion(), Rayo.getCentro(), 1f,
SpriteEffects.None, 0);

//Dibuja cada enemigo
foreach (ClaseChatarra Chatarra in Chatarras)
    if (Chatarra.getActivo() == true)
        spriteBatch.Draw(Chatarra.getTextura(), Chatarra.getPosicion(), Color.White);

//Dibuja el puntaje
spriteBatch.DrawString(Fuente, "Destruido: " + Puntaje.ToString(), new Vector2(80, 60), Color.White);

//Finaliza el "ciclo" de los sprite
spriteBatch.End();

base.Draw(gameTime);
}

//Metodo llamado desde Update
public void DispararRayo()
{
    //Busca un rayo inactivo
    foreach (ClaseRayo Rayo in Rayos)
    {
        //Si encuentra un rayo inactivo
        if (Rayo.getActivo() == false)
        {
            //Activa el rayo (para que sea dibujado)
            Rayo.setActivo(true);

            //El rayo inicia en la posición del lanzador
            Rayo.setPosicion(Lanzador.getPosicion());

            //En que dirección y a que velocidad sale el rayo. Hay que tener en cuenta la rotación del lanzador.
            Rayo.setVelocidad(new Vector2((float)Math.Cos(Lanzador.getRotacion()), (float)Math.Sin(Lanzador.getRotacion())) * 4.0f);

            //Rote el sprite del rayo para que coincida con el del lanzador
            Rayo.setRotacion(Lanzador.getRotacion());

            //Sonido del disparo
            BancoSonido.PlayCue("disparo");

            return;
        }
    }
}

//Método llamado desde Update
public void ActualizaRayos()
{
    //Va de rayo en rayo
    foreach (ClaseRayo Rayo in Rayos)
        if (Rayo.getActivo() == true)
        {
            //Solo es actualizar la velocidad porque el vector se actualiza gracias a esa magnitud
            Rayo.setPosicion(Rayo.getPosicion() + Rayo.getVelocidad());

            //Chequea la colisión

            //Deduce el rectángulo del cohete y debe tener en cuenta el tamaño con que se esté dibujando en pantalla en Draw()
            Rectangle RectRayo = new Rectangle((int)Rayo.getPosicion().X, (int)Rayo.getPosicion().Y, (int)(Rayo.getTextura().Width * 0.2), (int)(Rayo.getTextura().Height * 0.2));

            //Chequea de enemigo en enemigo si el cohete lo colisiona
            foreach (ClaseChatarra Chatarra in Chatarras)
            {
                //Deduce el rectángulo del enemigo y debe tener en cuenta el tamaño con que se esté dibujando en pantalla en Draw()

```

```

Rectangle RectEnemigo = new Rectangle((int)Chatarra.getPosicion().X, (int)Chatarra.getPosicion().Y,
(int)(Chatarra.getTextura().Width), (int)(Chatarra.getTextura().Height));

//Si ambos rectángulos se intersectan
if (RectRayo.Intersects(RectEnemigo) == true)
{
    Chatarra.setActivo(false); //inactiva el enemigo
    Rayo.setActivo(false); //inactiva el rayo
    Puntaje++;

    //Sonido de la explosión
    BancoSonido.PlayCue("explosion");

    break;
}

//Si el rayo se sale de la pantalla entonces se desactiva
if (RectFondo.Contains(new Point((int)Rayo.getPosicion().X, (int)Rayo.getPosicion().Y)) == false)
    Rayo.setActivo(false);
}

//Método llamado desde Update
public void ActualizaEnemigos()
{
    float AlturaMax = 0.1f; //No pegado del techo
    float AlturaMin = 0.5f; //La mitad de la pantalla
    float VelocidadMax = 5.0f; //Máxima velocidad
    float VelocidadMin = 1.0f; //Mínima velocidad

    foreach (ClaseChatarra Chatarra in Chatarras)
    {
        if (Chatarra.getActivo() == true)
        {
            //El enemigo avanza a determinada velocidad
            Chatarra.setPosicion(Chatarra.getPosicion() + Chatarra.getVelocidad());

            //Si el enemigo se sale de la pantalla
            if (RectFondo.Contains(new Point((int)Chatarra.getPosicion().X, (int)Chatarra.getPosicion().Y)) == false)
                Chatarra.setActivo(false);
        }
        else
        {
            //Activa el enemigo
            Chatarra.setActivo(true);

            //Interpolación. Posición eje Y aleatoria.
            Chatarra.setPosicion(new Vector2(RectFondo.Right, MathHelper.Lerp((float)RectFondo.Height * AlturaMin,
(float)RectFondo.Height * AlturaMax, (float)Aleatorio.NextDouble())));

            //Velocidad del enemigo. Aleatoria.
            Chatarra.setVelocidad(new Vector2(MathHelper.Lerp(-VelocidadMin, -VelocidadMax, (float)Aleatorio.NextDouble()), 0));
        }
    }
}
}

```

## 27. XNA: Mejorando los tipos de letra para mostrar texto en el juego

Para tener mejores fuentes (tipos de letra) en nuestro juego hay un paquete llamado SpriteFont que podemos descargarlo de esta ubicación: <http://www.nubik.com/SpriteFont/>

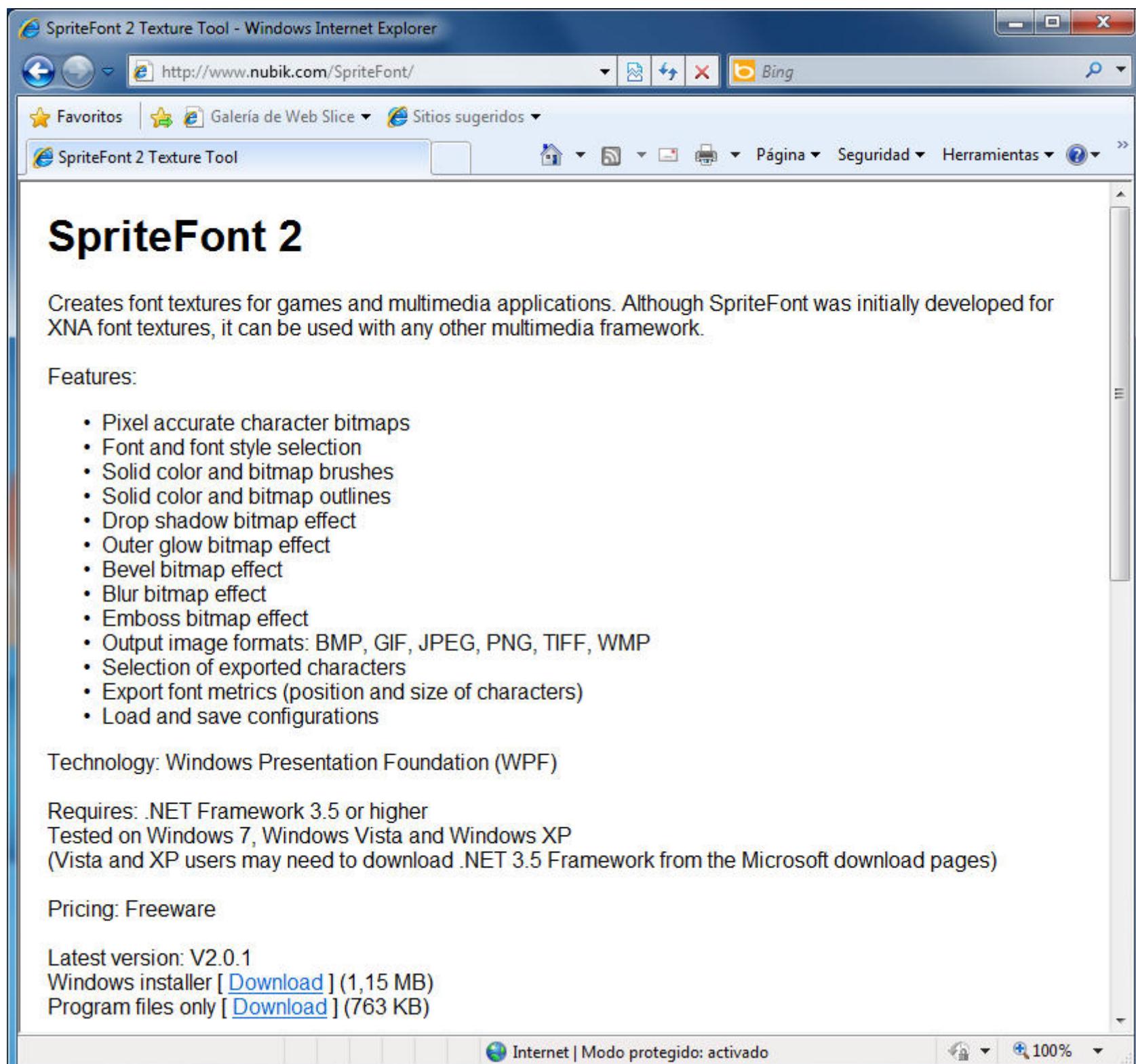


Figura 148: El paquete SpriteFont y su sitio oficial

Ejecutamos SpriteFont.exe



Figura 149: Pantalla inicial de SpriteFont

En la parte superior está el menú de opciones, recomendado cambiar la frase de ejemplo que trae por una propia

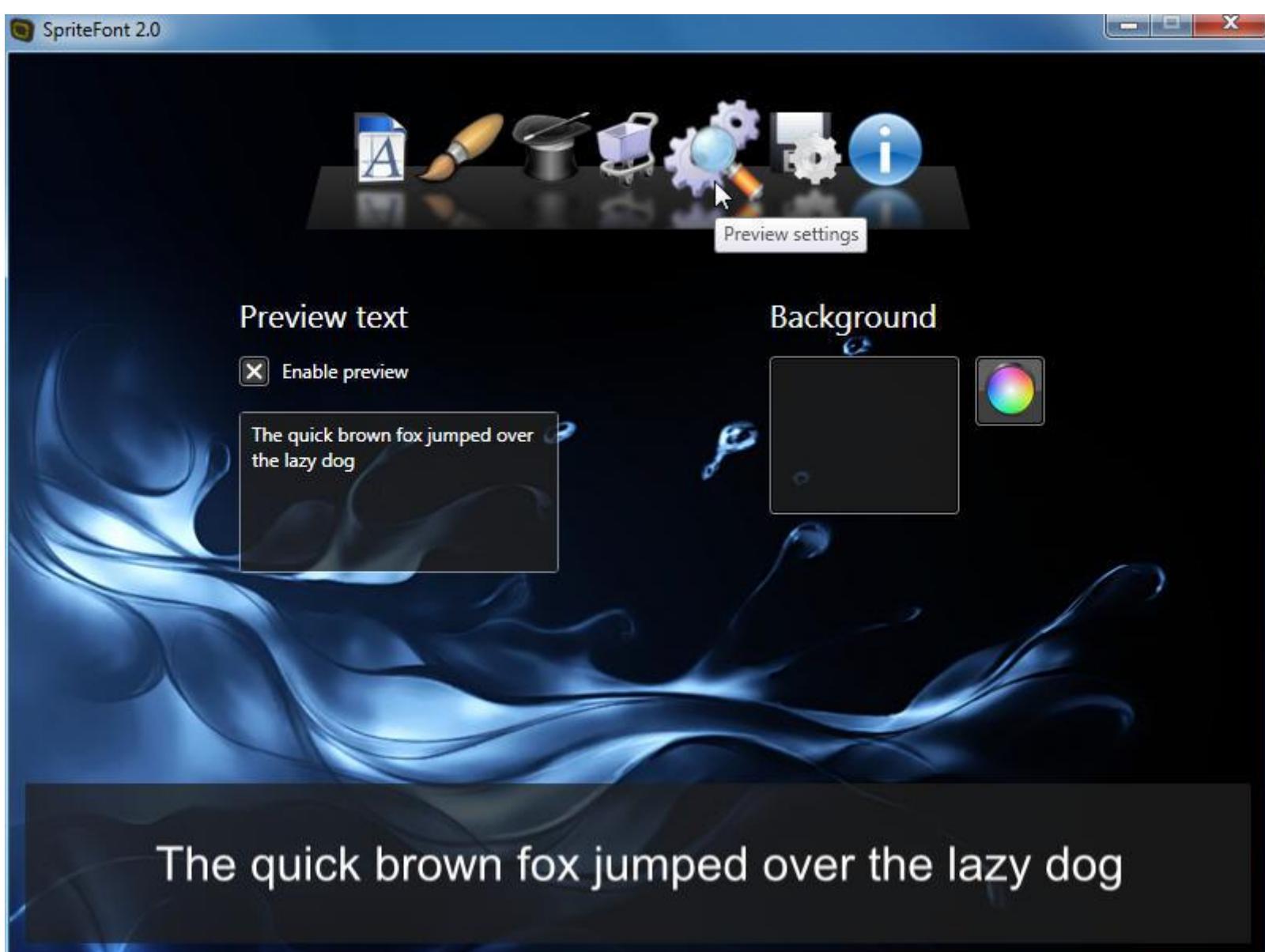


Figura 150: Cambiar la frase de ejemplo



Figura 151: Nueva frase personalizada para ver como quedará la fuente

Podemos cambiar la fuente



Figura 152: "Brush Settings" se cambian los colores de las letras

En "Brush Settings" podemos cambiar los colores de las letras



Figura 153: "Brush Settings" y sus pruebas en la fuente

En "Effects" podemos poner sombras, brillos, etc.. a las letras



Figura 154: "Effects" prueba diversos efectos sobre la fuente



Figura 155: "Outer glow"



Figura 156: "Bevel"

Finalmente exportamos las letras a un archivo plano



Figura 157: Se exporta la fuente para poder ser usada en nuestro proyecto

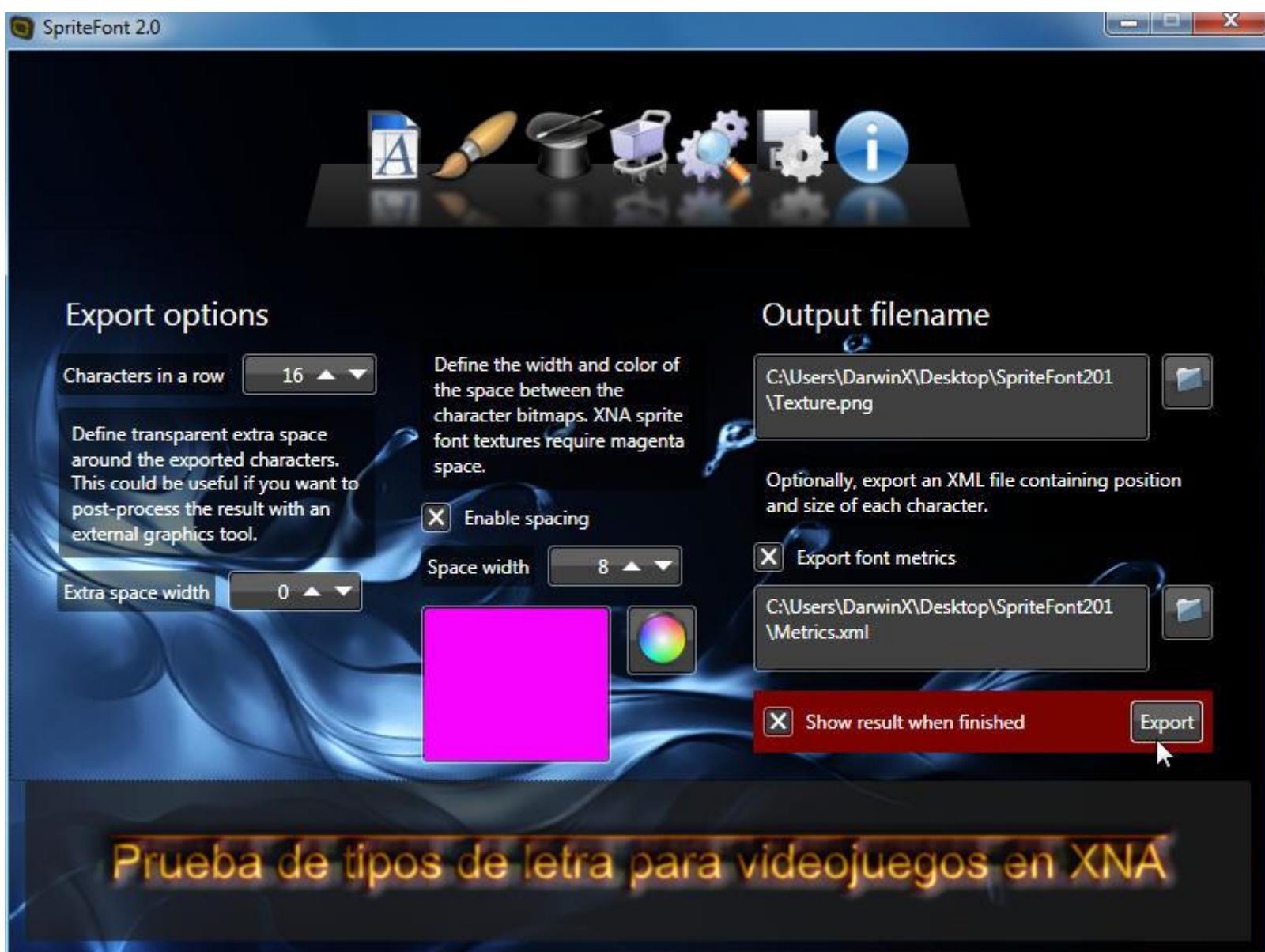
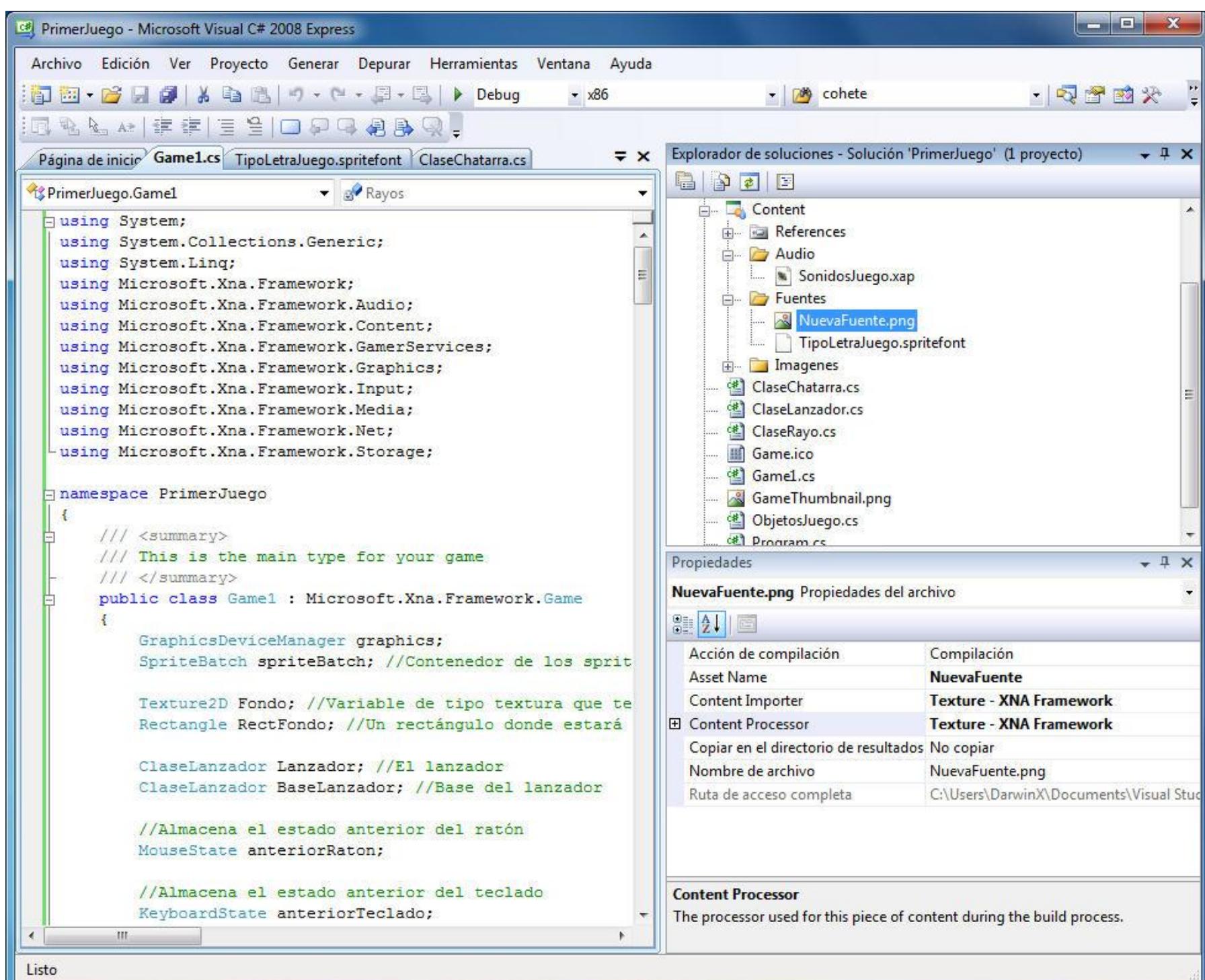


Figura 158: Ajustes para la exportación a un archivo .PNG



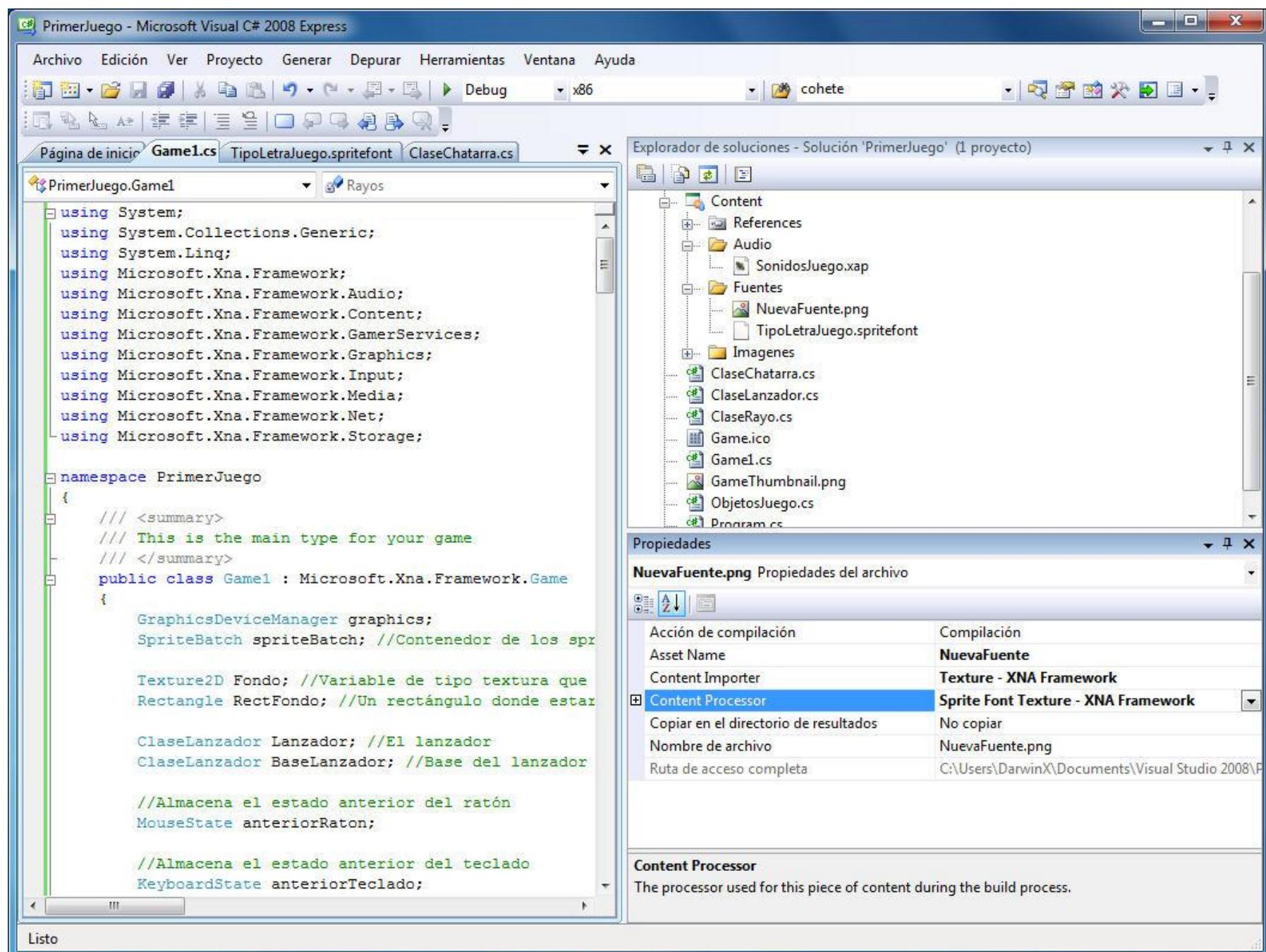
**Figura 159:** Esta es la fuente obtenida en un archivo .PNG para ser usada en un proyecto en XNA

En nuestro proyecto de juego en C#, vamos por el explorador de soluciones a la carpeta "Content" y ubicamos la subcarpeta Fuentes que habíamos creado anteriormente, se borra lo que había allí y luego se arrastra la imagen .PNG que generó el SpriteFont



**Figura 160:** Ubicamos el archivo .PNG en la carpeta fuentes de recursos del videojuego

Luego en las propiedades de esa nueva fuente debemos decirle a XNA en Content Processor que FuenteJuego.png es un objeto de tipo "Sprite Font Texture - XNA Framework"



**Figura 161: Ese recurso tendrá la propiedad "Content Processor" como "Sprite Font Texture"**

Luego es solo cambiar una sola línea en el código fuente (método LoadContent() ) de nuestro juego para que tome el nuevo tipo de letra

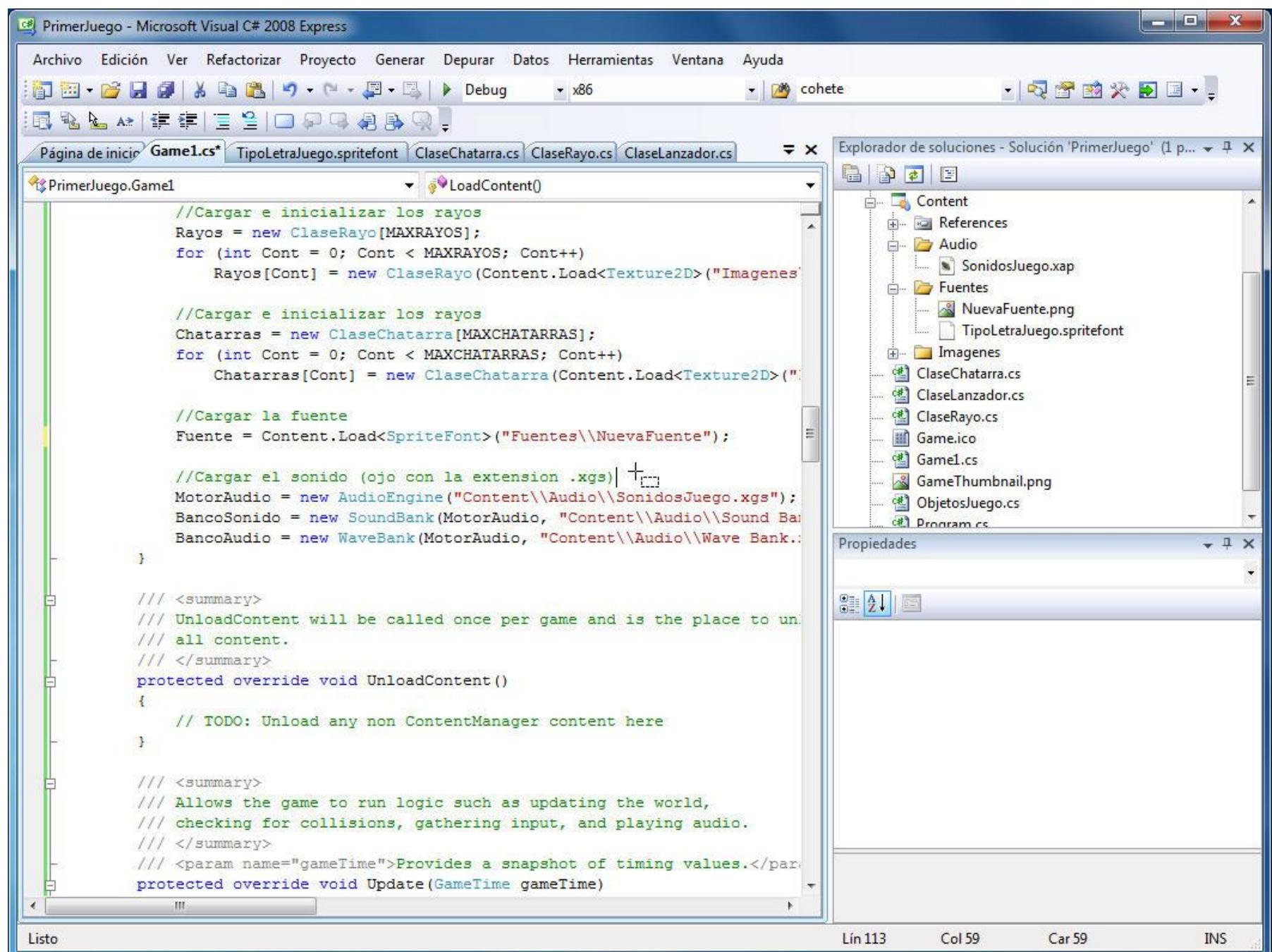


Figura 162: La nueva fuente se ha instalado y cargado en el método LoadContent()

Y al ejecutar este es el resultado:

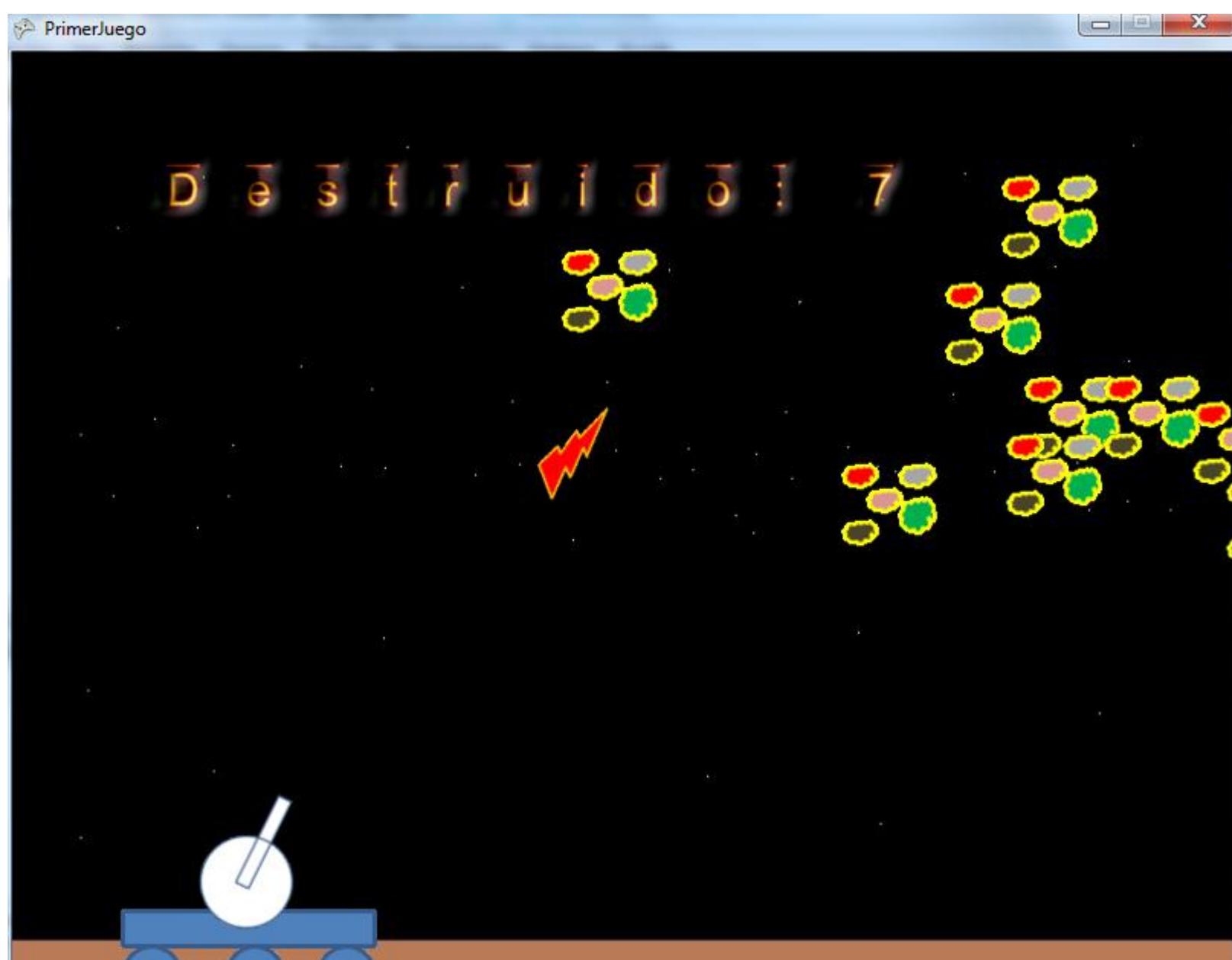
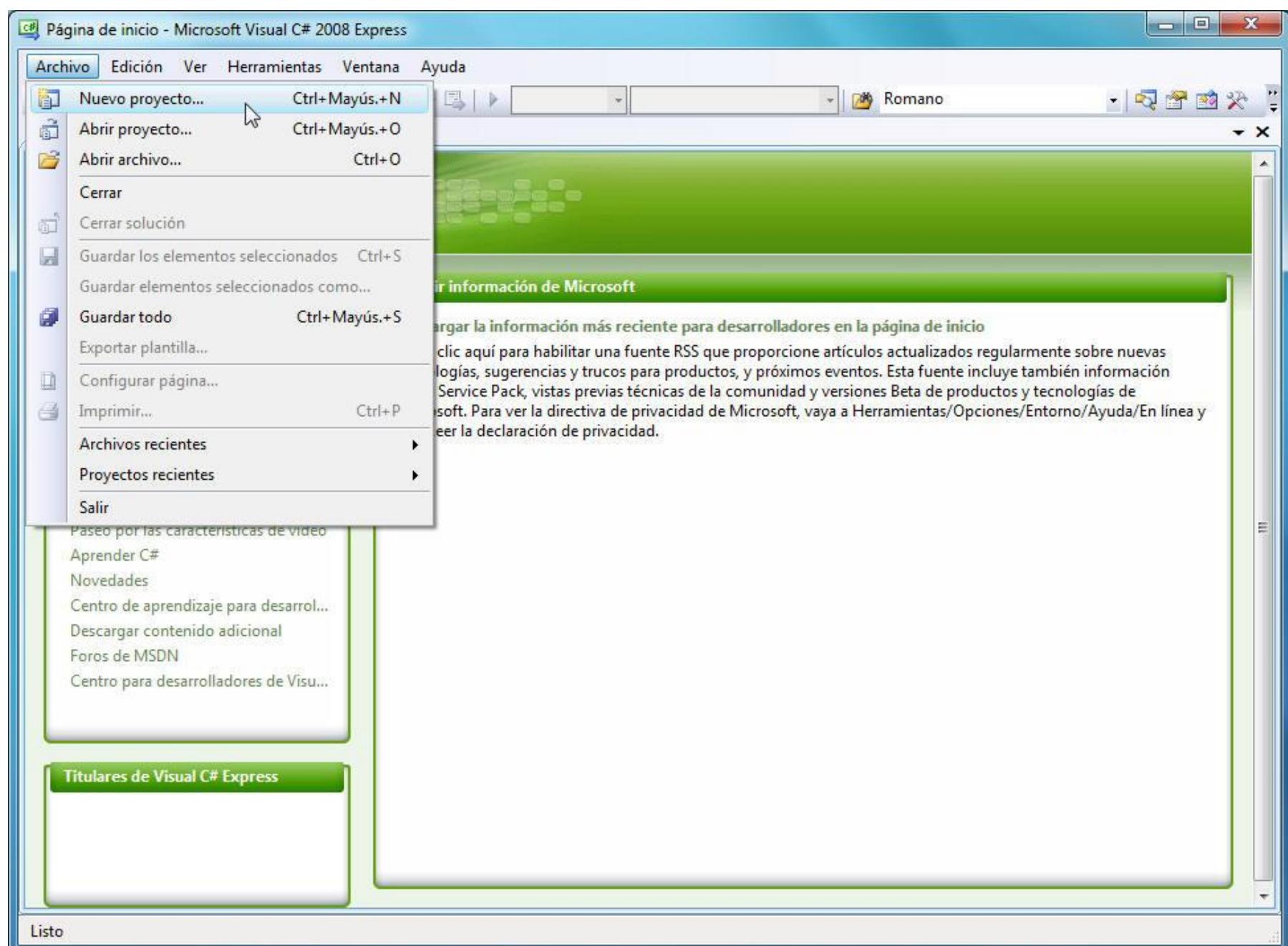


Figura 163: Así luce el juego con la nueva fuente

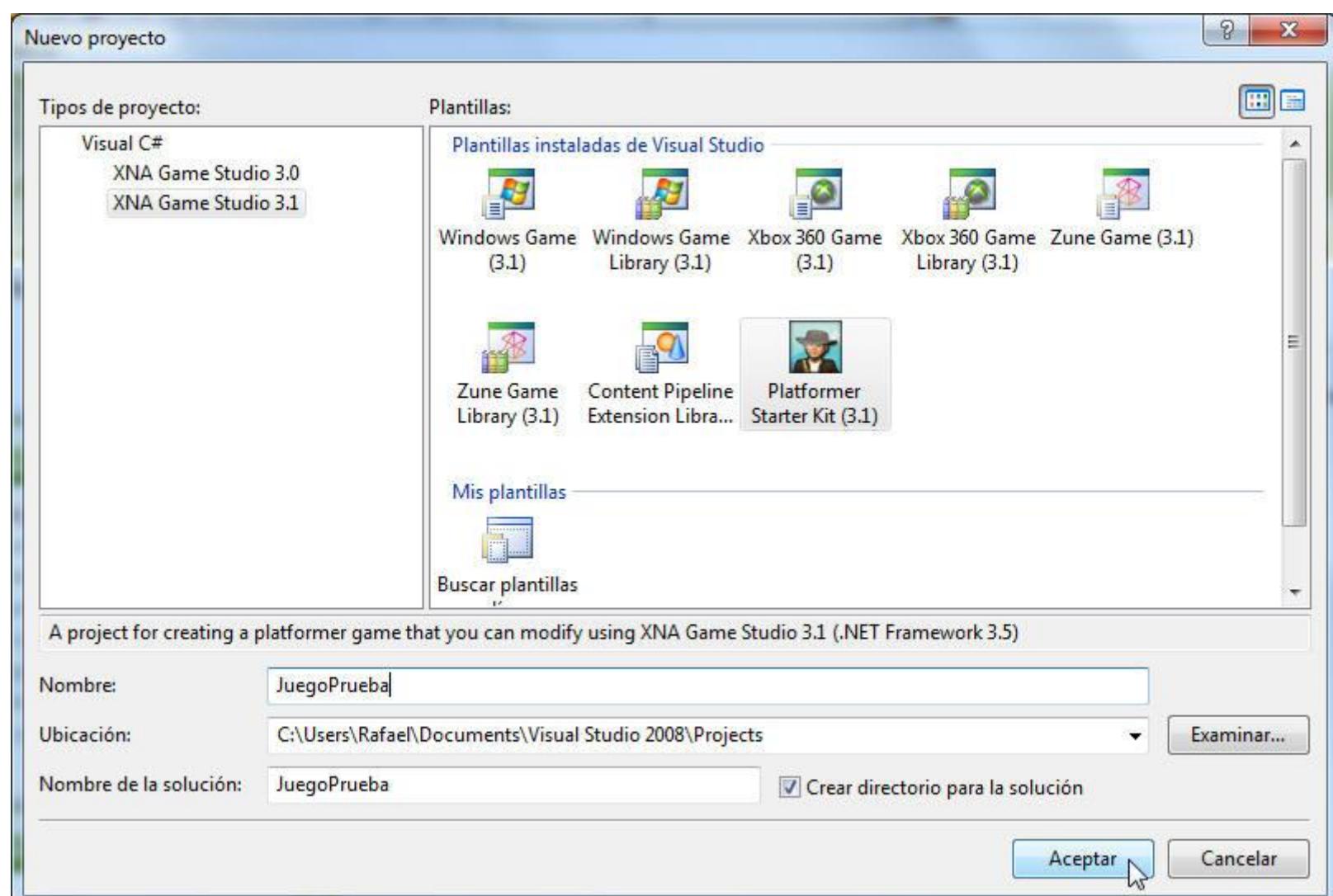
## 28. XNA: Platformer Starter Kit 3.1

Al instalar el XNA Game Studio, este viene con un juego de prueba, el cual nos permite ver las diversas características en 2D que se pueden hacer con este "framework". Es un juego incompleto a propósito.



**Figura 164: Generamos un nuevo proyecto**

Seleccionamos Platformer Starter Kit (3.1) y le damos un nombre



**Figura 165: Seleccionamos "Platformer Starter Kit (3.1)"**

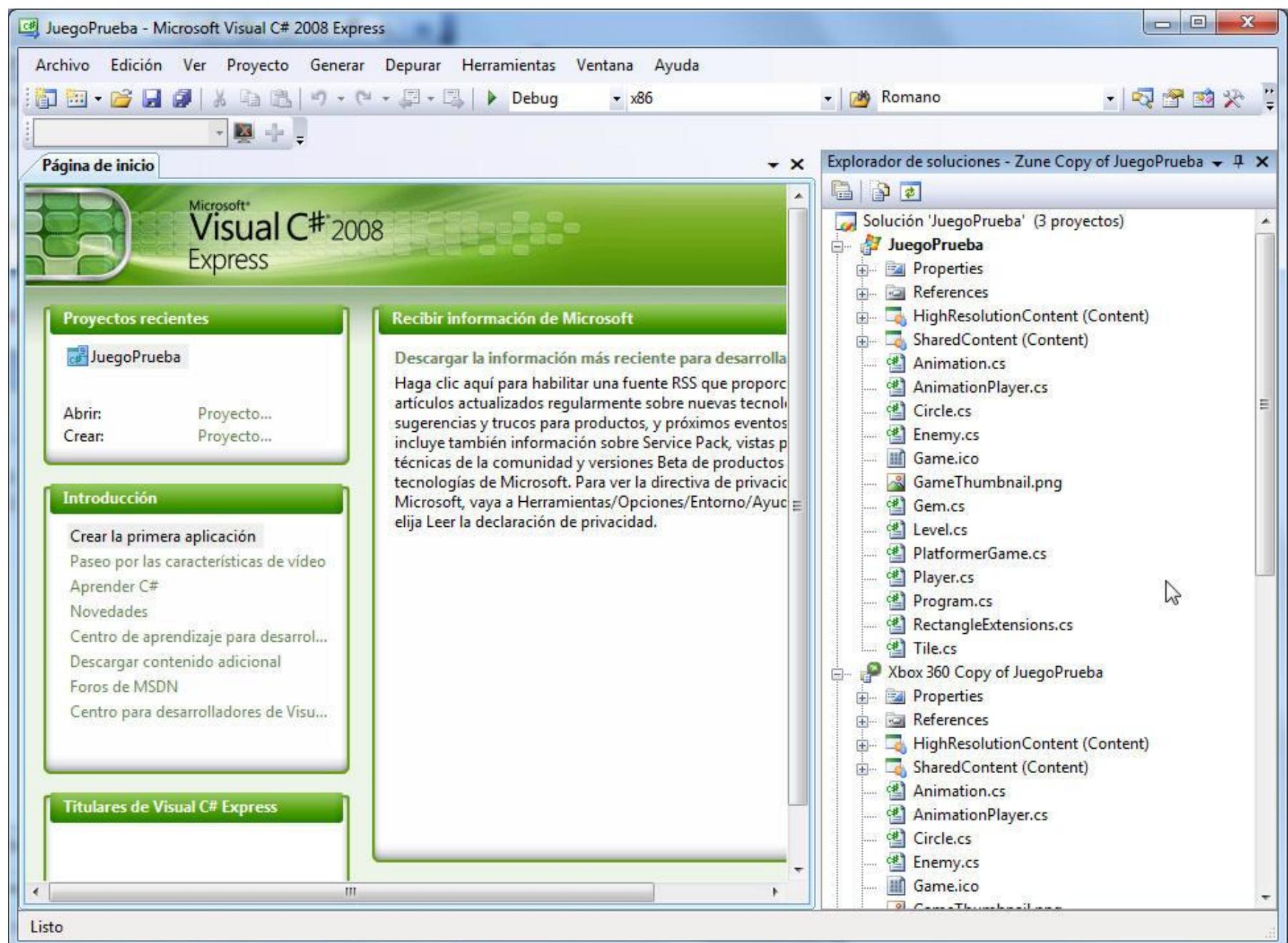


Figura 166: Proyecto generado

Estas son pantallas del juego. El protagonista camina, salta y es atacado por enemigos. El juego tiene música de fondo, efectos de sonido dependiendo de lo que le ocurra al protagonista.



Figura 167: Ejecución del juego de ejemplo

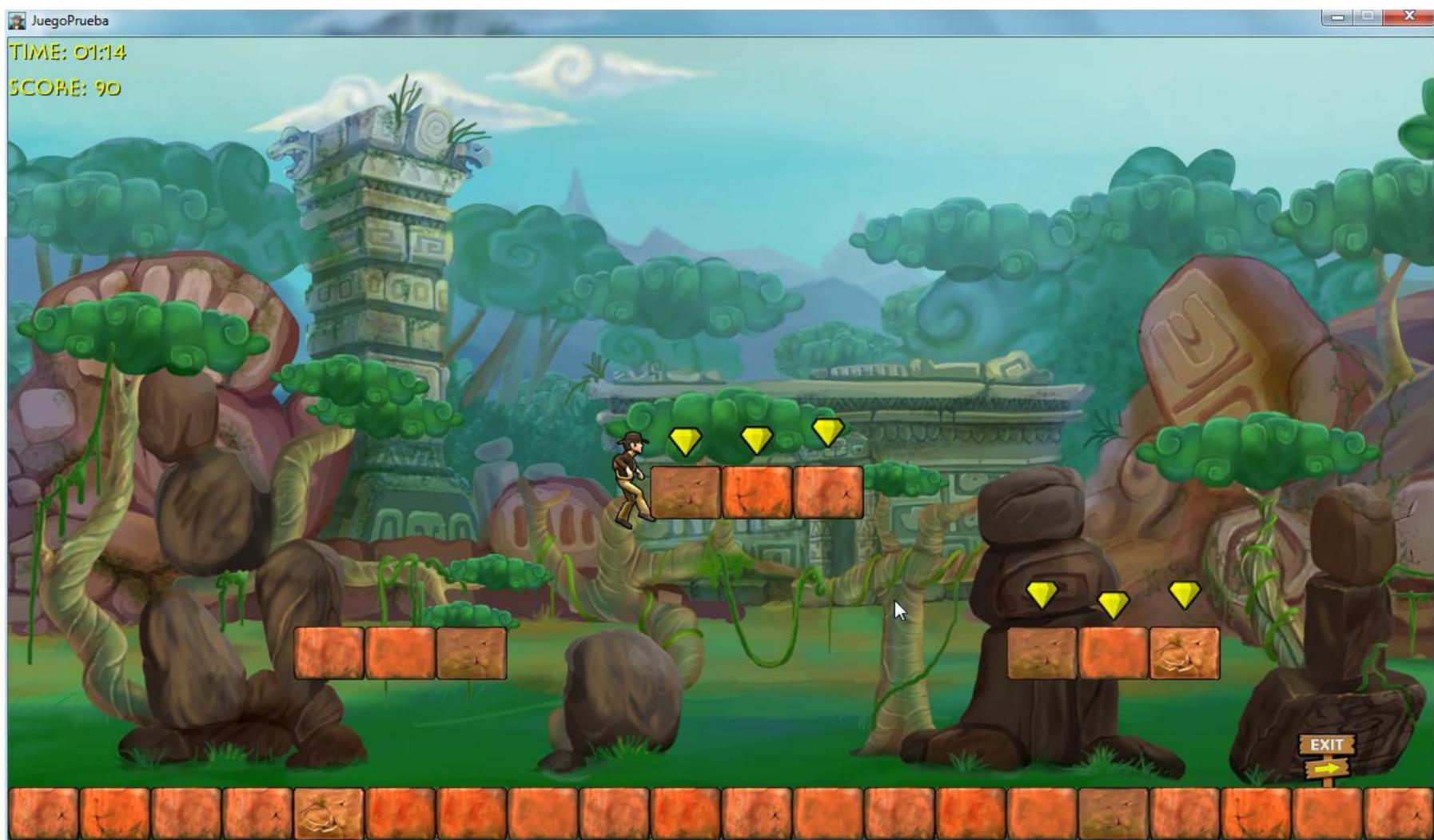


Figura 168: Protagonista puede hacer diversas maniobras y hay varias pantallas

¿Para qué nos sirve este juego?

Ver diversas técnicas para implementar animación, sonido, música de fondo y hasta historia.

En el explorador de soluciones podemos ver como está el juego hecho por dentro.

## 29. XNA: Animando un personaje del juego

En un videojuego, un personaje como un ser humano, al desplazarse, lo hace caminando o corriendo o saltando. Entonces debemos programar este comportamiento. Los pasos para hacer esto son:

**Paso 1:** Dibujar la animación en una sola imagen, como se ve a continuación. Esta imagen fue tomada de Platformer Starter Kit 3.1



Figura 169: Secuencia donde se muestra al protagonista corriendo

**Paso 2:** Crear un nuevo proyecto de videojuego desde cero

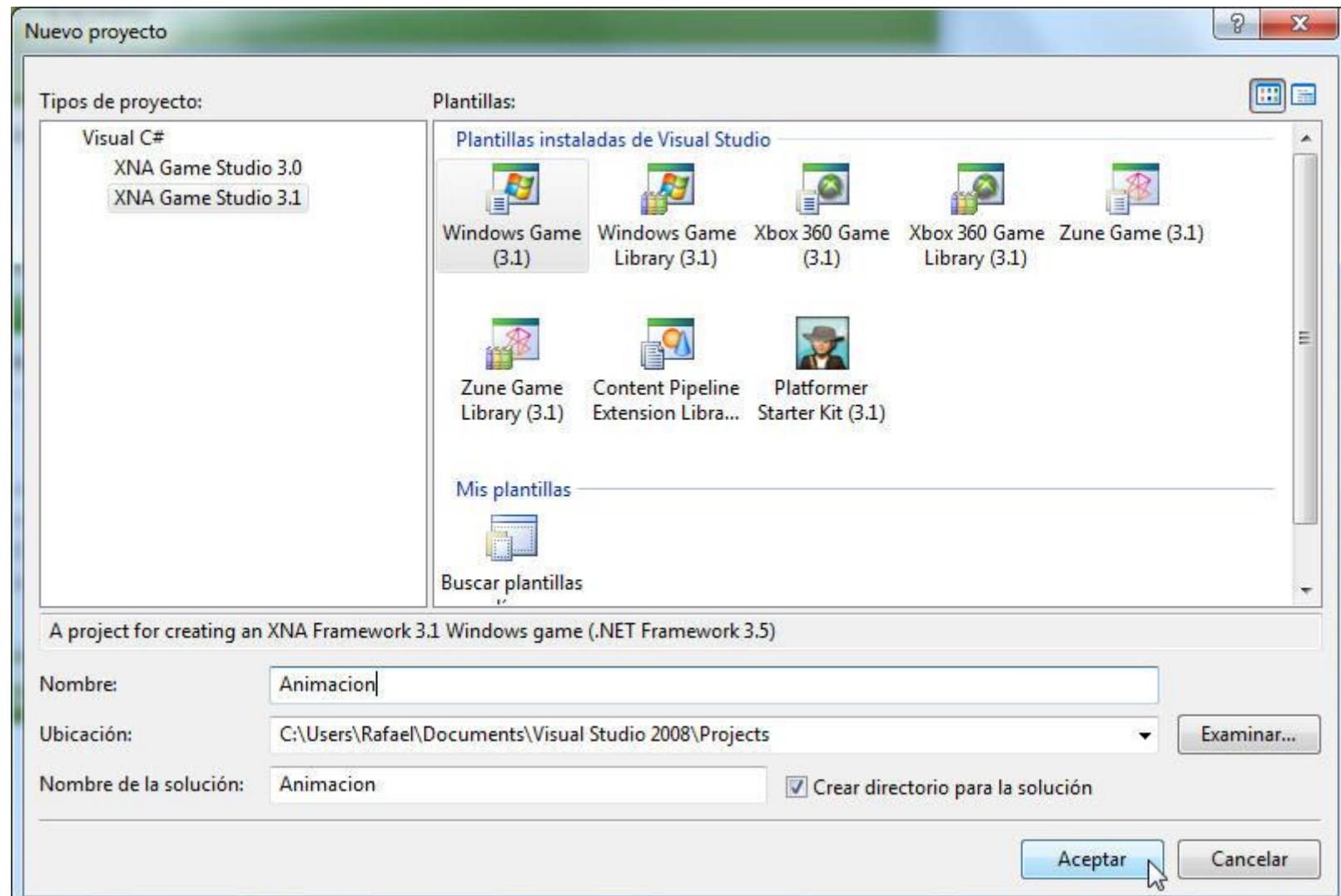
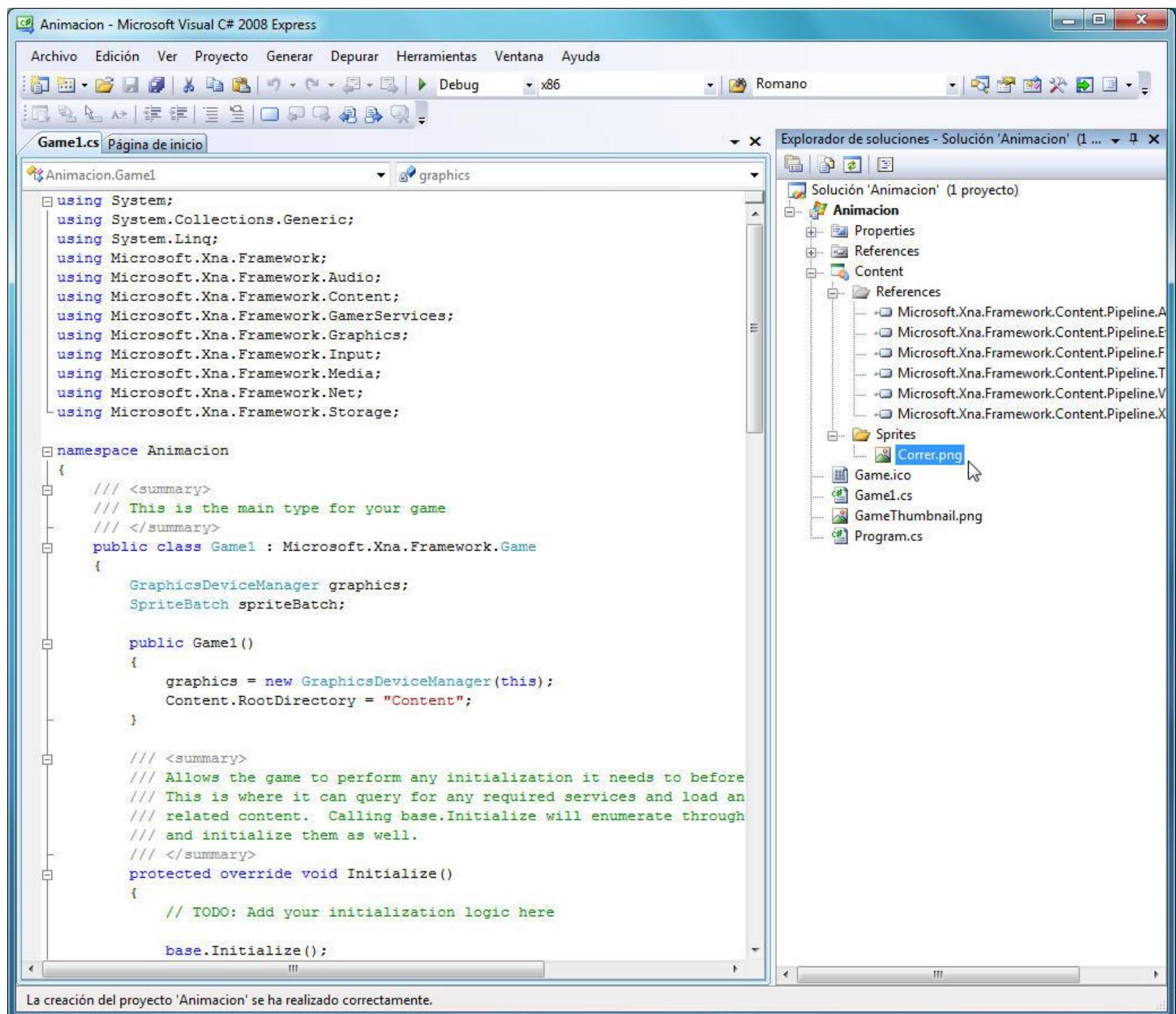


Figura 170: Generamos un nuevo proyecto tipo "Windows Game (3.1)"

**Paso 3:** Agregar la imagen al proyecto



**Figura 171: Ubicamos la secuencia como recurso imagen dentro del juego**

**Paso 4:** Ahora agregamos estas instrucciones al código. Son variables de clase.

```
//Personaje Animado
Texture2D PersonajeAnimado;

//Total de dibujos que tiene el personaje animado
int TotalDibujosFigura = 10;
int DibujoFiguraActual = 0;

//Tamaño del rectángulo que muestra un "frame" del personaje animado
int rectPersonajeAncho = 96;
int rectPersonajeAlto = 96;

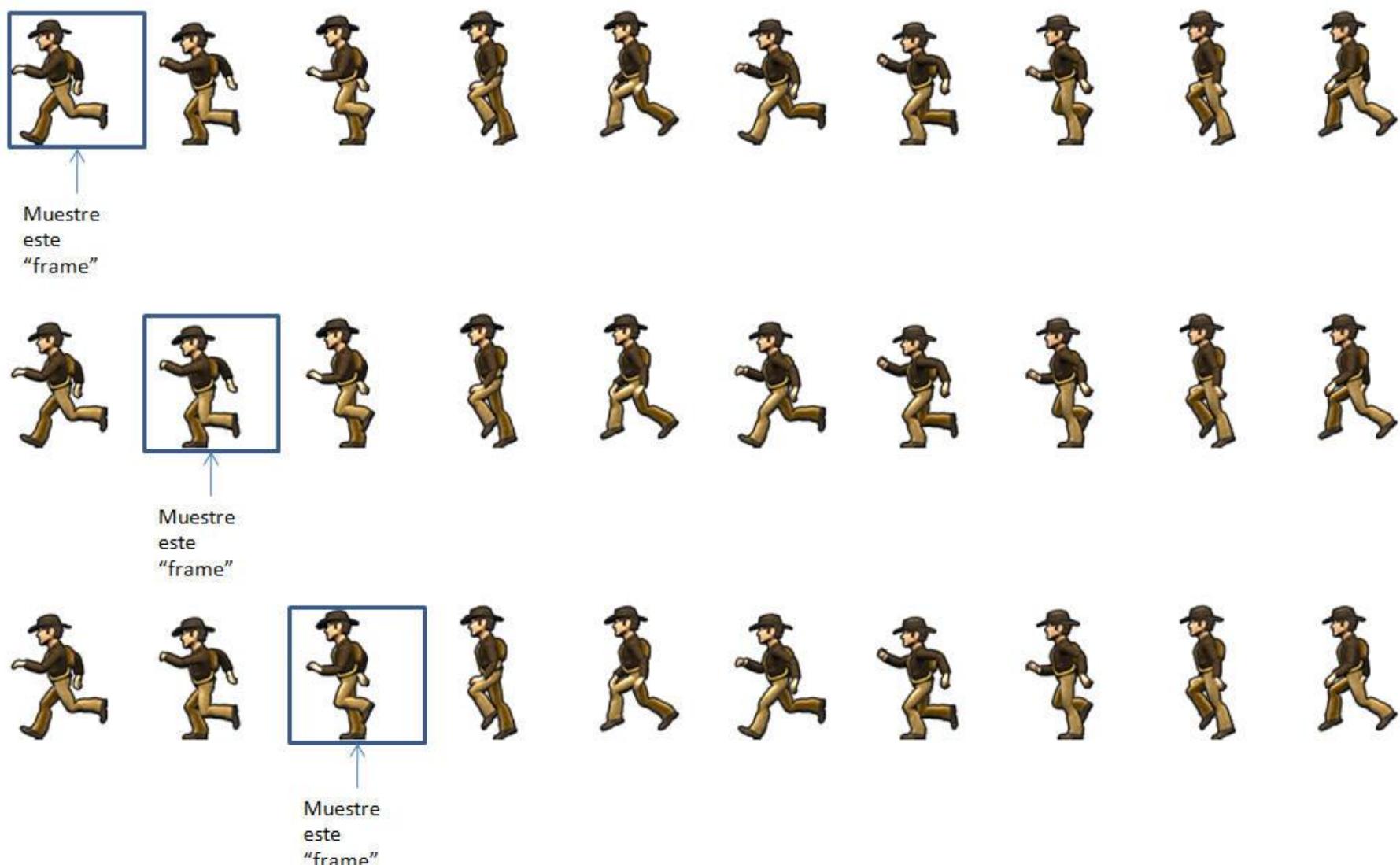
//Controla el cambio de un "frame" a otro en el tiempo
float CuentaTiempo = 0f;
float CadaCuantoCambia = 100f;

//Rectángulo que ubica que "frame" del archivo .PNG se dibuja
Rectangle rectOrigenPersonaje;
```

**Paso 5:** Cargamos la imagen en LoadContent()

```
PersonajeAnimado = Content.Load<Texture2D>("Sprites\\Correr");
```

**Paso 6:** Y esta es la parte de la animación. El truco es mostrar un dibujo, luego otro y otro, así sucesivamente dando la sensación que hay movimiento. ¿Y cómo hacemos eso?. Si observamos, la imagen que se carga (la del explorador corriendo), lo que debemos hacer es ubicar un rectángulo que cubra la primera figura y el contenido de ese rectángulo es el que se muestra. Luego se desplaza el rectángulo a la segunda figura y se muestra esa, así sucesivamente hasta que termine en la última figura y vuelve a la figura inicial en un ciclo sin fin.



**Figura 172: Como trabajará el algoritmo para mostrar la secuencia**

Esa lógica debe ir, por supuesto, en Update()

```
//Acumula los milisegundos que lleva ejecutando el juego
CuentaTiempo += (float)gameTime.ElapsedGameTime.TotalMilliseconds;

//Si el acumulado pasa de determinado tiempo
if (CuentaTiempo > CadaCuantoCambia)
{
    //Pasa al siguiente "frame"
    DibujoFiguraActual++;

    //Se controla no pasarse de todos los "frames" del personaje
    if (DibujoFiguraActual > TotalDibujosFigura - 1)
        DibujoFiguraActual = 0;

    //Acumulador de tiempo a cero
    CuentaTiempo = 0f;
}

//Determina donde ubicar el rectángulo para el siguiente "frame"
rectOrigenPersonaje = new Rectangle(this.DibujoFiguraActual * this.rectPersonajeAncho, 0, this.rectPersonajeAncho,
this.rectPersonajeAlto);
```

#### Paso 7: Dibujar la animación en Draw()

```
spriteBatch.Begin();

/* Los parámetros son:
* 1. Textura a cargar (todo el archivo .png que tiene la animación).
* 2. Posición donde colocar el sprite "animado"
* 3. Rectángulo origen. Carga el "frame" a mostrar que luego es dibujado en el rectángulo destino
* 4. Color de fondo
* 5. Rotación
* 6. Origen
* 7. Escala
* 8. Efecto del sprite. Reflejo horizontal.
* 9. Profundidad en el que se dibuja */
Vector2 Origen = new Vector2(0f, 0f);
Vector2 Posicion = new Vector2(0f, 0f);
spriteBatch.Draw(PersonajeAnimado, Posicion, rectOrigenPersonaje, Color.White, 0.0f, Origen, 1.0f, SpriteEffects.None,
0.0f);

spriteBatch.End();
```

#### Paso 8: Este es el código completo

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
```

```

using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace Animacion
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        //Personaje Animado
        Texture2D PersonajeAnimado;

        //Total de dibujos que tiene el personaje animado
        int TotalDibujosFigura = 10;
        int DibujoFiguraActual = 0;

        //Tamaño del rectángulo que muestra un "frame" del personaje animado
        int rectPersonajeAncho = 96;
        int rectPersonajeAlto = 96;

        //Controla el cambio de un "frame" a otro en el tiempo
        float CuentaTiempo = 0f;
        float CadaCuantoCambia = 100f;

        //Rectángulo que ubica que "frame" del archivo .PNG se dibuja
        Rectangle rectOrigenPersonaje;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // TODO: use this.Content to load your game content here
            PersonajeAnimado = Content.Load<Texture2D>("Sprites\\Correr");
        }

        /// <summary>
        /// UnloadContent will be called once per game and is the place to unload
        /// all content.
        /// </summary>
        protected override void UnloadContent()
        {
            // TODO: Unload any non ContentManager content here
        }

        /// <summary>
        /// Allows the game to run logic such as updating the world,
        /// checking for collisions, gathering input, and playing audio.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Update(GameTime gameTime)
        {
            // Allows the game to exit
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
                this.Exit();

            // TODO: Add your update logic here

            //Acumula los milisegundos que lleva ejecutando el juego
            CuentaTiempo += (float)gameTime.ElapsedGameTime.TotalMilliseconds;
        }
    }
}

```

```

//Si el acumulado pasa de determinado tiempo
if (CuentaTiempo > CadaCuantoCambia)
{
    //Pasa al siguiente "frame"
    DibujoFiguraActual++;

    //Se controla no pasarse de todos los "frames" del personaje
    if (DibujoFiguraActual > TotalDibujosFigura - 1)
        DibujoFiguraActual = 0;

    //Acumulador de tiempo a cero
    CuentaTiempo = 0f;
}

//Determina donde ubicar el rectángulo para el siguiente "frame"
rectOrigenPersonaje = new Rectangle(this.DibujoFiguraActual * this.rectPersonajeAncho, 0, this.rectPersonajeAncho,
this.rectPersonajeAlto);

base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here
    spriteBatch.Begin();

    /* Los parámetros son:
    * 1. Textura a cargar (todo el archivo .png que tiene la animación).
    * 2. Posición donde colocar el sprite "animado"
    * 3. Rectángulo origen. Carga el "frame" a mostrar que luego es dibujado en el rectángulo destino
    * 4. Color de fondo
    * 5. Rotación
    * 6. Origen
    * 7. Escala
    * 8. Efecto del sprite. Reflejo horizontal.
    * 9. Profundidad en el que se dibuja */
    Vector2 Origen = new Vector2(0f, 0f);
    Vector2 Posicion = new Vector2(0f, 0f);
    spriteBatch.Draw(PersonajeAnimado, Posicion, rectOrigenPersonaje, Color.White, 0.0f, Origen, 1.0f, SpriteEffects.None,
0.0f);

    spriteBatch.End();

    base.Draw(gameTime);
}
}
}

```



Figura 173: Ejemplo de ejecución de la secuencia



Figura 174: Ejemplo de ejecución de la secuencia

## 30. XNA: Cambiando la orientación de la animación del personaje del juego

Veamos la animación original.



**Figura 175: Secuencia de animación del personaje corriendo.**

En Draw() solo cambiamos la instrucción SpriteEffects.None por SpriteEffects.FlipHorizontally

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here
    spriteBatch.Begin();

    /* Los parámetros son:
     * 1. Textura a cargar (todo el archivo .png que tiene la animación).
     * 2. Posición donde colocar el sprite "animado"
     * 3. Rectángulo origen. Carga el "frame" a mostrar que luego es dibujado en el rectángulo destino
     * 4. Color de fondo
     * 5. Rotación
     * 6. Origen
     * 7. Escala
     * 8. Efecto del sprite. Reflejo horizontal.
     * 9. Profundidad en el que se dibuja */
    Vector2 Origen = new Vector2(0f, 0f);
    Vector2 Posicion = new Vector2(0f, 0f);
    spriteBatch.Draw(PersonajeAnimado, Posicion, rectOrigenPersonaje, Color.White, 0.0f, Origen, 1.0f,
        SpriteEffects.FlipHorizontally, 0.0f);

    spriteBatch.End();

    base.Draw(gameTime);
}
```

Este es el código completo

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace Animacion
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        //Personaje Animado
        Texture2D PersonajeAnimado;

        //Total de dibujos que tiene el personaje animado
        int TotalDibujosFigura = 10;
        int DibujoFiguraActual = 0;

        //Tamaño del rectángulo que muestra un "frame" del personaje animado
        int rectPersonajeAncho = 96;
        int rectPersonajeAlto = 96;

        //Controla el cambio de un "frame" a otro en el tiempo
        float CuentaTiempo = 0f;
        float CadaCuantoCambia = 100f;

        //Rectángulo que ubica que "frame" del archivo .PNG se dibuja
        Rectangle rectOrigenPersonaje;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }
}
```

```

/// <summary>
/// Allows the game to perform any initialization it needs to before starting to run.
/// This is where it can query for any required services and load any non-graphic
/// related content. Calling base.Initialize will enumerate through any components
/// and initialize them as well.
/// </summary>
protected override void Initialize()
{
    // TODO: Add your initialization logic here

    base.Initialize();
}

/// <summary>
/// LoadContent will be called once per game and is the place to load
/// all of your content.
/// </summary>
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here
    PersonajeAnimado = Content.Load<Texture2D>("Sprites\\Correr");
}

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // TODO: Add your update logic here

    //Acumula los milisegundos que lleva ejecutando el juego
    CuentaTiempo += (float)gameTime.ElapsedGameTime.TotalMilliseconds;

    //Si el acumulado pasa de determinado tiempo
    if (CuentaTiempo > CadaCuantoCambia)
    {
        //Pasa al siguiente "frame"
        DibujoFiguraActual++;

        //Se controla no pasarse de todos los "frames" del personaje
        if (DibujoFiguraActual > TotalDibujosFigura - 1)
            DibujoFiguraActual = 0;

        //Acumulador de tiempo a cero
        CuentaTiempo = 0f;
    }

    //Determina donde ubicar el rectángulo para el siguiente "frame"
    rectOrigenPersonaje = new Rectangle(this.DibujoFiguraActual * this.rectPersonajeAncho, 0, this.rectPersonajeAncho,
                                         this.rectPersonajeAlto);

    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here
    spriteBatch.Begin();

    /* Los parámetros son:
     * 1. Textura a cargar (todo el archivo .png que tiene la animación).
     * 2. Posición donde colocar el sprite "animado"
     * 3. Rectángulo origen. Carga el "frame" a mostrar que luego es dibujado en el rectángulo destino
     * 4. Color de fondo
     * 5. Rotación
     * 6. Origen
     * 7. Escala
     * 8. Efecto del sprite. Reflejo horizontal.
     * 9. Profundidad en el que se dibuja */
}

```

```
Vector2 Origen = new Vector2(0f, 0f);
Vector2 Posicion = new Vector2(0f, 0f);
spriteBatch.Draw(PersonajeAnimado, Posicion, rectOrigenPersonaje, Color.White, 0.0f, Origen, 1.0f,
SpriteEffects.FlipHorizontally, 0.0f);

spriteBatch.End();

base.Draw(gameTime);
}
}
```

Y el resultado es una figura en movimiento corriendo ahora a la derecha



Figura 176: Ejemplo de personaje corriendo en otro sentido



Figura 177: Ejemplo de personaje corriendo en otro sentido

## 31. XNA: Cambiando la orientación de la animación del personaje del juego con el teclado

Durante el juego queremos que si el usuario presiona la flecha izquierda el protagonista corra a la izquierda, y por supuesto, si el usuario presiona la flecha derecha el protagonista corra a la derecha. ¿Cómo hacemos eso?

**Paso 1:** Creamos una variable de clase llamada Dirección de tipo SpriteEffect

```
//En que dirección va corriendo el protagonista
SpriteEffects Direccion = SpriteEffects.None;
```

**Paso 2:** En la rutina Draw() cambiamos la instrucción SpriteEffects.FlipHorizontally (que teníamos anteriormente) por la variable Direccion

```
spriteBatch.Draw(PersonajeAnimado, Posicion, rectOrigenPersonaje, Color.White, 0.0f, Origen, 1.0f, Direccion, 0.0f);
```

**Paso 3:** En la rutina Update() capturamos los golpes de tecla del usuario

```
#if !XBOX
    //Código para leer el teclado
    KeyboardState estadoTeclado = Keyboard.GetState();
    if (estadoTeclado.IsKeyDown(Keys.Left)) Direccion = SpriteEffects.None; //Protagonista avanza a la izquierda
    if (estadoTeclado.IsKeyDown(Keys.Right)) Direccion = SpriteEffects.FlipHorizontally; //Protagonista avanza a la derecha
#endif
```

Este es el código completo

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace Animacion
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        //Personaje Animado
        Texture2D PersonajeAnimado;

        //Total de dibujos que tiene el personaje animado
        int TotalDibujosFigura = 10;
        int DibujoFiguraActual = 0;

        //Tamaño del rectángulo que muestra un "frame" del personaje animado
        int rectPersonajeAncho = 96;
        int rectPersonajeAlto = 96;

        //Controla el cambio de un "frame" a otro en el tiempo
        float CuentaTiempo = 0f;
        float CadaCuantoCambia = 100f;

        //Rectángulo que ubica que "frame" del archivo .PNG se dibuja
        Rectangle rectOrigenPersonaje;

        //En que dirección va corriendo el protagonista
        SpriteEffects Direccion = SpriteEffects.None;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here
            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
    }
}
```

```

/// all of your content.
/// </summary>
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here
    PersonajeAnimado = Content.Load<Texture2D>("Sprites\\Correr");
}

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // TODO: Add your update logic here

#if !XBOX
    //Código para leer el teclado
    KeyboardState estadoTeclado = Keyboard.GetState();
    if (estadoTeclado.IsKeyDown(Keys.Left)) Direccion = SpriteEffects.None; //Protagonista avanza a la izquierda
    if (estadoTeclado.IsKeyDown(Keys.Right)) Direccion = SpriteEffects.FlipHorizontally; //Protagonista avanza a la derecha
#endif

    //Acumula los milisegundos que lleva ejecutando el juego
    CuentaTiempo += (float)gameTime.ElapsedGameTime.TotalMilliseconds;

    //Si el acumulado pasa de determinado tiempo
    if (CuentaTiempo > CadaCuantoCambia)
    {
        //Pasa al siguiente "frame"
        DibujoFiguraActual++;

        //Se controla no pasarse de todos los "frames" del personaje
        if (DibujoFiguraActual > TotalDibujosFigura - 1)
            DibujoFiguraActual = 0;

        //Acumulador de tiempo a cero
        CuentaTiempo = 0f;
    }

    //Determina donde ubicar el rectángulo para el siguiente "frame"
    rectOrigenPersonaje = new Rectangle(this.DibujoFiguraActual * this.rectPersonajeAncho, 0, this.rectPersonajeAncho,
    this.rectPersonajeAlto);

    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here
    spriteBatch.Begin();

    /* Los parámetros son:
     * 1. Textura a cargar (todo el archivo .png que tiene la animación).
     * 2. Posición donde colocar el sprite "animado"
     * 3. Rectángulo origen. Carga el "frame" a mostrar que luego es dibujado en el rectángulo destino
     * 4. Color de fondo
     * 5. Rotación
     * 6. Origen
     * 7. Escala
     * 8. Efecto del sprite. Reflejo horizontal.
     * 9. Profundidad en el que se dibuja */
    Vector2 Origen = new Vector2(0f, 0f);
    Vector2 Posicion = new Vector2(0f, 0f);
    spriteBatch.Draw(PersonajeAnimado, Posicion, rectOrigenPersonaje, Color.White, 0.0f, Origen, 1.0f, Direccion, 0.0f);

    spriteBatch.End();

    base.Draw(gameTime);
}

```

```
[ }
```

## 32. XNA: Haciendo scroll (desplazamiento) del fondo de la pantalla.

En esta lección trataremos de como hacer un típico efecto de los juegos 2D: el desplazamiento (scroll) de la pantalla.

### Paso 1: Las variables de clase

```
Texture2D Fondo; //Objeto de tipo textura que tendrá el fondo
public Vector2 Posicion = new Vector2(0, 0); //Objeto de tipo Vector2 que controla la posición de la imagen.
```

### Paso 2: En LoadContent se carga el fondo

```
//Utiliza genéricos <> por eso el uso de <> En C++ se llaman plantillas. Content carga el contenido binario.
Fondo = Content.Load<Texture2D>("Fondo");
```

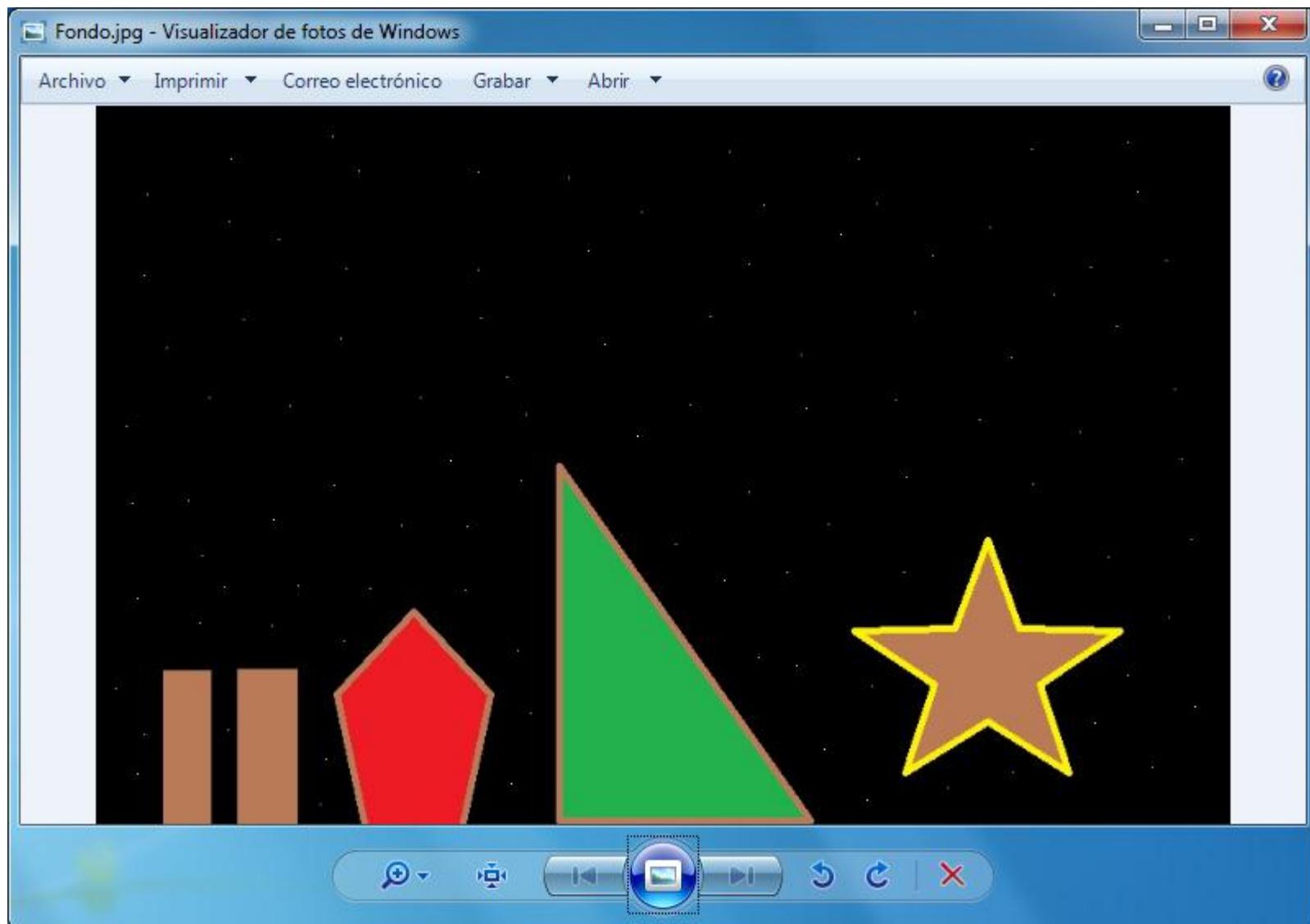


Figura 178: Imagen que se deslizará horizontalmente en el juego

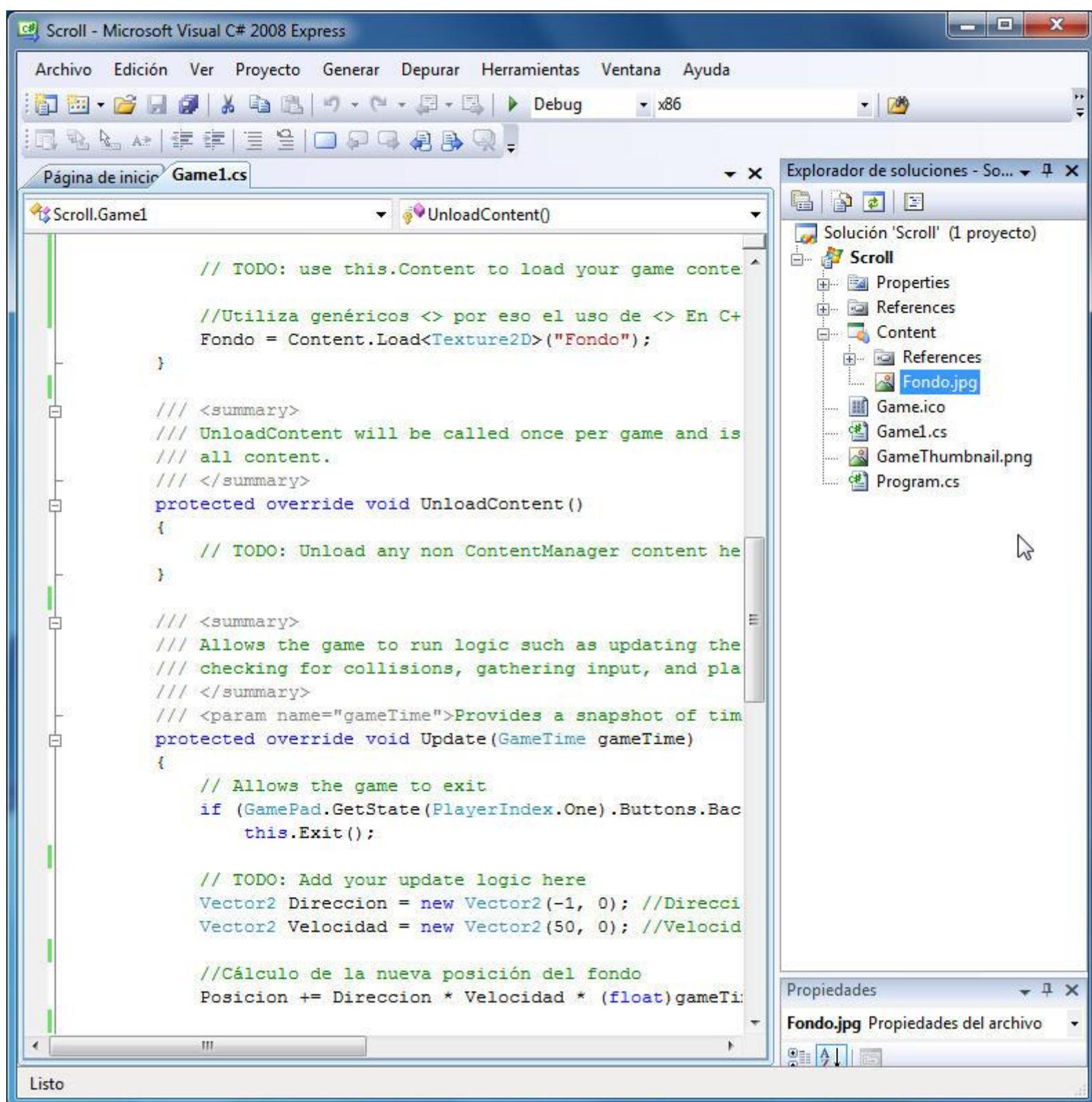


Figura 179: Se ubica esa imagen como recurso del videojuego

**Paso 3:** En Update, se calcula el valor de la nueva Posicion del fondo

```
// TODO: Add your update logic here
Vector2 Dirección = new Vector2(-1, 0); //Dirección a la que se desplaza el fondo
Vector2 Velocidad = new Vector2(50, 0); //Velocidad con la que se desplaza el fondo

//Cálculo de la nueva posición del fondo
Posición += Dirección * Velocidad * (float)gameTime.ElapsedGameTime.TotalSeconds;
```

**Paso 4:** Sólo es dibujar el fondo en su nueva posición

```
// TODO: Add your drawing code here
float Escala = 1.0f;

//Se inicia el spritebatch (Entre Begin y End se dibuja en cada unidad de tiempo los sprites)
spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

//Dibujar el fondo combinando el color blanco con el fondo
spriteBatch.Draw(Fondo, Posición,
new Rectangle(0, 0, Fondo.Width, Fondo.Height), Color.White,
0.0f, Vector2.Zero, Escala, SpriteEffects.None, 0);

//Finaliza el "ciclo" de los sprite
spriteBatch.End();
```

**Paso 5:** Y la clase principal (Game1.cs), la que ejecuta el juego tiene:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
```

```

using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace Scroll
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        Texture2D Fondo; //Objeto de tipo textura que tendrá el fondo
        public Vector2 Posicion = new Vector2(0, 0); //Objeto de tipo Vector2 que controla la posición de la imagen.

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // TODO: use this.Content to load your game content here

            //Utiliza genéricos <> por eso el uso de <> En C++ se llaman plantillas. Content carga el contenido binario.
            Fondo = Content.Load<Texture2D>("Fondo");
        }

        /// <summary>
        /// UnloadContent will be called once per game and is the place to unload
        /// all content.
        /// </summary>
        protected override void UnloadContent()
        {
            // TODO: Unload any non ContentManager content here
        }

        /// <summary>
        /// Allows the game to run logic such as updating the world,
        /// checking for collisions, gathering input, and playing audio.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Update(GameTime gameTime)
        {
            // Allows the game to exit
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
                this.Exit();

            // TODO: Add your update logic here
            Vector2 Direccion = new Vector2(-1, 0); //Dirección a la que se desplaza el fondo
            Vector2 Velocidad = new Vector2(50, 0); //Velocidad con la que se desplaza el fondo

            //Cálculo de la nueva posición del fondo
            Posicion += Direccion * (float)gameTime.ElapsedGameTime.TotalSeconds;

            base.Update(gameTime);
        }

        /// <summary>
        /// This is called when the game should draw itself.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            // TODO: Add your drawing code here
            float Escala = 1.0f;
        }
    }
}

```

```
//Se inicia el spritebatch (Entre Begin y End se dibuja en cada unidad de tiempo los sprites)
spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

//Dibujar el fondo combinando el color blanco con el fondo
spriteBatch.Draw(Fondo, Posicion,
    new Rectangle(0, 0, Fondo.Width, Fondo.Height), Color.White,
    0.0f, Vector2.Zero, Escala, SpriteEffects.None, 0);

//Finaliza el "ciclo" de los sprite
spriteBatch.End();

base.Draw(gameTime);
}

}
```

Este es el resultado

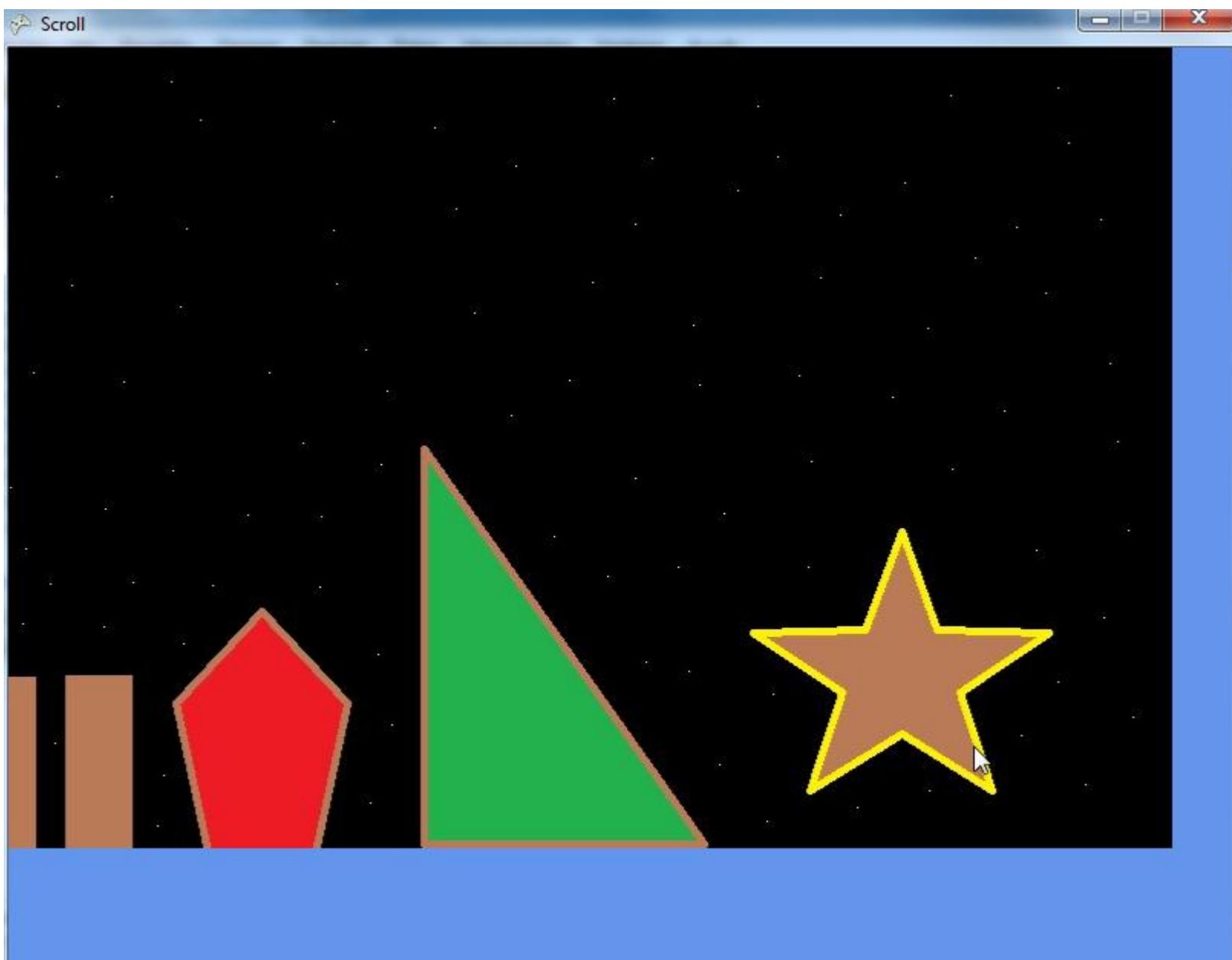


Figura 180: Ejemplo de ejecución del deslizamiento (scroll) de la imagen

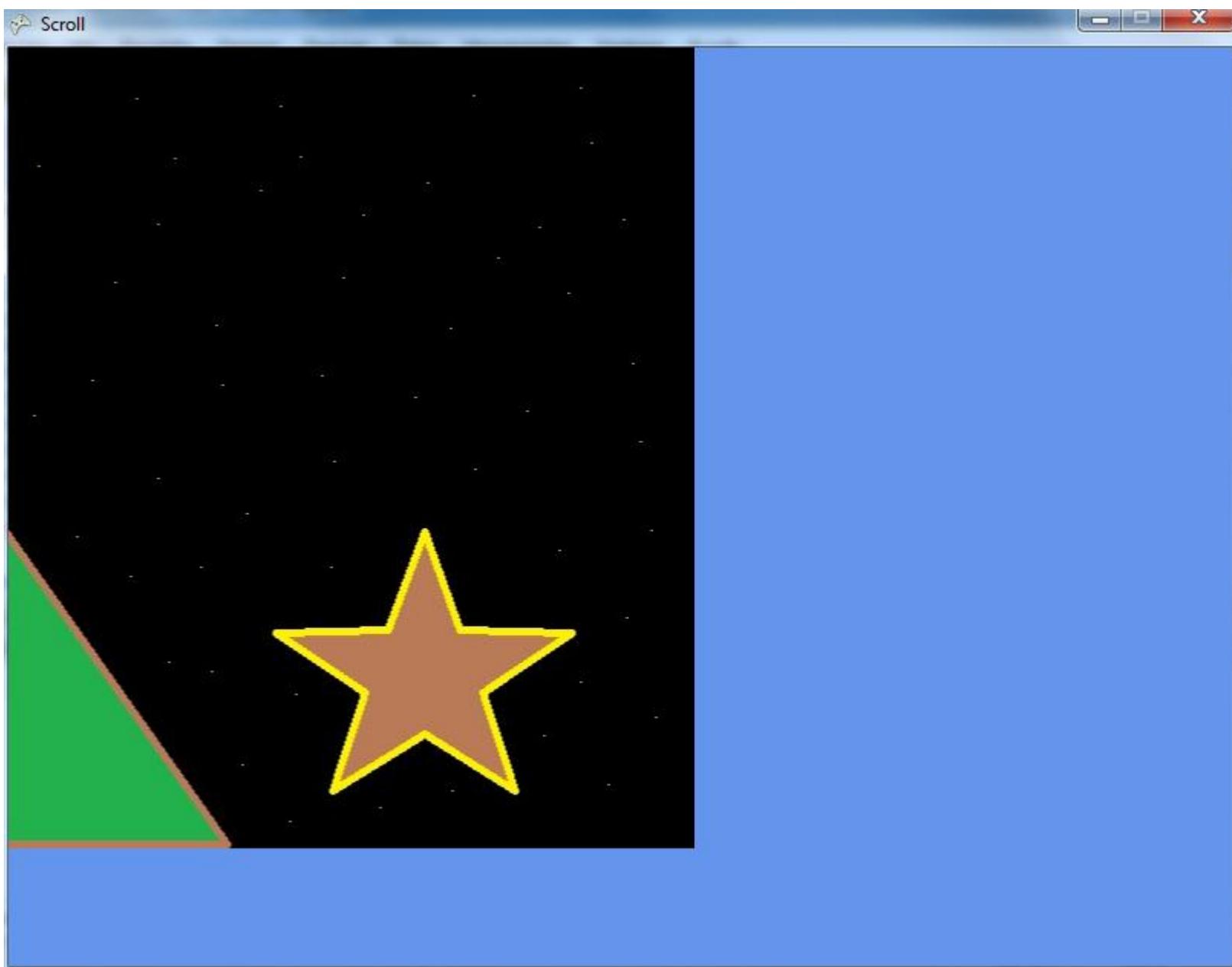


Figura 181: Ejemplo de ejecución del deslizamiento (scroll) de la imagen

La idea ahora es desplazar el fondo y mostrarlo como una cinta sin fin. Para eso necesitamos varias imágenes de fondo, a medida que se desplaza una imagen va apareciendo la otra. La imagen que desaparece entonces hace cola de nuevo para aparecer en una suerte de cinta sin fin.

**Paso 6:** Tenemos varios fondos

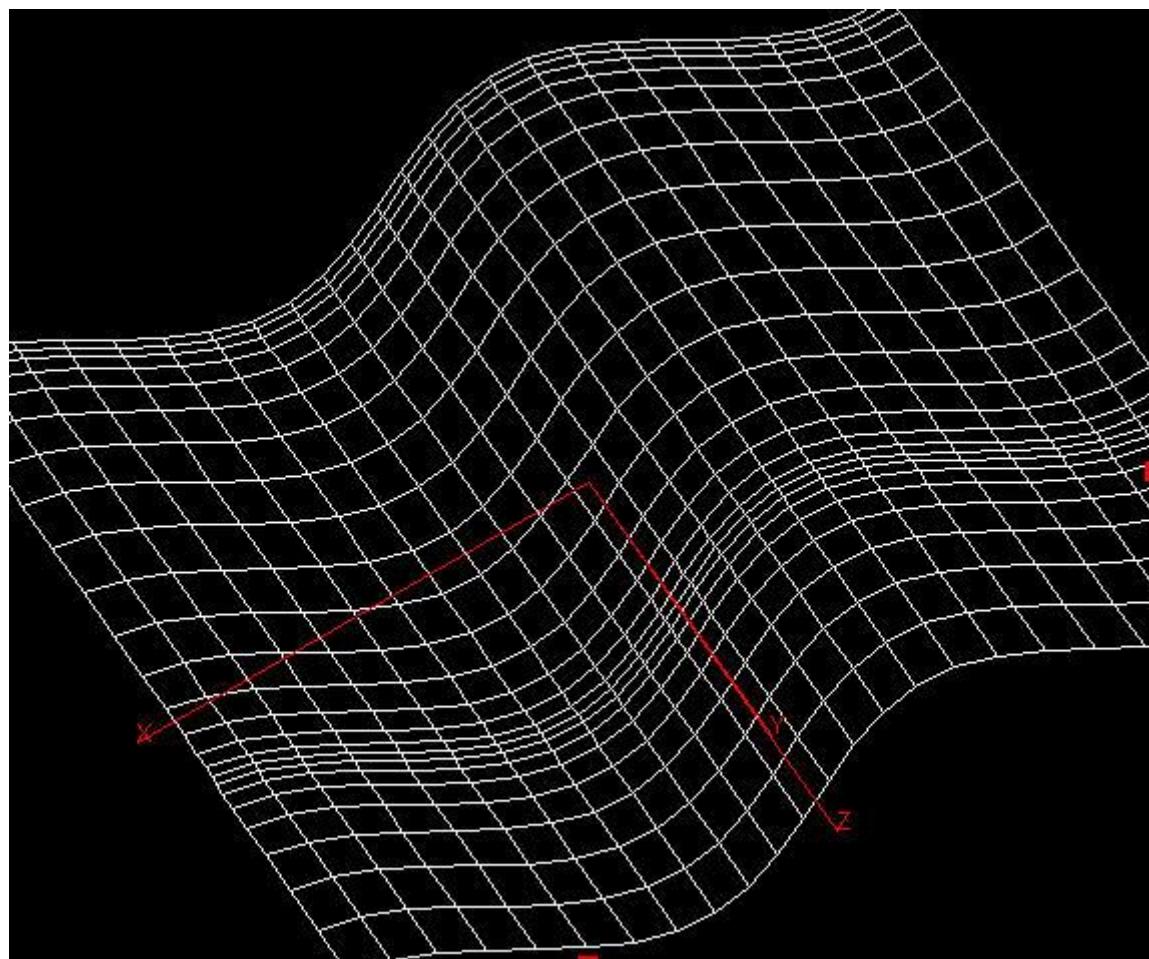
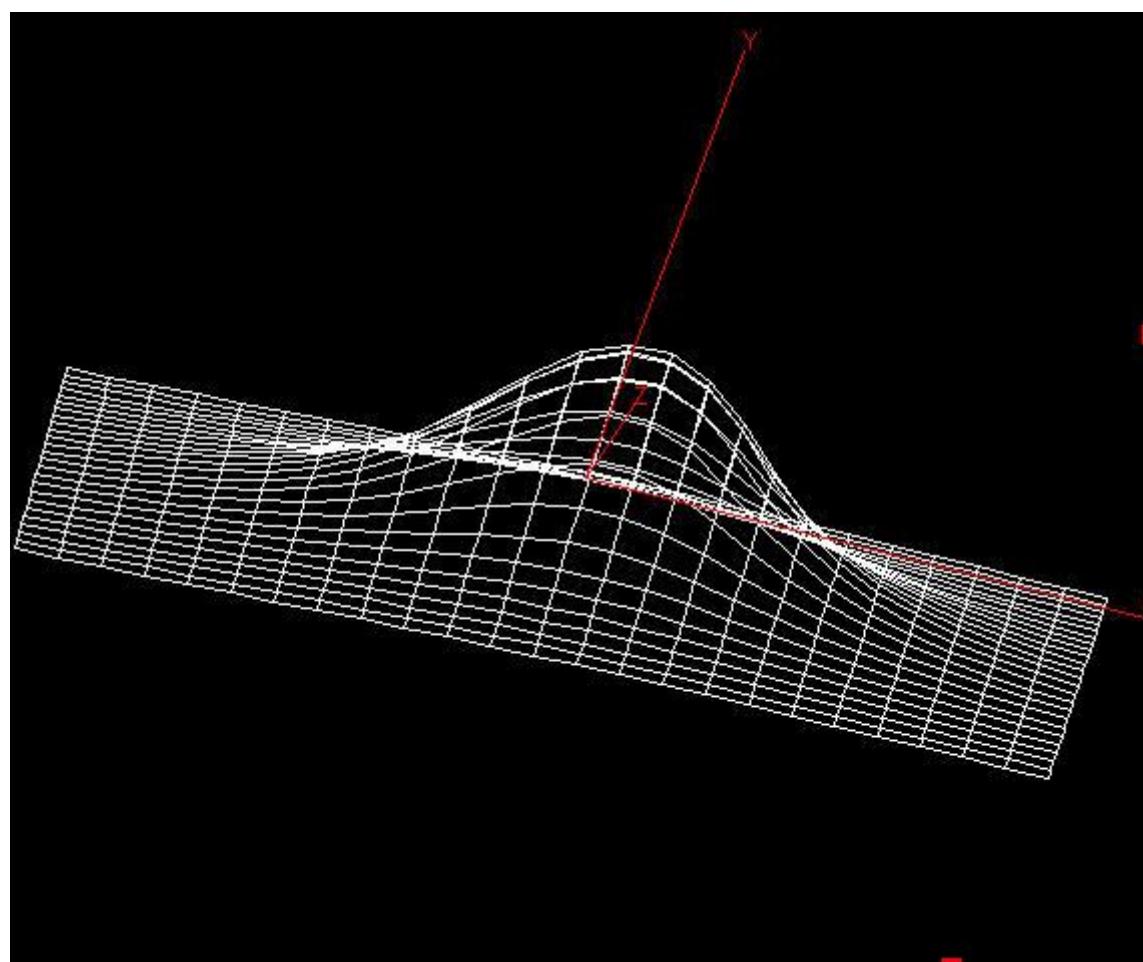
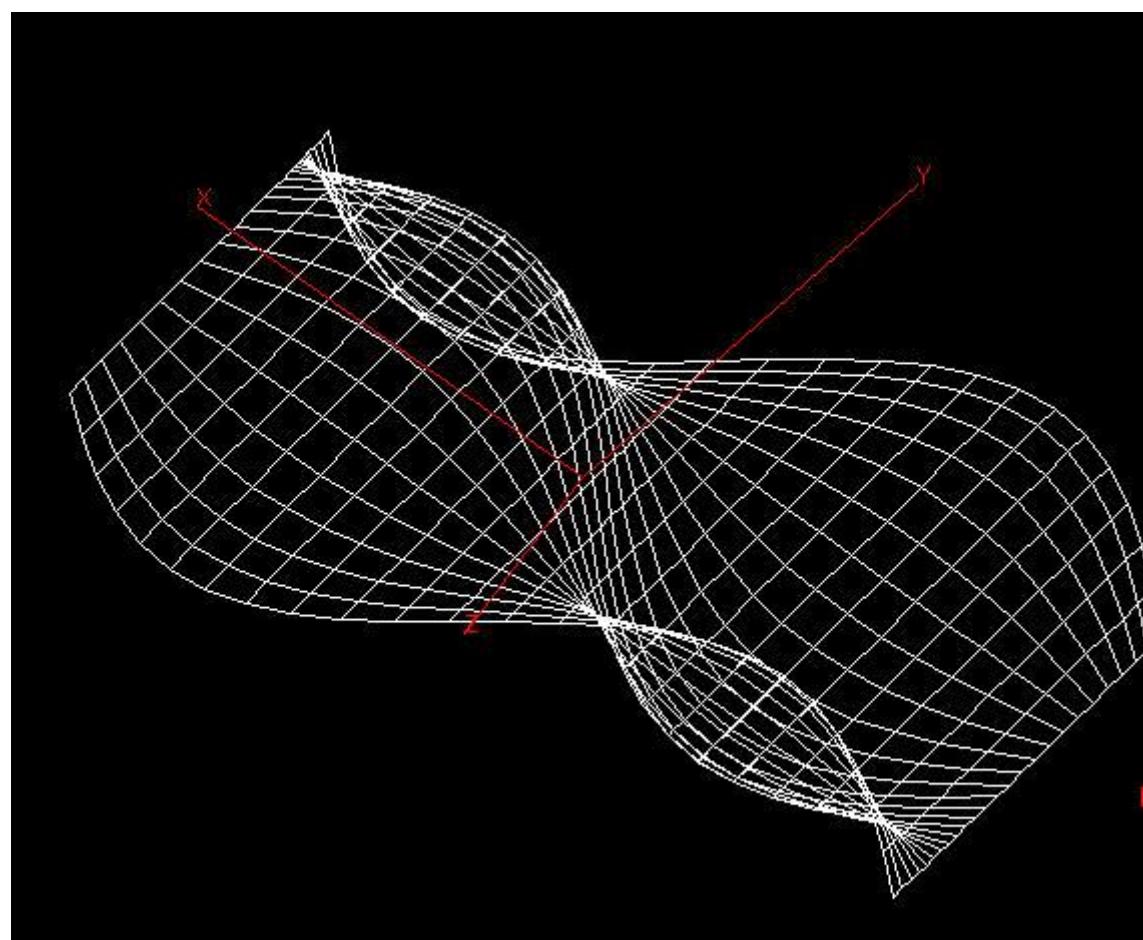


Figura 182: Una imagen de ejemplo para hacer deslizamiento (scroll) horizontal



**Figura 183:** Una imagen de ejemplo para hacer deslizamiento (scroll) horizontal



**Figura 184:** Una imagen de ejemplo para hacer deslizamiento (scroll) horizontal

**Paso 7:** Necesitamos por lo tanto tres variables de tipo textura (que almacenen los fondos) y tres variables que sirvan para llevar la posición de esos fondos.

```
Texture2D Fondo1; //Objeto de tipo textura que tendrá el fondo 1
Texture2D Fondo2; //Objeto de tipo textura que tendrá el fondo 2
Texture2D Fondo3; //Objeto de tipo textura que tendrá el fondo 3

public Vector2 Posicion1 = new Vector2(0, 0); //Posición Fondo 1
public Vector2 Posicion2 = new Vector2(0, 0); //Posición Fondo 2
public Vector2 Posicion3 = new Vector2(0, 0); //Posición Fondo 3
```

**Paso 8:** Por supuesto, se cargan los fondos en el proyecto

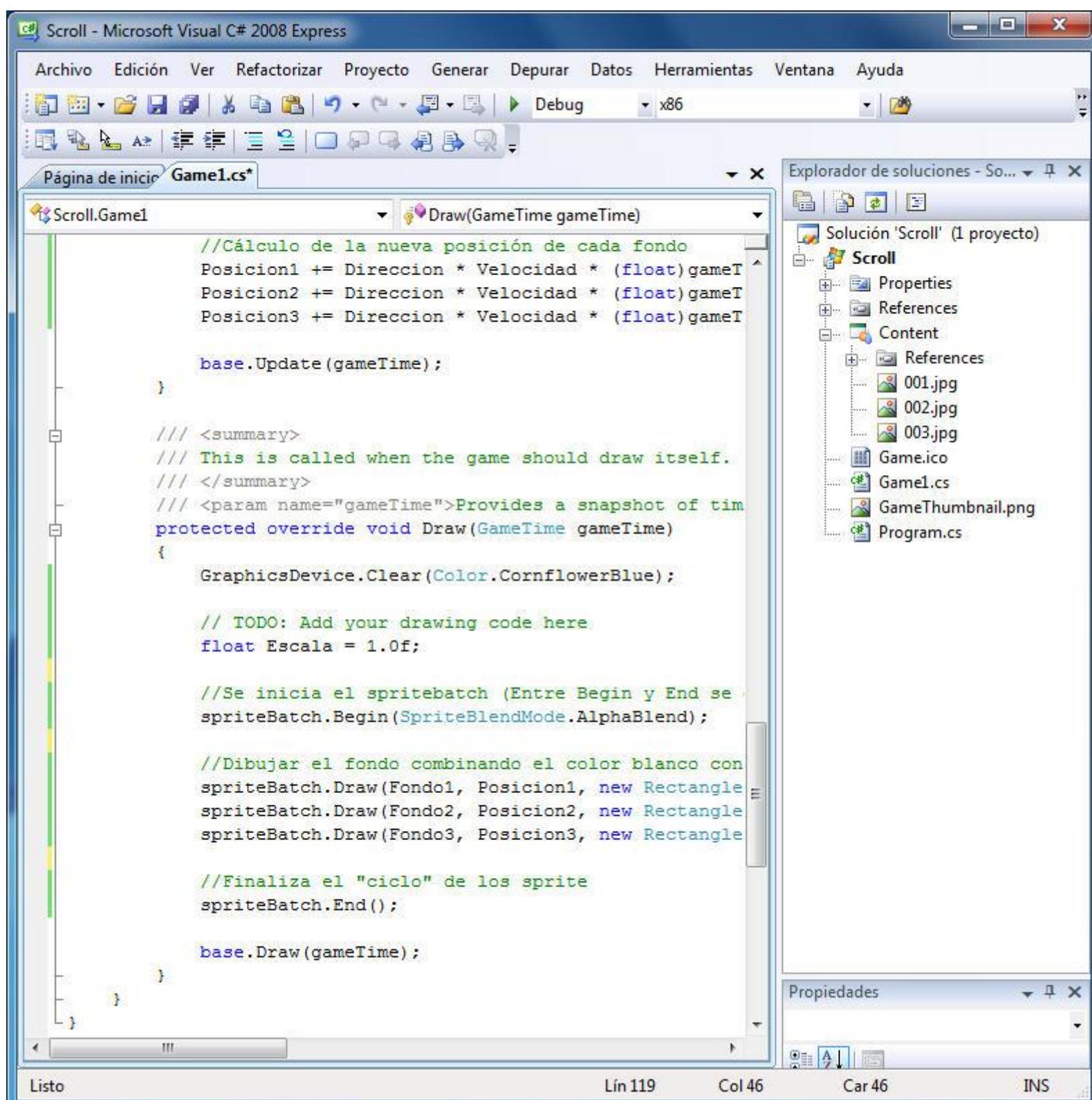


Figura 185: Las tres imágenes se ubican como recursos en el proyecto de videojuego

**Paso 9:** Las instrucciones de carga de los fondos en LoadContent y la inicialización de las posiciones (un fondo después de otro)

```
//Utiliza genéricos <> por eso el uso de <> En C++ se llaman plantillas. Content carga el contenido binario.
Fondo1 = Content.Load<Texture2D>("001");
Fondo2 = Content.Load<Texture2D>("002");
Fondo3 = Content.Load<Texture2D>("003");

//Inicializa las posiciones, un fondo después de otro en el eje X
Posicion1.X = 0;
Posicion2.X = Posicion1.X + Fondo1.Width;
Posicion3.X = Posicion2.X + Fondo2.Width;
```

**Paso 10:** La parte de actualización, se desarrolla así: los fondos se van desplazando, cuando el fondo1 desaparezca hace cola después del fondo 3 y así se da el efecto de cinta sin fin.

```
Vector2 Dirección = new Vector2(-1, 0); //Dirección a la que se desplaza el fondo
Vector2 Velocidad = new Vector2(200, 0); //Velocidad con la que se desplaza el fondo

//Para el efecto de cinta sin fin
if (Posicion1.X < -Fondo1.Width) Posicion1.X = Posicion3.X + Fondo3.Width;
if (Posicion2.X < -Fondo2.Width) Posicion2.X = Posicion1.X + Fondo1.Width;
if (Posicion3.X < -Fondo3.Width) Posicion3.X = Posicion2.X + Fondo2.Width;

//Cálculo de la nueva posición de cada fondo
Posicion1 += Dirección * (float)gameTime.ElapsedGameTime.TotalSeconds;
Posicion2 += Dirección * (float)gameTime.ElapsedGameTime.TotalSeconds; //va después del fondo 1
Posicion3 += Dirección * (float)gameTime.ElapsedGameTime.TotalSeconds; //va después del fondo 2
```

**Paso 11:** Y en Draw se dibujan los tres fondos en las posiciones

```
// TODO: Add your drawing code here
float Escala = 1.0f;
```

```
//Se inicia el spritebatch (Entre Begin y End se dibuja en cada unidad de tiempo los sprites)
spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

//Dibujar el fondo combinando el color blanco con el fondo
spriteBatch.Draw(Fondo1, Posicion1, new Rectangle(0, 0, Fondo1.Width, Fondo1.Height), Color.White, 0.0f, Vector2.Zero,
Escala, SpriteEffects.None, 0);
spriteBatch.Draw(Fondo2, Posicion2, new Rectangle(0, 0, Fondo2.Width, Fondo2.Height), Color.White, 0.0f, Vector2.Zero,
Escala, SpriteEffects.None, 0);
spriteBatch.Draw(Fondo3, Posicion3, new Rectangle(0, 0, Fondo3.Width, Fondo3.Height), Color.White, 0.0f, Vector2.Zero,
Escala, SpriteEffects.None, 0);

//Finaliza el "ciclo" de los sprite
spriteBatch.End();
```

Este es el código completo de la clase Game1.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace Scroll
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        Texture2D Fondo1; //Objeto de tipo textura que tendrá el fondo 1
        Texture2D Fondo2; //Objeto de tipo textura que tendrá el fondo 2
        Texture2D Fondo3; //Objeto de tipo textura que tendrá el fondo 3

        public Vector2 Posicion1 = new Vector2(0, 0); //Posición Fondo 1
        public Vector2 Posicion2 = new Vector2(0, 0); //Posición Fondo 2
        public Vector2 Posicion3 = new Vector2(0, 0); //Posición Fondo 3

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // TODO: use this.Content to load your game content here

            //Utiliza genéricos <> por eso el uso de <> En C++ se llaman plantillas. Content carga el contenido binario.
            Fondo1 = Content.Load<Texture2D>("001");
            Fondo2 = Content.Load<Texture2D>("002");
            Fondo3 = Content.Load<Texture2D>("003");

            //Inicializa las posiciones, un fondo después de otro en el eje X
            Posicion1.X = 0;
            Posicion2.X = Posicion1.X + Fondo1.Width;
            Posicion3.X = Posicion2.X + Fondo2.Width;
        }

        /// <summary>
        /// UnloadContent will be called once per game and is the place to unload
        /// all content.
        /// </summary>
```

```

protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // TODO: Add your update logic here

    Vector2 Direccion = new Vector2(-1, 0); //Dirección a la que se desplaza el fondo
    Vector2 Velocidad = new Vector2(200, 0); //Velocidad con la que se desplaza el fondo

    //Para el efecto de cinta sin fin
    if (Posicion1.X < -Fondo1.Width) Posicion1.X = Posicion3.X + Fondo3.Width;
    if (Posicion2.X < -Fondo2.Width) Posicion2.X = Posicion1.X + Fondo1.Width;
    if (Posicion3.X < -Fondo3.Width) Posicion3.X = Posicion2.X + Fondo2.Width;

    //Cálculo de la nueva posición de cada fondo
    Posicion1 += Direccion * (float)gameTime.ElapsedGameTime.TotalSeconds;
    Posicion2 += Direccion * (float)gameTime.ElapsedGameTime.TotalSeconds; //va después del fondo 1
    Posicion3 += Direccion * (float)gameTime.ElapsedGameTime.TotalSeconds; //va después del fondo 2

    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here
    float Escala = 1.0f;

    //Se inicia el spritebatch (Entre Begin y End se dibuja en cada unidad de tiempo los sprites)
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

    //Dibujar el fondo combinando el color blanco con el fondo
    spriteBatch.Draw(Fondo1, Posicion1, new Rectangle(0, 0, Fondo1.Width, Fondo1.Height), Color.White, 0.0f, Vector2.Zero,
        Escala, SpriteEffects.None, 0);
    spriteBatch.Draw(Fondo2, Posicion2, new Rectangle(0, 0, Fondo2.Width, Fondo2.Height), Color.White, 0.0f, Vector2.Zero,
        Escala, SpriteEffects.None, 0);
    spriteBatch.Draw(Fondo3, Posicion3, new Rectangle(0, 0, Fondo3.Width, Fondo3.Height), Color.White, 0.0f, Vector2.Zero,
        Escala, SpriteEffects.None, 0);

    //Finaliza el "ciclo" de los sprite
    spriteBatch.End();

    base.Draw(gameTime);
}
}
}

```

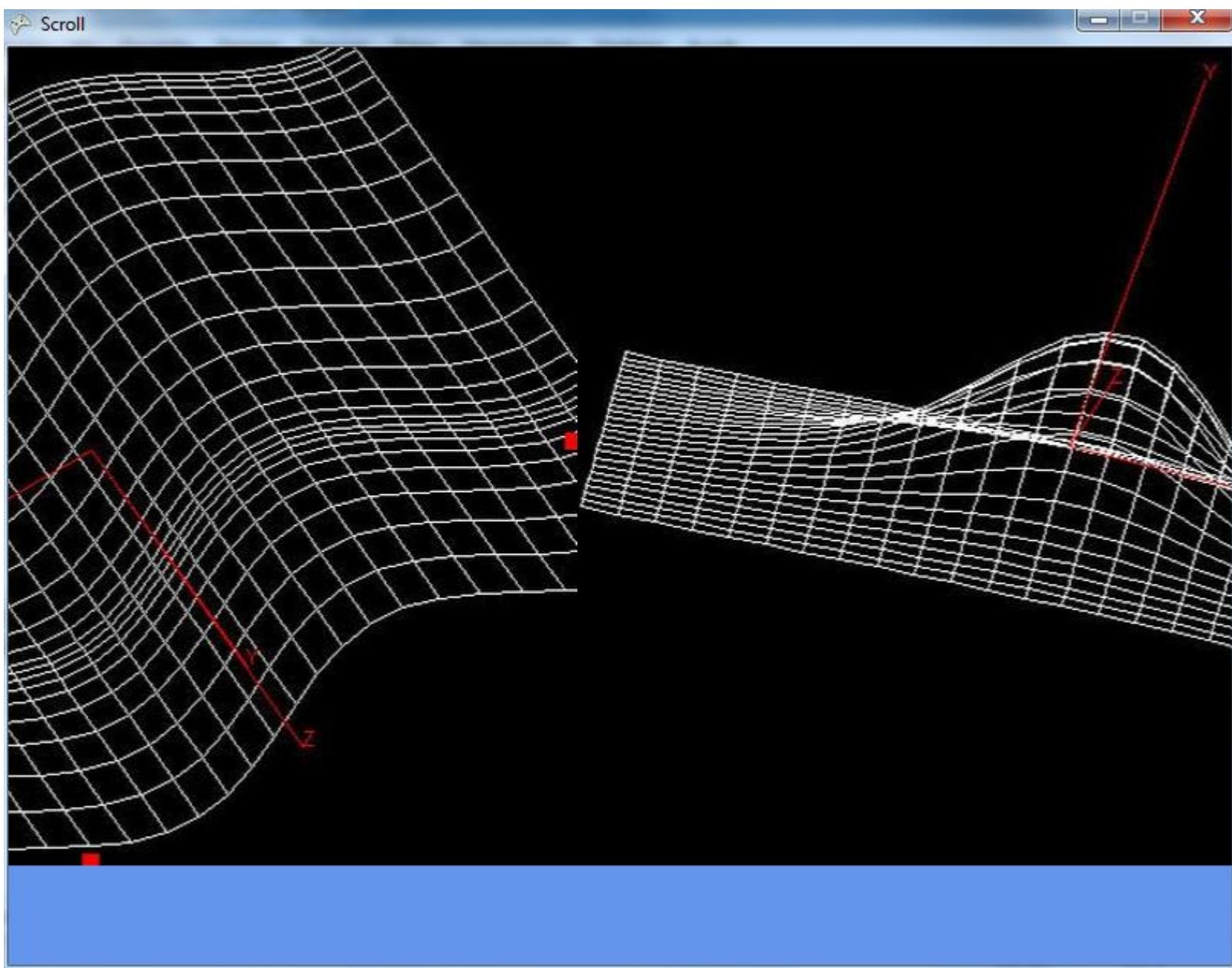


Figura 186: Ejemplo de ejecución del deslizamiento (scroll) horizontal sin fin

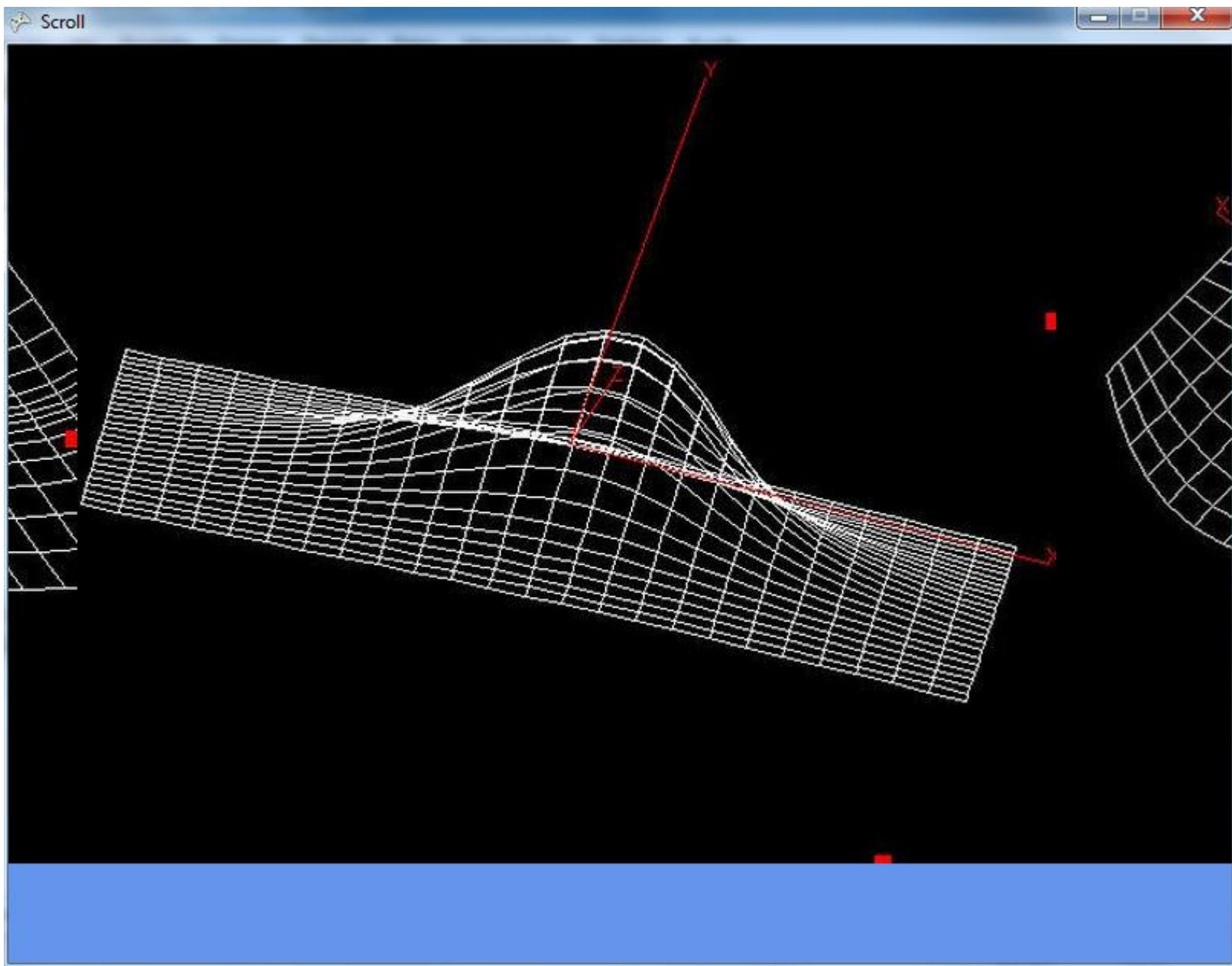


Figura 187: Ejemplo de ejecución del deslizamiento (scroll) horizontal sin fin

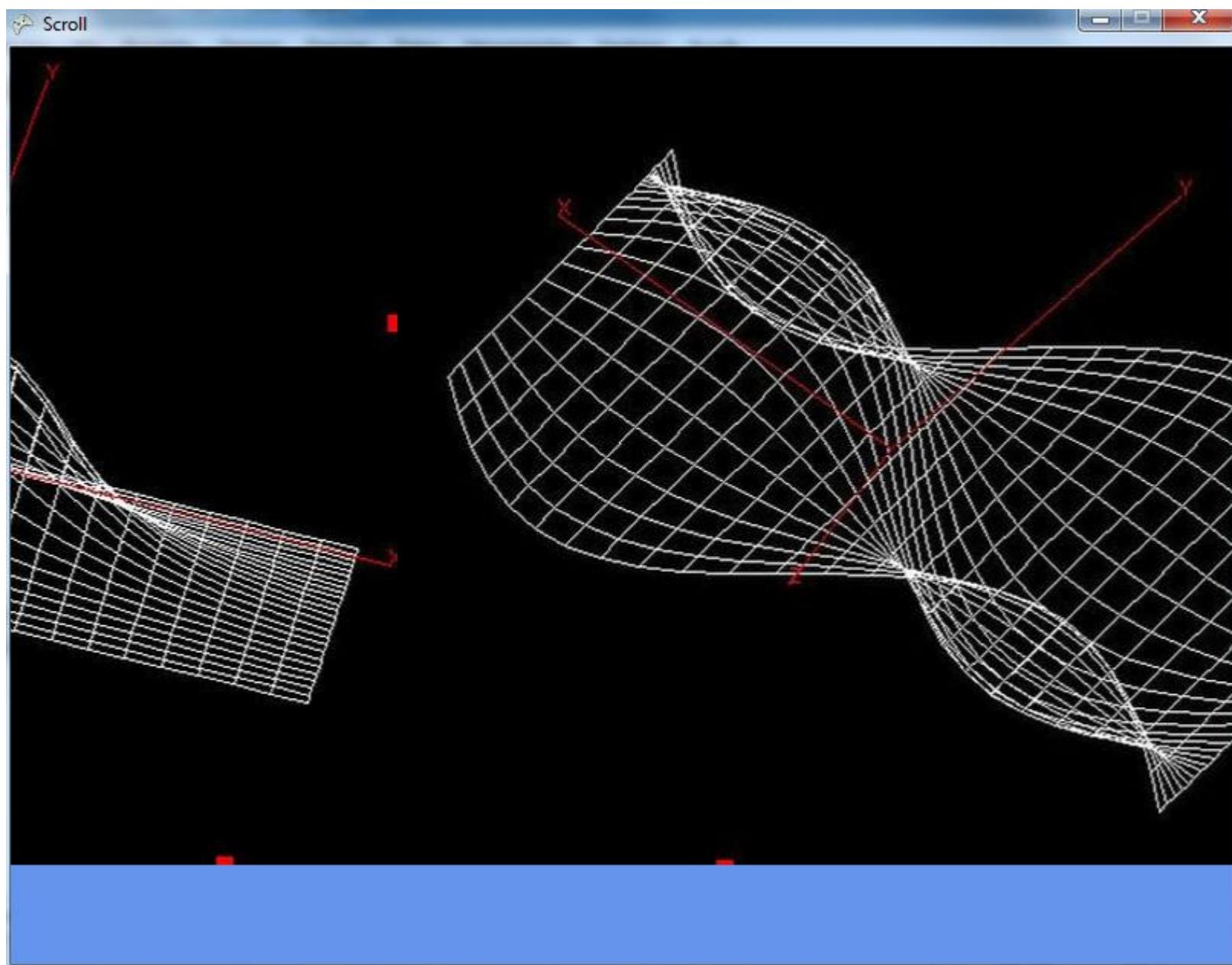


Figura 188: Ejemplo de ejecución del deslizamiento (scroll) horizontal sin fin

### 33. XNA: Mejorando el código para hacerlo más fácil de entender.

Es muy buena idea en el desarrollo de un videojuego separar en diferentes clases los diferentes items (fondos, actores). De esa forma es más sencillo hacer mantenimiento al software y también detectar errores o hacer depuración del código. En el siguiente ejemplo se muestra como el juego hecho al principio es dividido en varias clases, dejando a la clase principal con poco código y lo mejor más sencillo de entender.

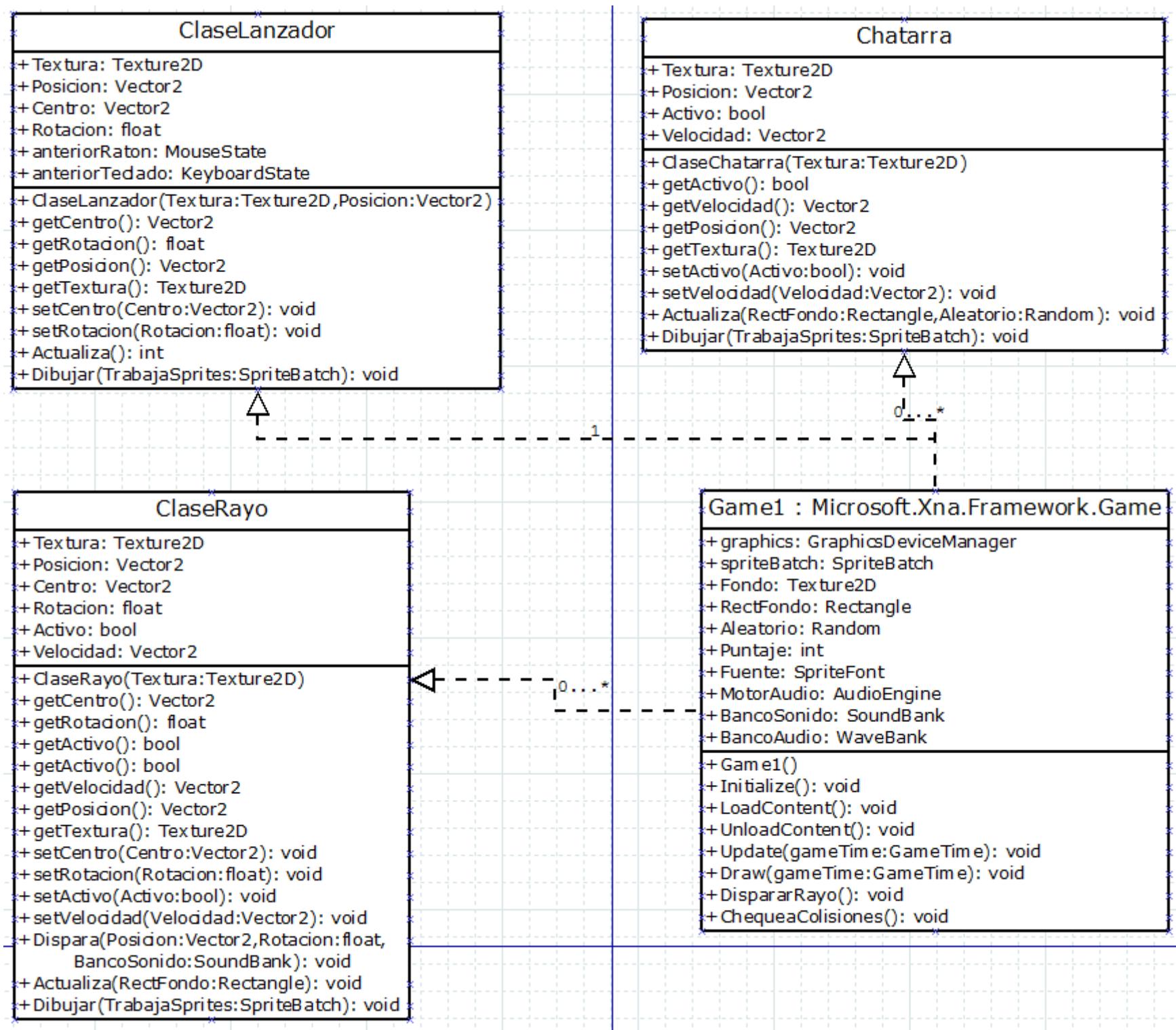


Figura 189: Diagrama de clases del juego

#### LANZADOR y BASE DEL LANZADOR

```

//Clase que administra el lanzador y la base. Es el protagonista del videojuego.
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    //Clase que administra el lanzador
    class ClaseLanzador
    {
        private Texture2D Textura; //La textura (o imagen) que tendrá el sprite
        private Vector2 Posicion; //En que punto se encuentra ubicado el sprite

        private Vector2 Centro; //El centro por donde gira el lanzador
        private float Rotacion; //Angulo de rotación del lanzador

        //Almacena el estado anterior del ratón
        private MouseState anteriorRaton;

        //Almacena el estado anterior del teclado
        private KeyboardState anteriorTeclado;

        //Constructor
    }
}

```

```

public ClaseLanzador(Texture2D Textura, Vector2 Posicion)
{
    this.Textura = Textura; //Recibe la textura por parámetro
    this.Posicion = Posicion; // Posición [0,0] por defecto
    Rotacion = (float)0.0; //Rotación 0.0 en radianes por defecto
    Centro = new Vector2(Textura.Width / 2, Textura.Height / 2); //El centro del sprite
}

//Leyendo los valores de los atributos
public Vector2 getCentro() { return Centro; }
public float getRotacion() { return Rotacion; }
public Vector2 getPosicion() { return Posicion; }
public Texture2D getTextura() { return Textura; }

//Dando valores a los atributos
public void setCentro(Vector2 Centro) { this.Centro = Centro; }
public void setRotacion(float Rotacion) { this.Rotacion = Rotacion; }

//Actualiza el estado del lanzador dependiendo de los comandos del jugador
public int Actualiza()
{
    //Retorna si oprime algún comando
    int Comando = 0;

    // Código para leer el estado del Gamepad de una XBOX. Se considera que hay un jugador al menos.
    GamePadState EstadoControl = GamePad.GetState(PlayerIndex.One);

    //Como hay que tener en cuenta los controles del XBOX llamados thumbstick, se
    //observa que botón del XBOX oprimió, eso genera una constante algo grande por lo que se multiplica por 0.1
    Rotacion += EstadoControl.ThumbSticks.Left.X * 0.1f;

    //Esta es una compilación condicional. El teclado y el mouse solo son leídos si la aplicación ejecuta en un PC
#if !XBOX
    //Código para leer el teclado
    KeyboardState estadoTeclado = Keyboard.GetState();
    if (estadoTeclado.IsKeyDown(Keys.Left)) Rotacion -= 0.1f; //Gira el lanzador hacia la izquierda
    if (estadoTeclado.IsKeyDown(Keys.Right)) Rotacion += 0.1f; //Gira el lanzador hacia la derecha

    //Código para leer el ratón
    MouseState Raton = Mouse.GetState();

    //Si movió el ratón entonces el lanzador cambia de ángulo
    if (Raton != anteriorRaton)
    {
        int DiferX = Raton.X - anteriorRaton.X; //La diferencia entre la anterior posición del ratón y la nueva.
        Rotacion += (float) DiferX / 100; //El movimiento en X del ratón cambia el ángulo del lanzador
    }

    //Se actualiza el estado del ratón
    anteriorRaton = Raton;

    //Dispara un solo rayo.
    //Chequea si el usuario mantiene presionada la tecla de disparo por lo que hace caso omiso a eso,
    //el usuario debe presionar y soltar la tecla de disparo para lanzar mas rayos.
    if (estadoTeclado.IsKeyDown(Keys.Space) && anteriorTeclado.IsKeyUp(Keys.Space))
        Comando = 1;

    anteriorTeclado = estadoTeclado;
#endif

    //Esta instrucción limita entre 0 y 90 grados el giro del lanzador. Recordar que se hace uso de radianes.
    Rotacion = MathHelper.Clamp(this.getRotacion(), -MathHelper.PiOver2, 0);

    return Comando;
}

//Dibuja el sprite
public void Dibujar(SpriteBatch TrabajaSprites)
{
    /* Dibuja el lanzador como tal, estos son los parámetros
     * Parámetro 1: El sprite del lanzador
     * Parámetro 2: La posición del lanzador
     * Parámetro 3: null porque es solo una imagen (no una sucesión de estas)
     * Parámetro 4: Color.White, combinación de color
     * Parámetro 5: Angulo de rotación por defecto
     * Parámetro 6: Centro, el centro del sprite para girarlo después
     * Parámetro 7: Escala a dibujar
     * Parametro 8: Dibujar tal como es el sprite (no reflejo vertical ni horizontal)
     * Parámetro 9: Ponerlo en la capa superior */
    TrabajaSprites.Draw(Textura, Posicion, null, Color.White, Rotacion, Centro, 1f, SpriteEffects.None, 0);
}
}
}

```

## RAYO

```

//Clase que administra el rayo disparado
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;

```

```

using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    class ClaseRayo
    {
        private Texture2D Textura; //La textura (o imagen) que tendrá el sprite
        private Vector2 Posicion; //En que punto se encuentra ubicado el sprite

        private Vector2 Centro; //El centro por donde gira el lanzador
        private float Rotacion; //Angulo de rotación del lanzador
        private bool Activo; //Sabe si esta activo o no el rayo
        private Vector2 Velocidad; //Dirección y velocidad del rayo

        //Constructor
        public ClaseRayo(Texture2D Textura)
        {
            this.Textura = Textura; //Recibe la textura por parámetro
            Posicion = Vector2.Zero; // Posición [0,0] por defecto

            Rotacion = (float)0.0; //Rotación 0.0 en radianes por defecto
            Centro = new Vector2(Textura.Width / 2, Textura.Height / 2); //El centro del sprite

            Activo = false; //Por defecto no se muestra el rayo
            Velocidad = Vector2.Zero; //Por defecto el rayo no se mueve
        }

        //Leyendo los valores de los atributos
        public Vector2 getCentro() { return Centro; }
        public float getRotacion() { return Rotacion; }
        public bool getActivo() { return Activo; }
        public Vector2 getVelocidad() { return Velocidad; }
        public Vector2 getPosicion() { return Posicion; }
        public Texture2D getTextura() { return Textura; }

        //Dando valores a los atributos
        public void setCentro(Vector2 Centro) { this.Centro = Centro; }
        public void setRotacion(float Rotacion) { this.Rotacion = Rotacion; }
        public void setActivo(bool Activo) { this.Activo = Activo; }
        public void setVelocidad(Vector2 Velocidad) { this.Velocidad = Velocidad; }

        //Se dispara el rayo
        public void Dispara(Vector2 Posicion, float Rotacion, SoundBank BancoSonido)
        {
            //Activa el rayo (para que sea dibujado)
            Activo = true;

            //El rayo inicia en la posición del lanzador
            this.Posicion = Posicion;

            //En que dirección y a que velocidad sale el rayo. Hay que tener en cuenta la rotación del lanzador.
            Velocidad = new Vector2((float)Math.Cos(Rotacion), (float)Math.Sin(Rotacion)) * 4.0f;

            //Rota el sprite del rayo para que coincida con el del lanzador
            this.Rotacion = Rotacion;

            //Sonido del disparo
            BancoSonido.PlayCue("disparo");
        }

        //Se actualiza estado del rayo
        public void Actualiza(Rectangle RectFondo)
        {
            if (Activo)
            {
                //Solo es actualizar la velocidad porque el vector se actualiza gracias a esa magnitud
                Posicion += Velocidad;

                //Si el rayo se sale de la pantalla entonces se desactiva
                if (RectFondo.Contains(new Point((int)Posicion.X, (int)Posicion.Y)) == false)
                    Activo = false;
            }
        }

        //Dibuja el sprite
        public void Dibujar(SpriteBatch TrabajaSprites)
        {
            if (Activo)
                TrabajaSprites.Draw(Textura, Posicion, null, Color.White, Rotacion, Centro, 1f, SpriteEffects.None, 0);
        }
    }
}

```

#### CHATARRA

//Clase que administra el enemigo: chatarra espacial

```

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    class ClaseChatarra
    {
        private Texture2D Textura; //La textura (o imagen) que tendrá el sprite
        private Vector2 Posicion; //En que punto se encuentra ubicado el sprite

        private bool Activo; //Sabe si esta activo o no la chatarra
        private Vector2 Velocidad; //Dirección y velocidad de la chatarra

        //Constructor
        public ClaseChatarra(Texture2D Textura)
        {
            this.Textura = Textura; //Recibe la textura por parámetro
            Posicion = Vector2.Zero; // Posición [0,0] por defecto

            Activo = false; //Por defecto no se muestra la chatarra
            Velocidad = Vector2.Zero; //Por defecto la chatarra no se mueve
        }

        //Leyendo los valores de los atributos
        public bool getActivo() { return Activo; }
        public Vector2 getVelocidad() { return Velocidad; }
        public Vector2 getPosicion() { return Posicion; }
        public Texture2D getTextura() { return Textura; }

        //Dando valores a los atributos
        public void setActivo(bool Activo) { this.Activo = Activo; }
        public void setVelocidad(Vector2 Velocidad) { this.Velocidad = Velocidad; }

        //Actualiza el estado de la chatarra en el juego
        public void Actualiza(Rectangle RectFondo, Random Aleatorio)
        {
            float AlturaMax = 0.1f; //No pegado del techo
            float AlturaMin = 0.5f; //La mitad de la pantalla
            float VelocidadMax = 5.0f; //Máxima velocidad
            float VelocidadMin = 1.0f; //Mínima velocidad

            if (Activo)
            {
                //El enemigo avanza a determinada velocidad
                Posicion += Velocidad;

                //Si el enemigo se sale de la pantalla
                if (RectFondo.Contains(new Point((int)Posicion.X, (int)Posicion.Y)) == false)
                    Activo = false;
            }
            else
            {
                //Activa el enemigo
                Activo = true;

                //Interpolación. Posición eje Y aleatoria.
                Posicion = new Vector2(RectFondo.Right, MathHelper.Lerp((float)RectFondo.Height * AlturaMin, (float)RectFondo.Height * AlturaMax, (float)Aleatorio.NextDouble()));

                //Velocidad del enemigo. Aleatoria.
                Velocidad = new Vector2(MathHelper.Lerp(-VelocidadMin, -VelocidadMax, (float)Aleatorio.NextDouble()), 0);
            }
        }

        //Dibuja el sprite
        public void Dibujar(SpriteBatch TrabajaSprites)
        {
            if (Activo)
                TrabajaSprites.Draw(Textura, Posicion, Color.White);
        }
    }
}

```

### CLASE PRINCIPAL

```

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;

```

```

using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace PrimerJuego
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch; //Contenedor de los sprite

        Texture2D Fondo; //Variable de tipo textura que tendrá el fondo
        Rectangle RectFondo; //Un rectángulo donde estará la textura del fondo

        ClaseLanzador Lanzador; //El lanzador
        ClaseLanzador BaseLanzador; //Base del lanzador

        //Rays
        ClaseRayo[] Rayos;
        const int MAXRAYOS = 7;

        //Chatars
        ClaseChatarra[] Chatarras;
        const int MAXCHATARRAS = 9;

        //Generador de números aleatorios usado para ubicar a una altura al azar la chatarra
        Random Aleatorio = new Random();

        //Puntaje
        int Puntaje = 0; //Lleva cuantos enemigos han sido destruidos
        SpriteFont Fuente; //Variable tipo de letra para mostrar el puntaje

        //Audio
        AudioEngine MotorAudio;
        SoundBank BancoSonido;
        WaveBank BancoAudio;

        //Constructor
        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here
            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // TODO: use this.Content to load your game content here

            //Utiliza genéricos <> por eso el uso de <> En C++ se llaman plantillas. Content carga el contenido binario.
            Fondo = Content.Load<Texture2D>("Imagenes\\FondoJuego");

            //El rectángulo en el que estará contenido el fondo
            RectFondo = new Rectangle(0, 0, graphics.GraphicsDevice.Viewport.Width, graphics.GraphicsDevice.Viewport.Height);

            //Cargar e inicializar el Lanzador
            Lanzador = new ClaseLanzador(Content.Load<Texture2D>("Imagenes\\Lanzador"), new Vector2(130, 510)); //No es necesaria la extensión .tga

            //Cargar e inicializar la base
            BaseLanzador = new ClaseLanzador(Content.Load<Texture2D>("Imagenes\\Base"), new Vector2(90, 560)); //No es necesaria la extensión .tga

            //Cargar e inicializar los rayos
            Rayos = new ClaseRayo[MAXRAYOS];
            for (int Cont = 0; Cont < MAXRAYOS; Cont++)
                Rayos[Cont] = new ClaseRayo(Content.Load<Texture2D>("Imagenes\\Rayo"));

            //Cargar e inicializar los rayos
            Chatarras = new ClaseChatarra[MAXCHATARRAS];
        }
    }
}

```

```

for (int Cont = 0; Cont < MAXCHATARRAS; Cont++)
    Chatarras[Cont] = new ClaseChatarra(Content.Load<Texture2D>("Imagenes\\Chatarra"));

//Cargar la fuente
Fuente = Content.Load<SpriteFont>("Fuentes\\NuevaFuente");

//Cargar el sonido (ojo con la extension .xgs)
MotorAudio = new AudioEngine("Content\\Audio\\SonidosJuego.xgs"); //xna game studio
BancoSonido = new SoundBank(MotorAudio, "Content\\Audio\\Sound Bank.xsb"); //xna sound bank
BancoAudio = new WaveBank(MotorAudio, "Content\\Audio\\Wave Bank.xwb"); //xna wave bank
}

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    //Actualiza lanzador
    if (Lanzador.Actualiza() == 1) DispararRayo();

    //Actualiza los rayos
    foreach (ClaseRayo Rayo in Rayos)
        Rayo.Actualiza(RectFondo);

    //Actualiza los enemigos
    foreach (ClaseChatarra Chatarra in Chatarras)
        Chatarra.Actualiza(RectFondo, Aleatorio);

    //Chequea las colisiones
    ChequeaColisiones();

    // TODO: Add your update logic here
    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here

    //Se inicia el spritebatch (Entre Begin y End se dibuja en cada unidad de tiempo los sprites)
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

    //Dibujar el fondo combinando el color blanco con el fondo
    spriteBatch.Draw(Fondo, RectFondo, Color.White);

    //Dibuja la base del lanzador combinando con blanco
    BaseLanzador.Dibujar(spriteBatch);

    // Dibuja el lanzador como tal, estos son los parámetros
    Lanzador.Dibujar(spriteBatch);

    //Dibuja cada rayo
    foreach (ClaseRayo Rayo in Rayos)
        Rayo.Dibujar(spriteBatch);

    //Dibuja cada enemigo
    foreach (ClaseChatarra Chatarra in Chatarras)
        Chatarra.Dibujar(spriteBatch);

    //Dibuja el puntaje
    spriteBatch.DrawString(Fuente, "Destruido: " + Puntaje.ToString(), new Vector2(80, 60), Color.White);

    //Finaliza el "ciclo" de los sprite
    spriteBatch.End();

    base.Draw(gameTime);
}

//Metodo llamado desde Update
public void DispararRayo()
{
    //Busca un rayo inactivo
}

```

```

foreach (ClaseRayo Rayo in Rayos)
{
    //Si encuentra un rayo inactivo
    if (Rayo.getActivo() == false)
    {
        Rayo.Dispara(Lanzador.getPosicion(), Lanzador.getRotacion(), BancoSonido);
        return;
    }
}

//Método llamado desde Update
public void ChequeaColisiones()
{
    //Va de rayo en rayo
    foreach (ClaseRayo Rayo in Rayos)
        if (Rayo.getActivo() == true)
        {
            //Deduce el rectángulo del cohete y debe tener en cuenta el tamaño con que se esté dibujando en pantalla en Draw()
            Rectangle RectRayo = new Rectangle((int)Rayo.getPosicion().X, (int)Rayo.getPosicion().Y, (int)(Rayo.getTextura().Width * 0.2), (int)(Rayo.getTextura().Height * 0.2));

            //Chequea de enemigo en enemigo si el cohete lo colisiona
            foreach (ClaseChatarra Chatarra in Chatarras)
            {
                //Deduce el rectángulo del enemigo y debe tener en cuenta el tamaño con que se esté dibujando en pantalla en Draw()
                Rectangle RectEnemigo = new Rectangle((int)Chatarra.getPosicion().X, (int)Chatarra.getPosicion().Y, (int)(Chatarra.getTextura().Width), (int)(Chatarra.getTextura().Height));

                //Si ambos rectángulos se intersectan
                if (RectRayo.Intersects(RectEnemigo) == true)
                {
                    Chatarra.setActivo(false); //inactiva el enemigo
                    Rayo.setActivo(false); //inactiva el rayo
                    Puntaje++;

                    //Sonido de la explosión
                    BancoSonido.PlayCue("explosion");
                }
            }
        }
}
}

```

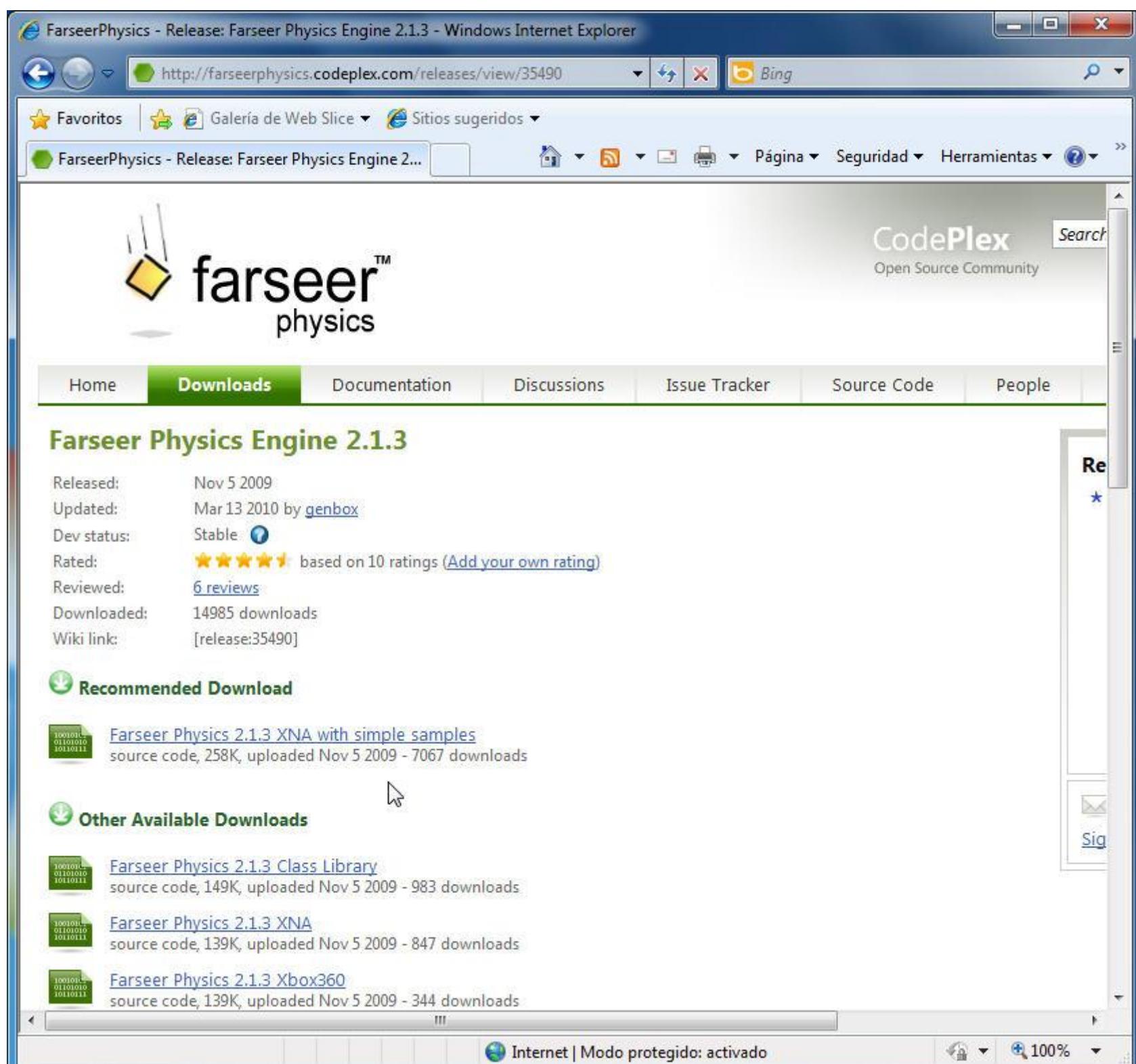
## 34. XNA: Uso de un motor de Física

En capítulos anteriores, hicimos que el protagonista corriera y saltara, pero el salto como tal no tenía realismo. La razón es que es necesario aplicar física a diversas acciones del protagonista para que el jugador capte el realismo. Para lograr esto, existen los denominados motores de física. El que se muestra a continuación es libre y dirigido para XNA.

Está en la dirección: <http://www.codeplex.com/FarseerPhysics>

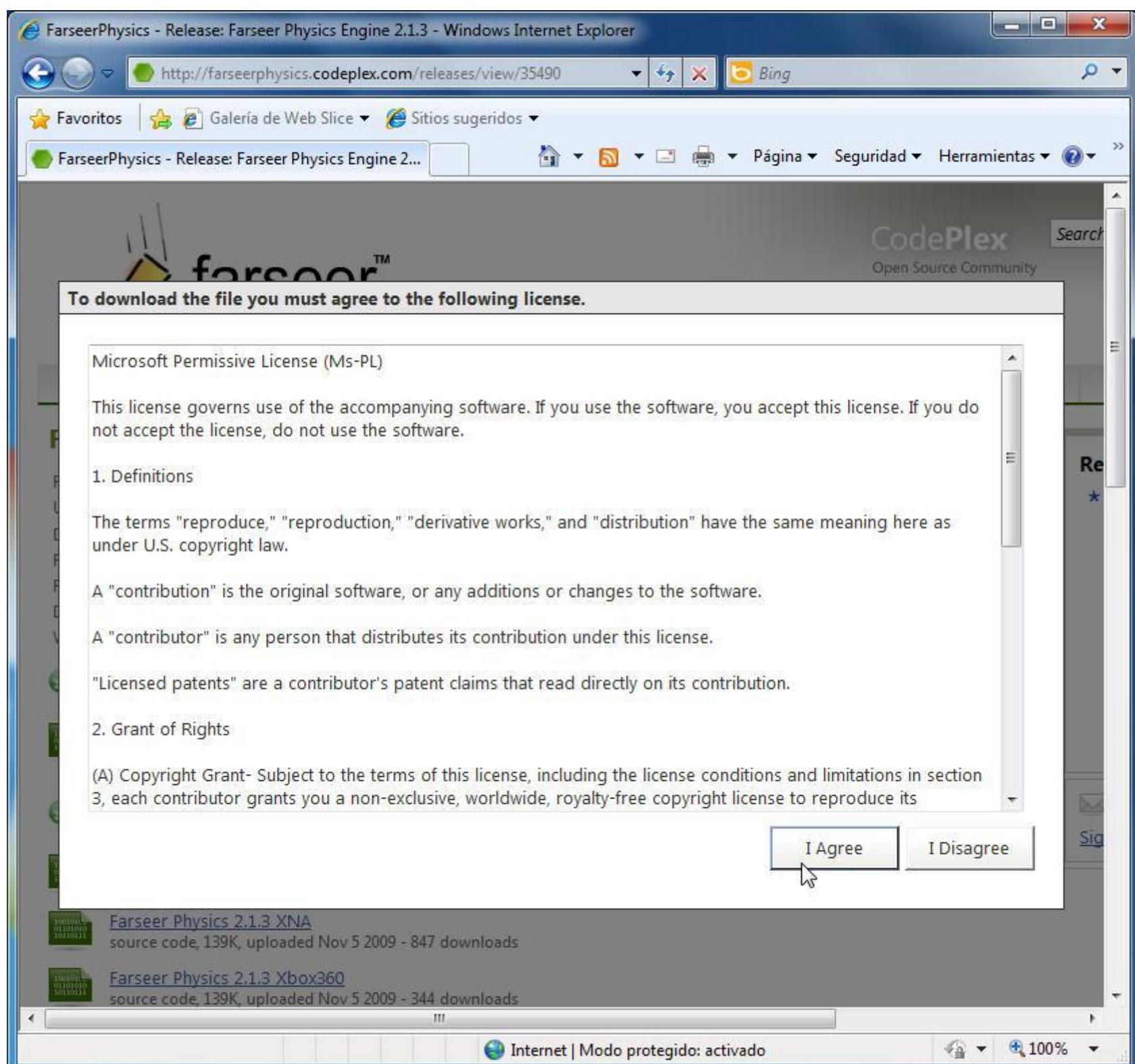


**Figura 190: Sitio oficial del motor de física**

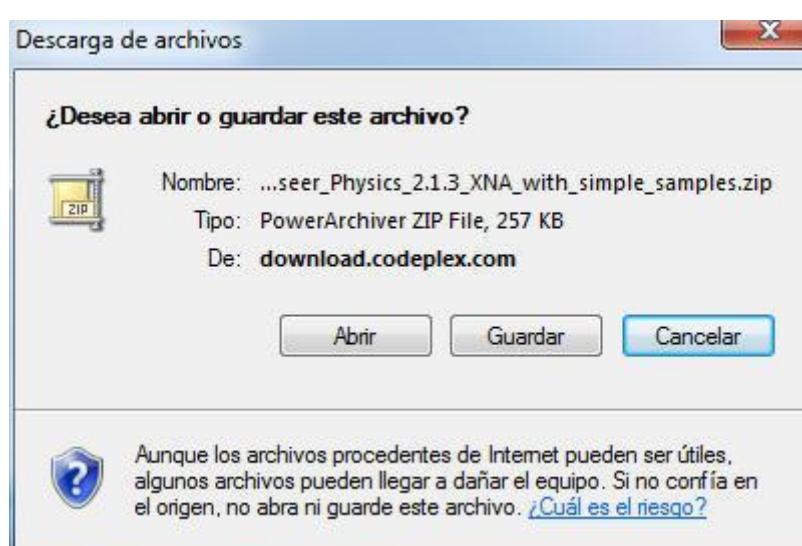


**Figura 191: Se descarga la versión 2.1.3 del motor de física**

Para empezar descargamos el Farseer Physics 2.1.3 XNA with simple samples



**Figura 192: Licencia del motor de física**



**Figura 193: Características del archivo de motor de física**

Una vez descomprimida la carpeta, se procede a abrir con Visual C#.

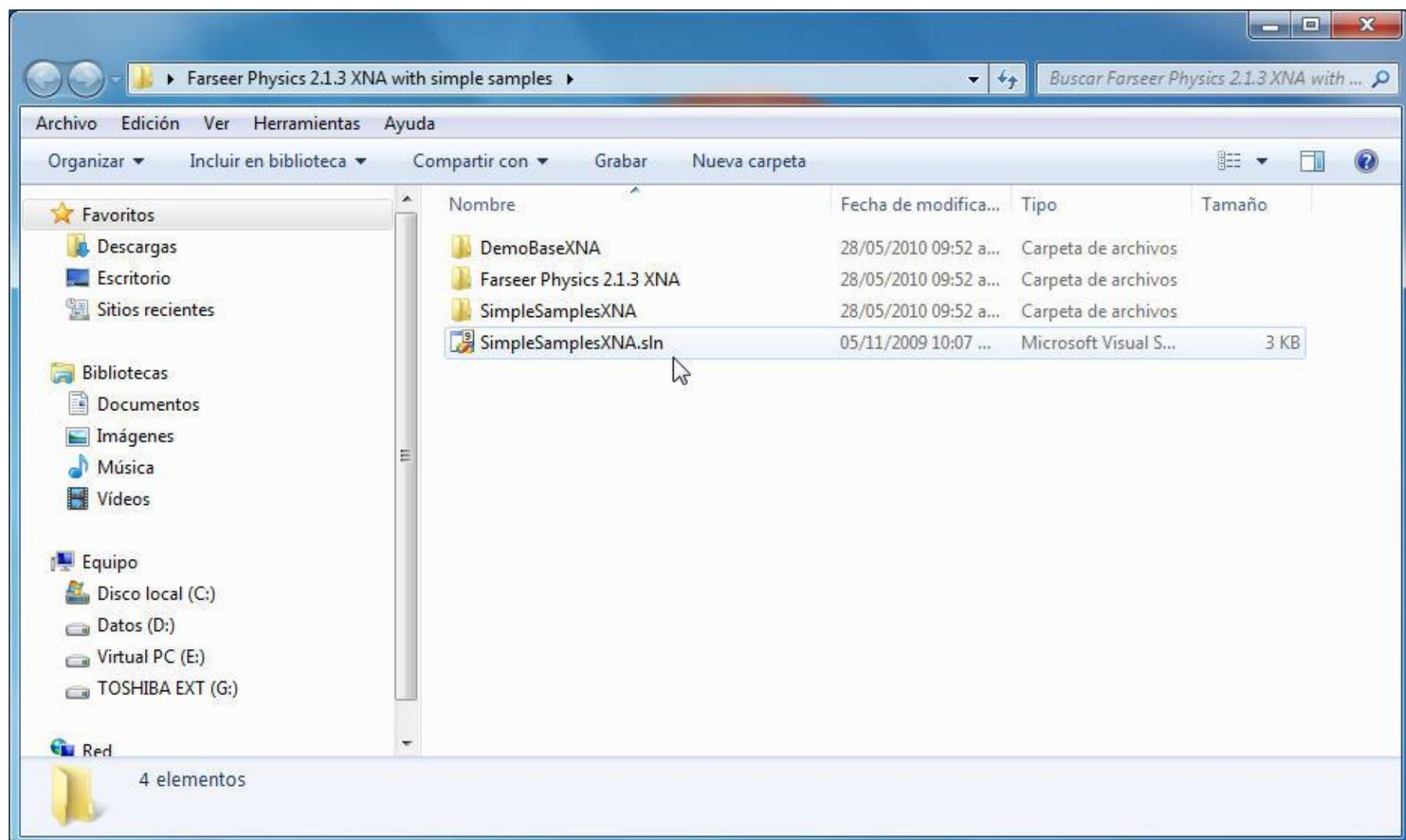


Figura 194: Descomprimiendo el archivo, tenemos un ejemplo listo para ser ejecutado

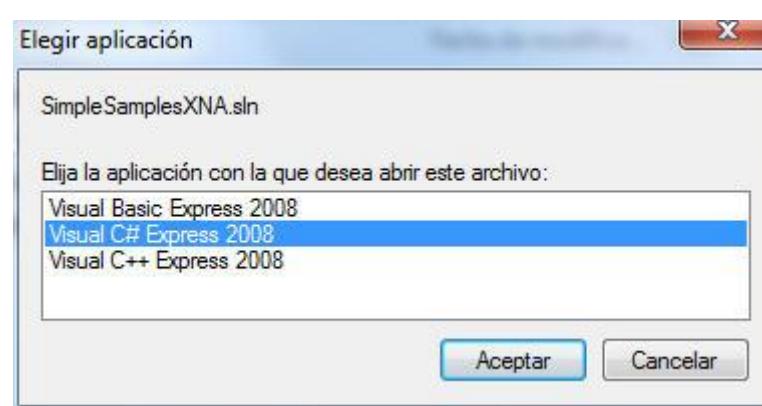


Figura 195: Ejecutamos Visual C#

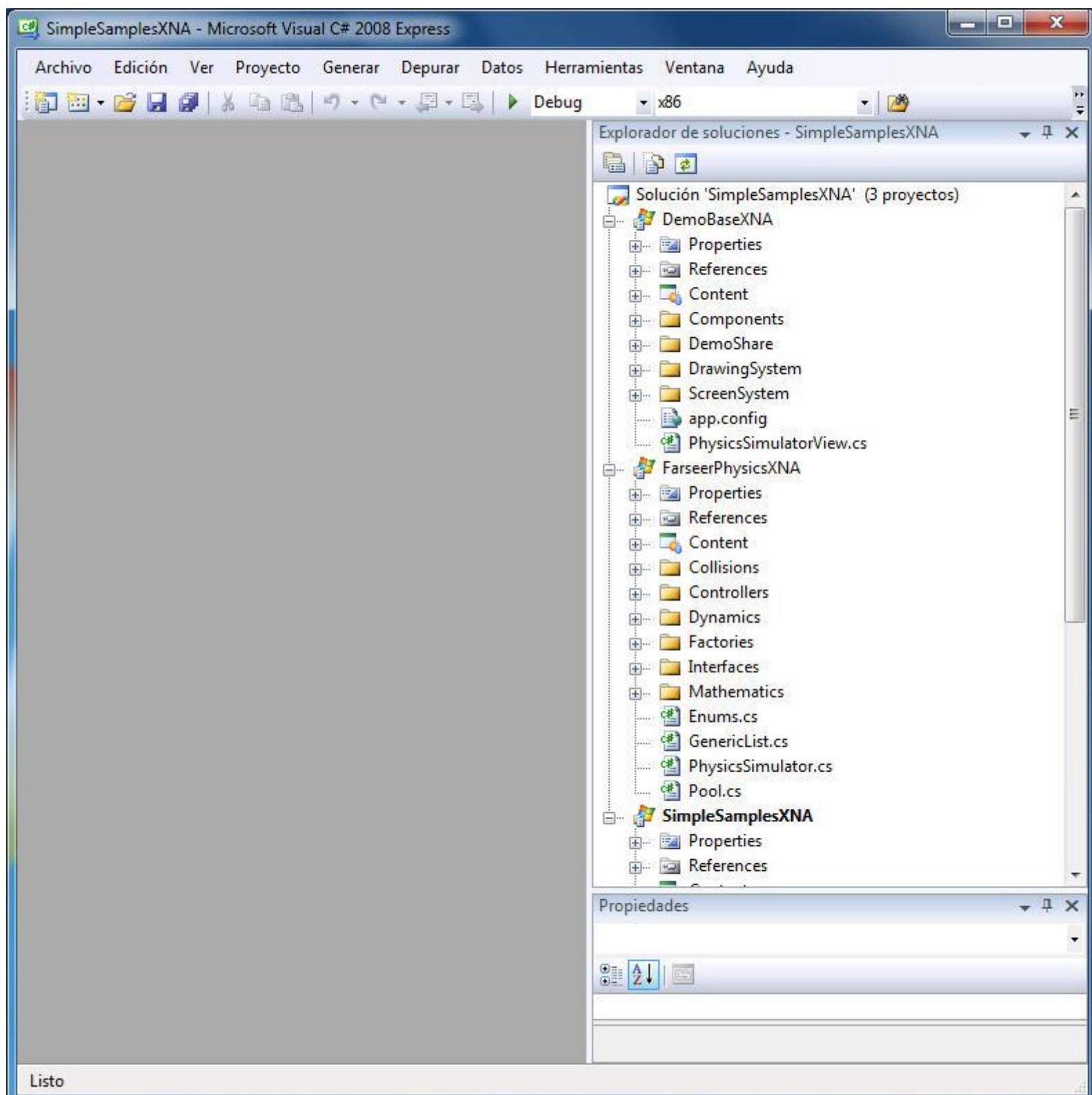


Figura 196: Proyecto de ejemplo con el motor de física

Es recomendable probar todos los ejemplos para ver que nos ofrece ese motor de física.

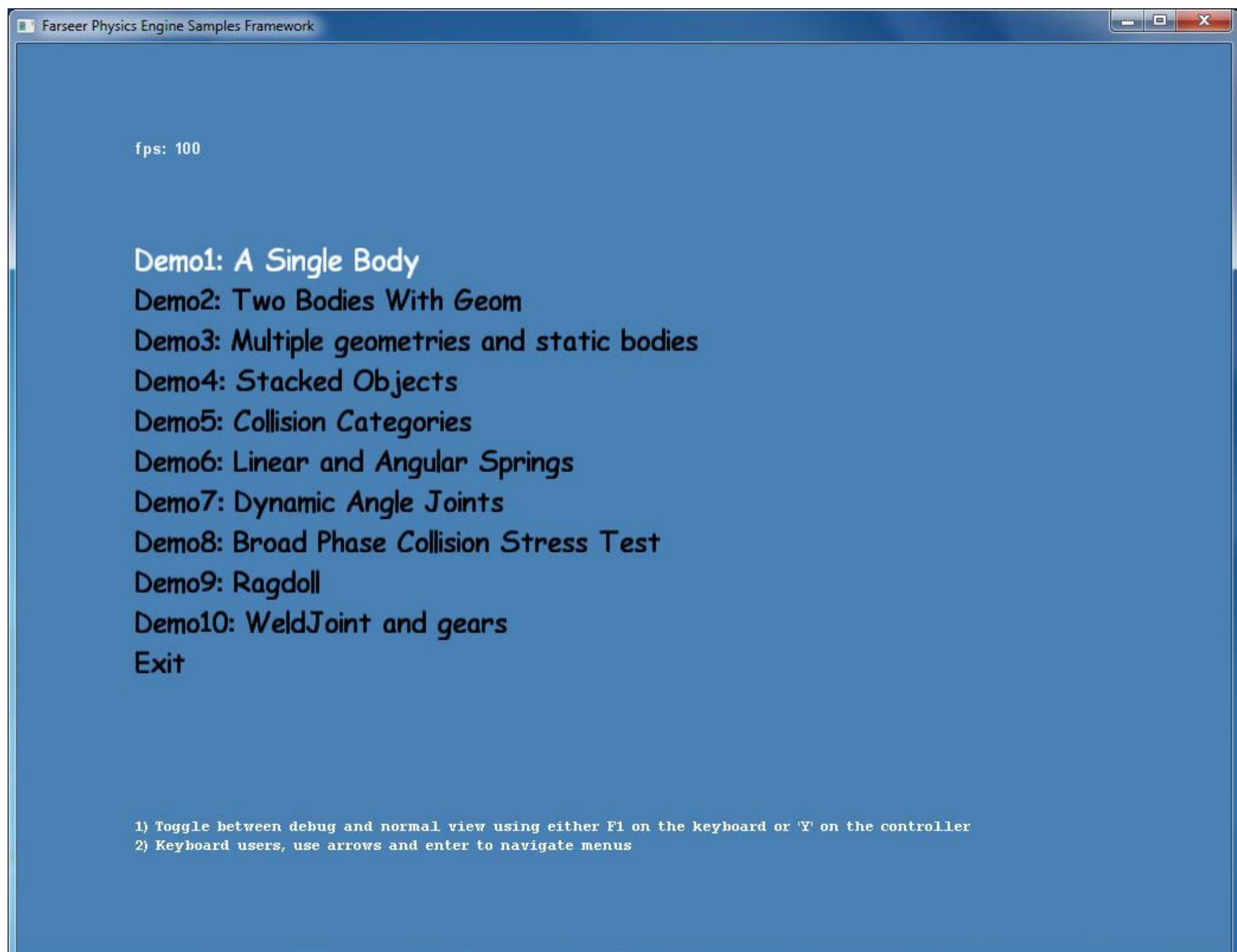


Figura 197: Menú de ejemplos del motor de física

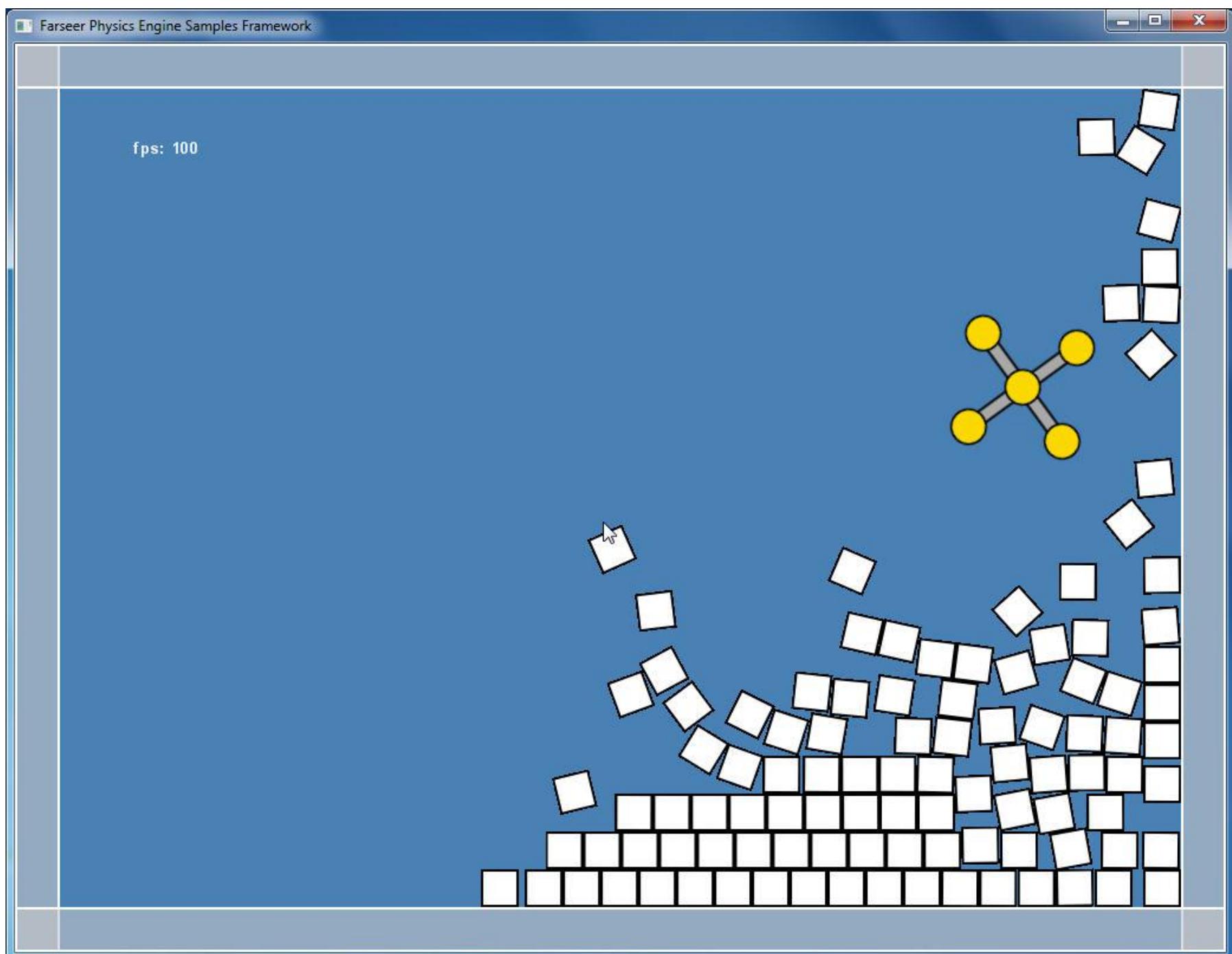


Figura 198: Ejecución de un ejemplo del motor de física

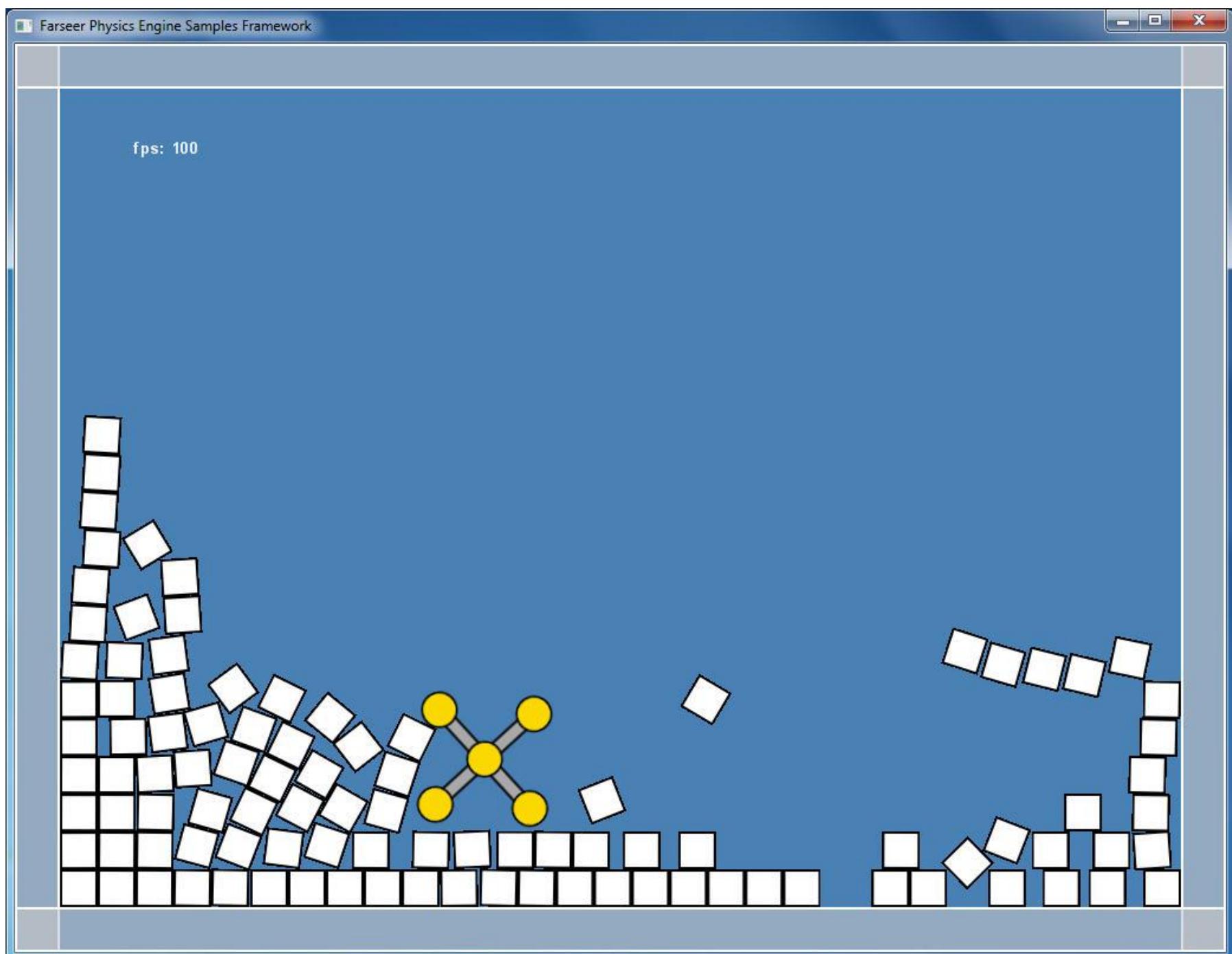


Figura 199: Ejecución de un ejemplo del motor de física

## 35. XNA: Usando un Motor de Física. Colisión entre dos objetos rectangulares.

En esta lección comenzamos a utilizar instrucciones para hacer uso del motor de física. Una vez tenga descargado el motor, estos son los pasos para usarlo:

Paso 1: Creamos un nuevo proyecto de videojuego

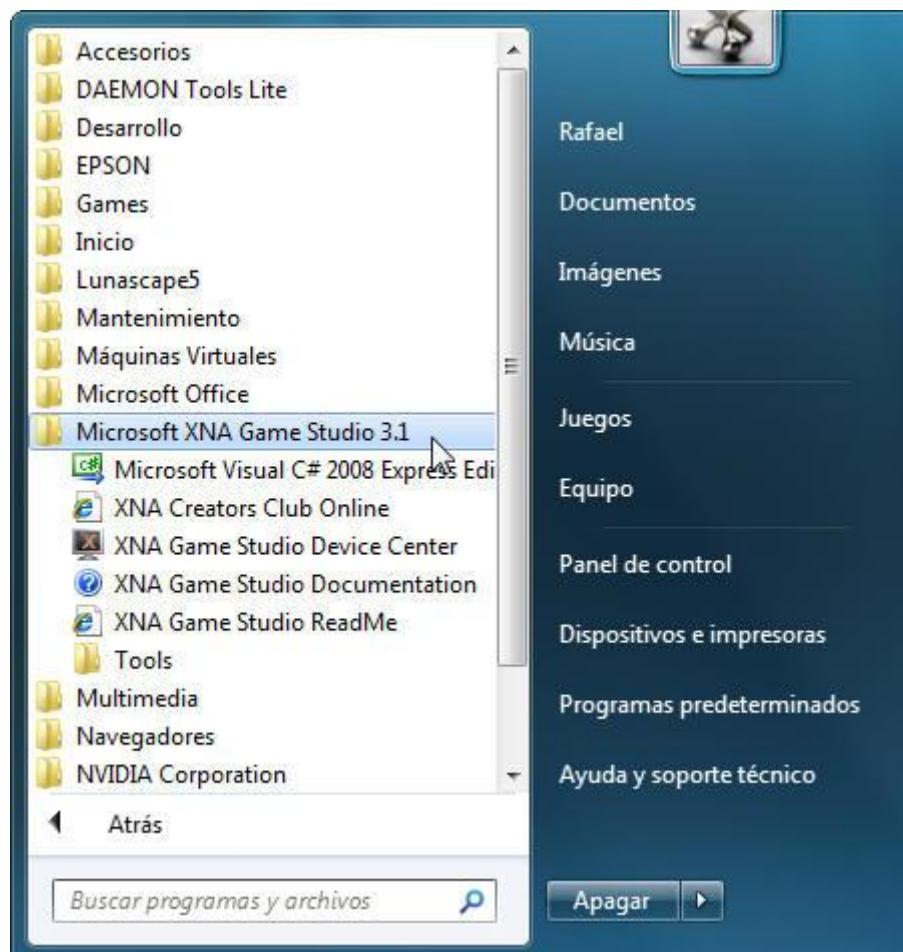


Figura 200: Iniciamos Visual C#

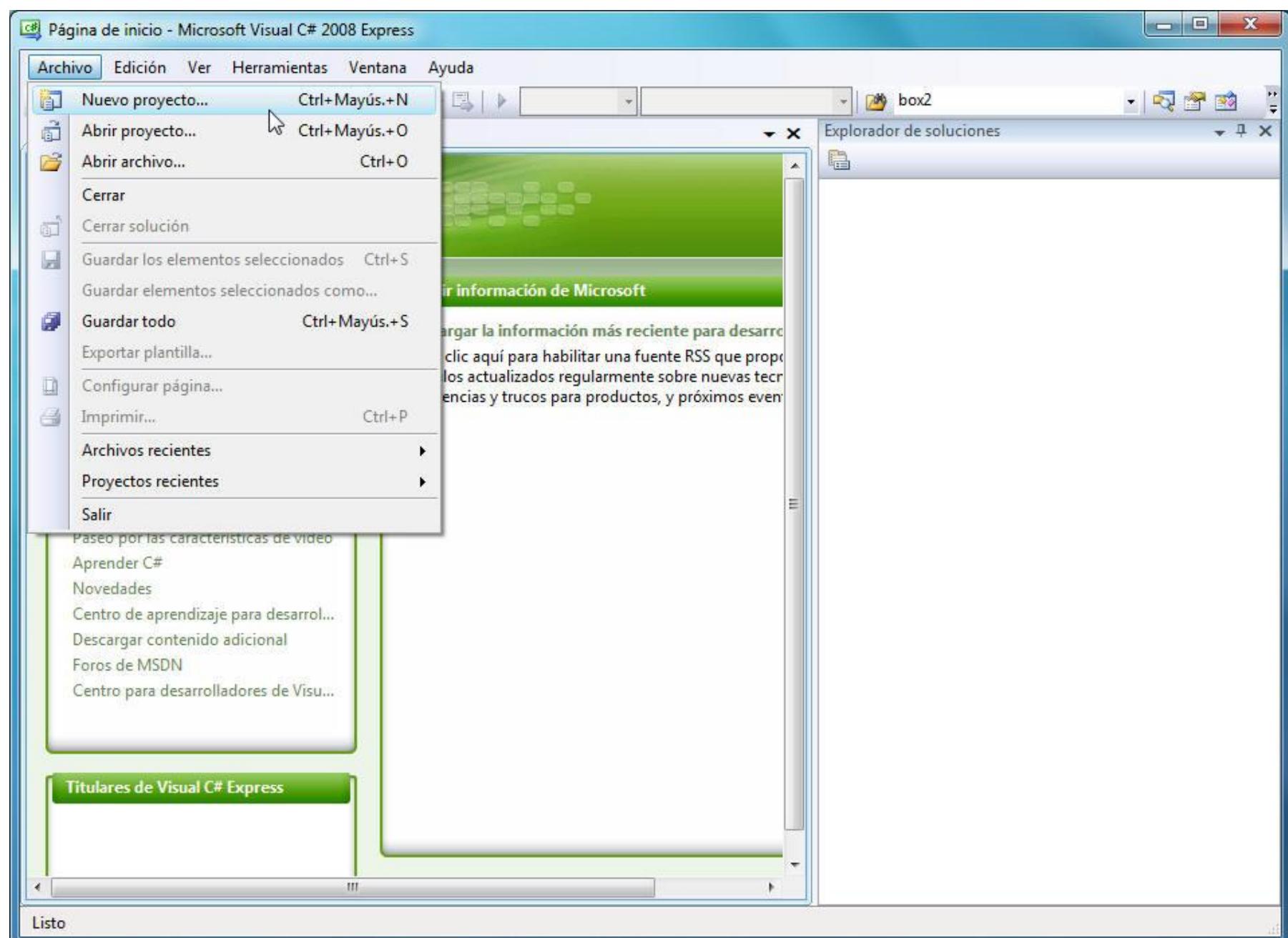


Figura 201: Creamos nuevo proyecto

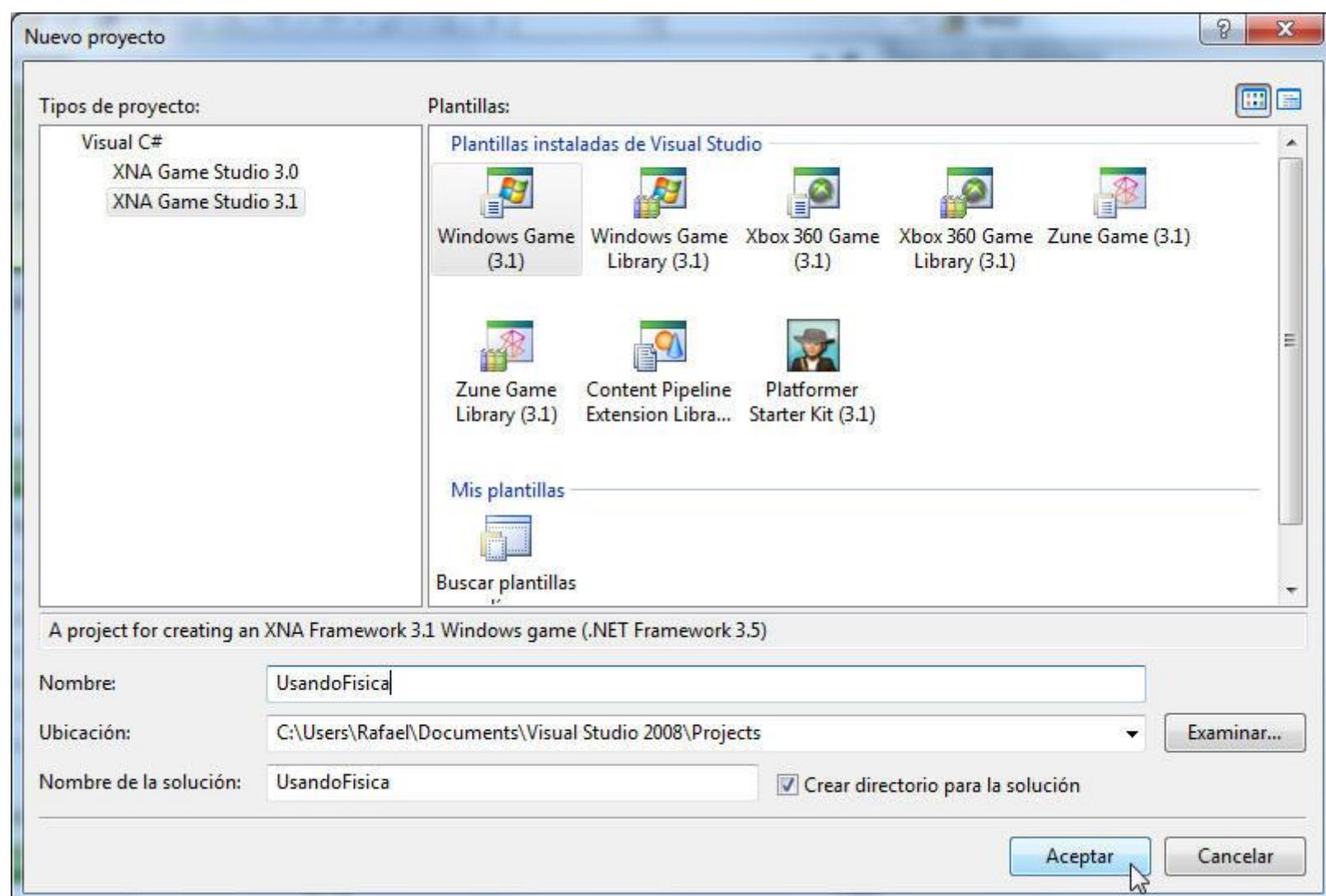


Figura 202: Generamos un proyecto Windows Game (3.1)

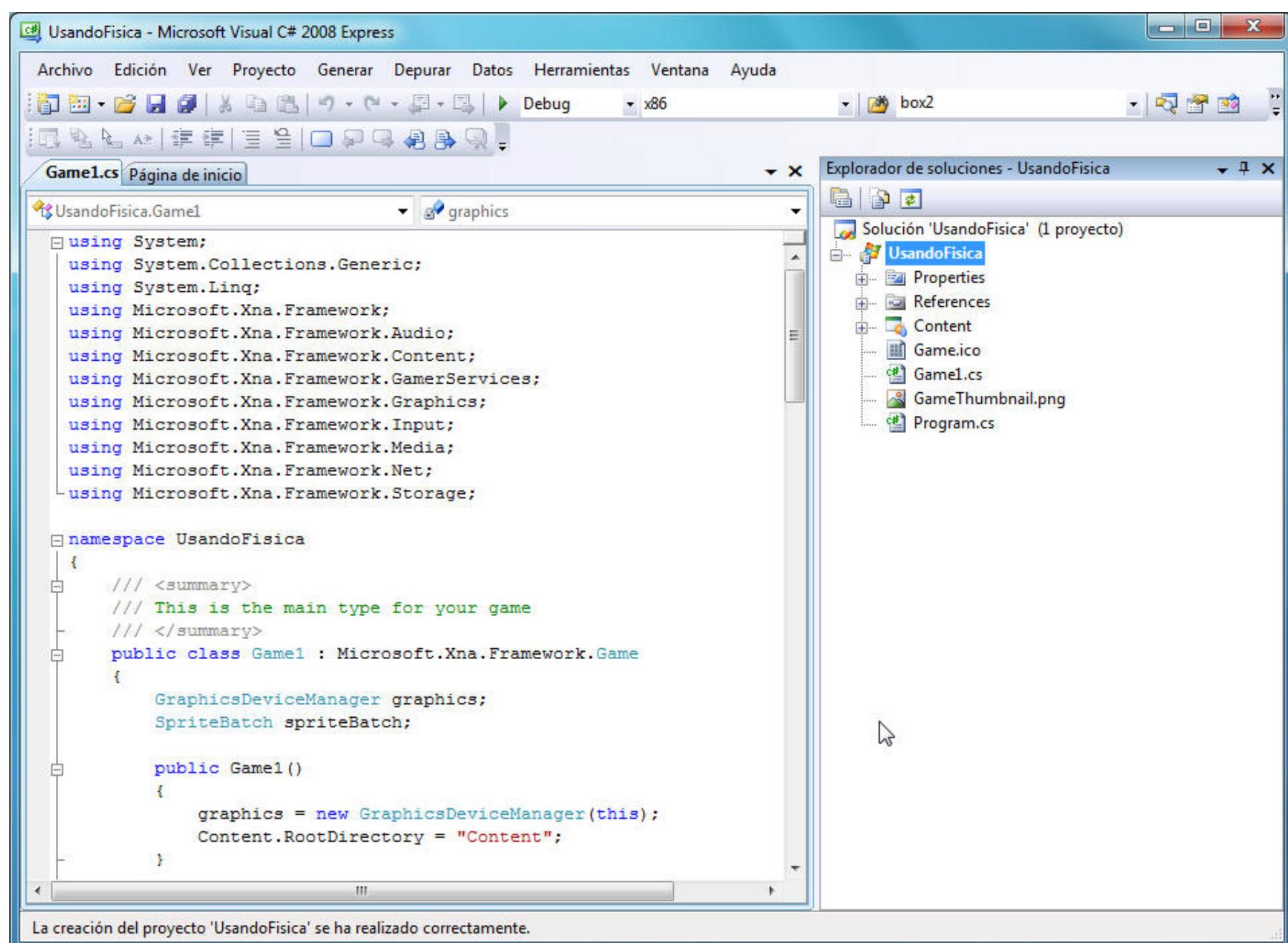


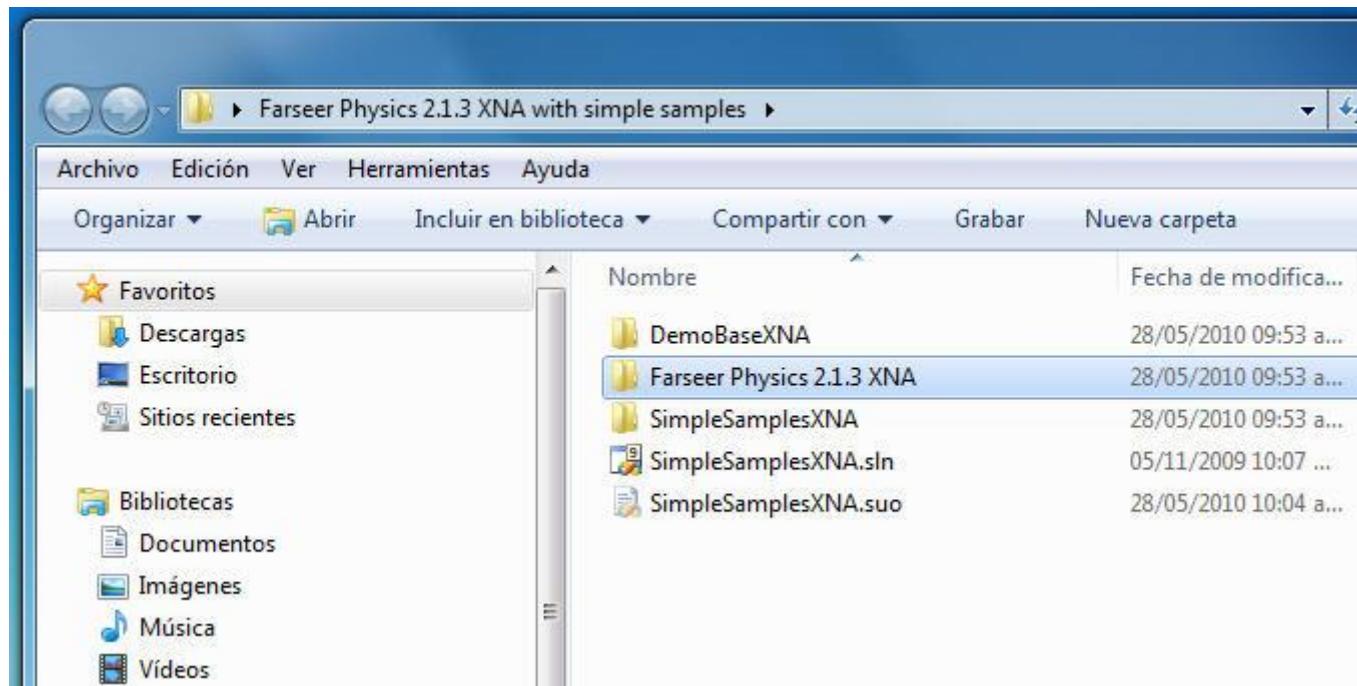
Figura 203: Tenemos nuestro esqueleto de proyecto típico en XNA

Paso 2: Descomprimimos la librería Farseer Physics



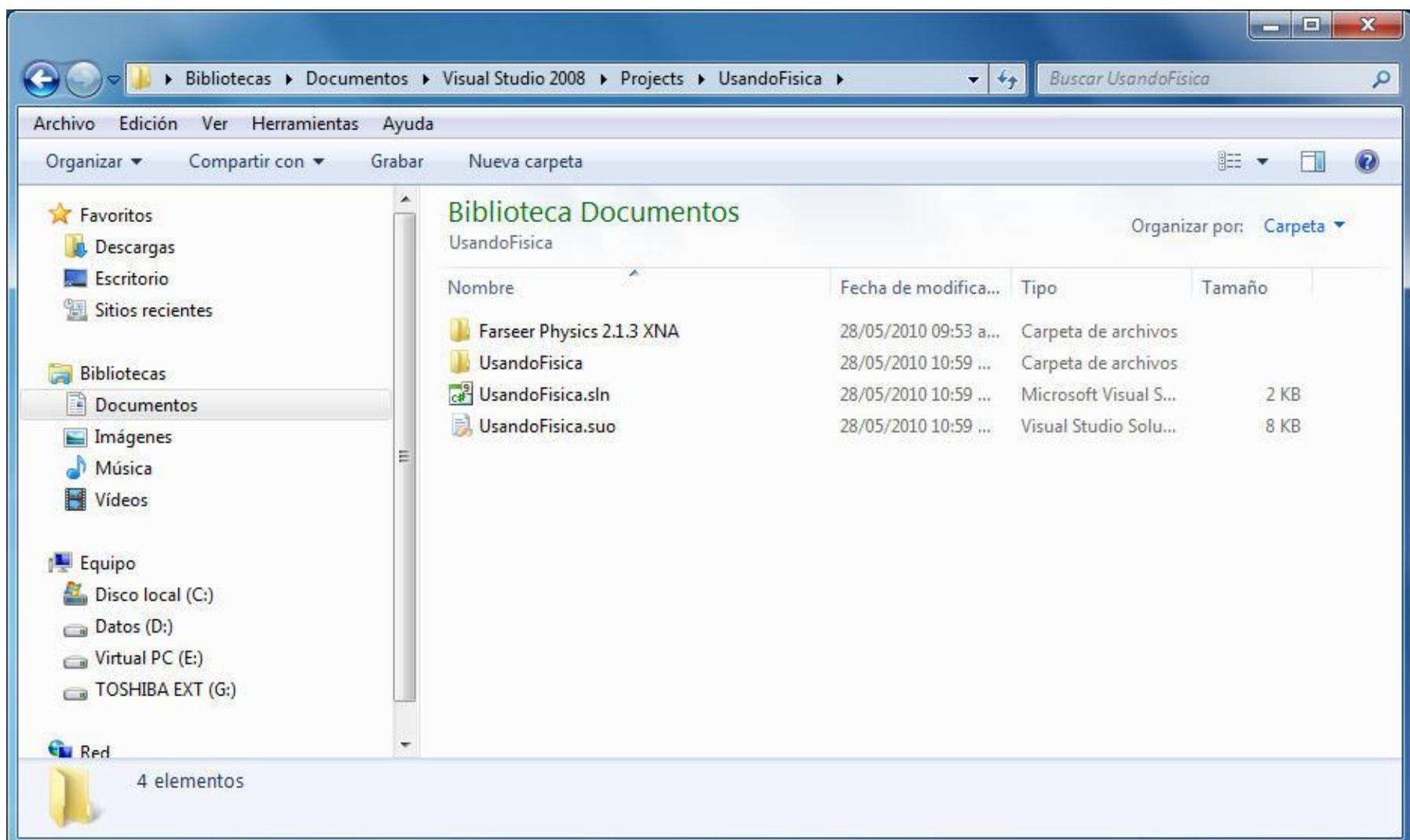
**Figura 204: Se descomprime la librería Farseer Physics**

Nos interesa es la carpeta "Farseer Physics 2.1.3 XNA"



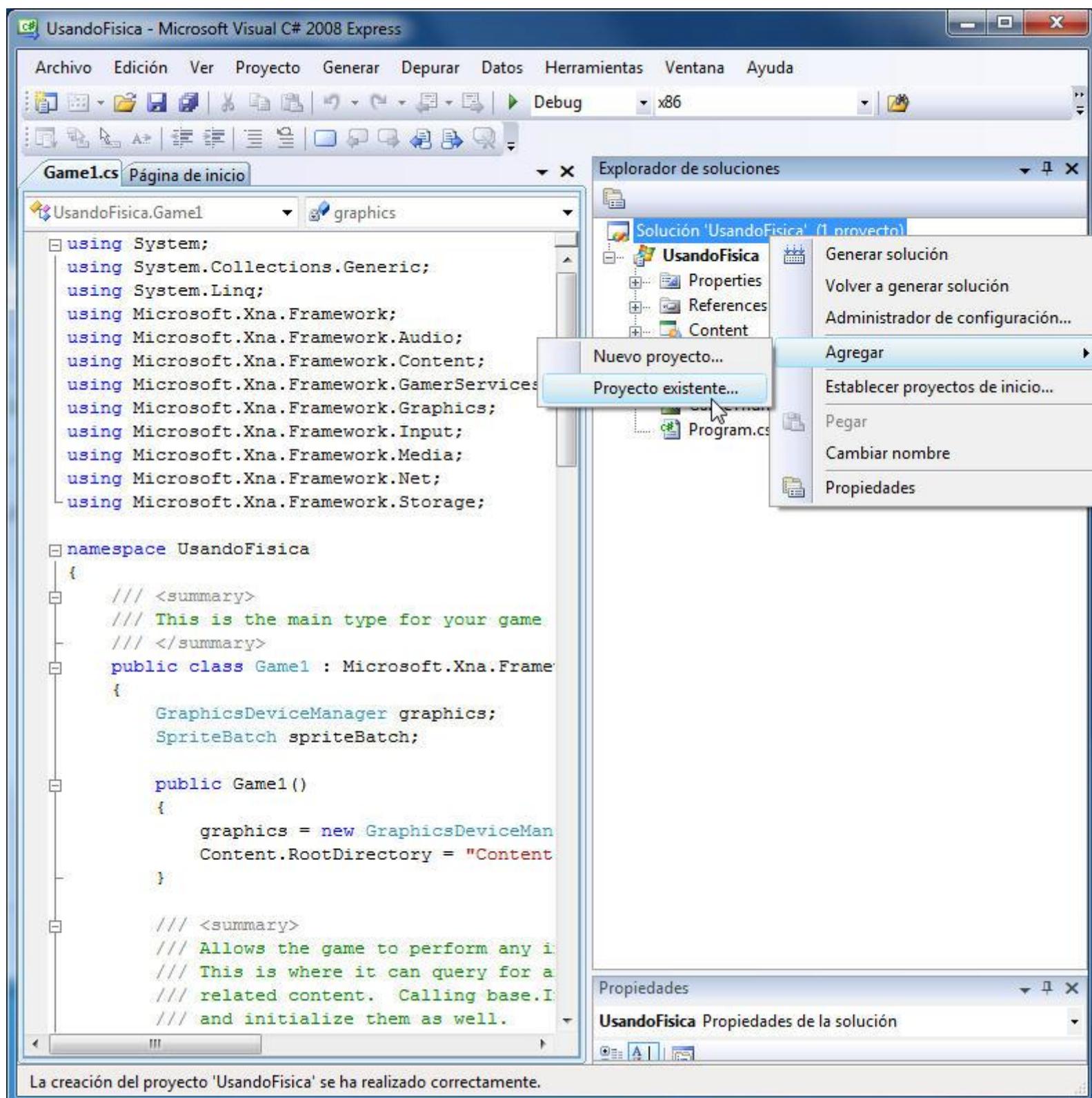
**Figura 205: Seleccionamos la carpeta Farseer Physics 2.1.3 XNA**

Paso 3: Vamos a la carpeta creada por nuestro proyecto de juego. Usualmente es "Visual Studio 2008 -> Projects ->" y arrastramos "Farseer Physics 2.1 XNA" a esa carpeta de nuestro juego.



**Figura 206: A la carpeta del proyecto de videojuego se mueve esa carpeta del motor**

Paso 4: Ahora debemos agregar el proyecto "Farseer Physics 2.1 XNA a la solución del videojuego". Hay que dar clic con el botón derecho del ratón a la solución.



**Figura 207: Damos agregar un proyecto existente**

Buscamos el archivo FarseerPhysicsXNA.csproj (ojo, se supone que Windows tiene habilitado ver las extensiones de archivos conocidos).

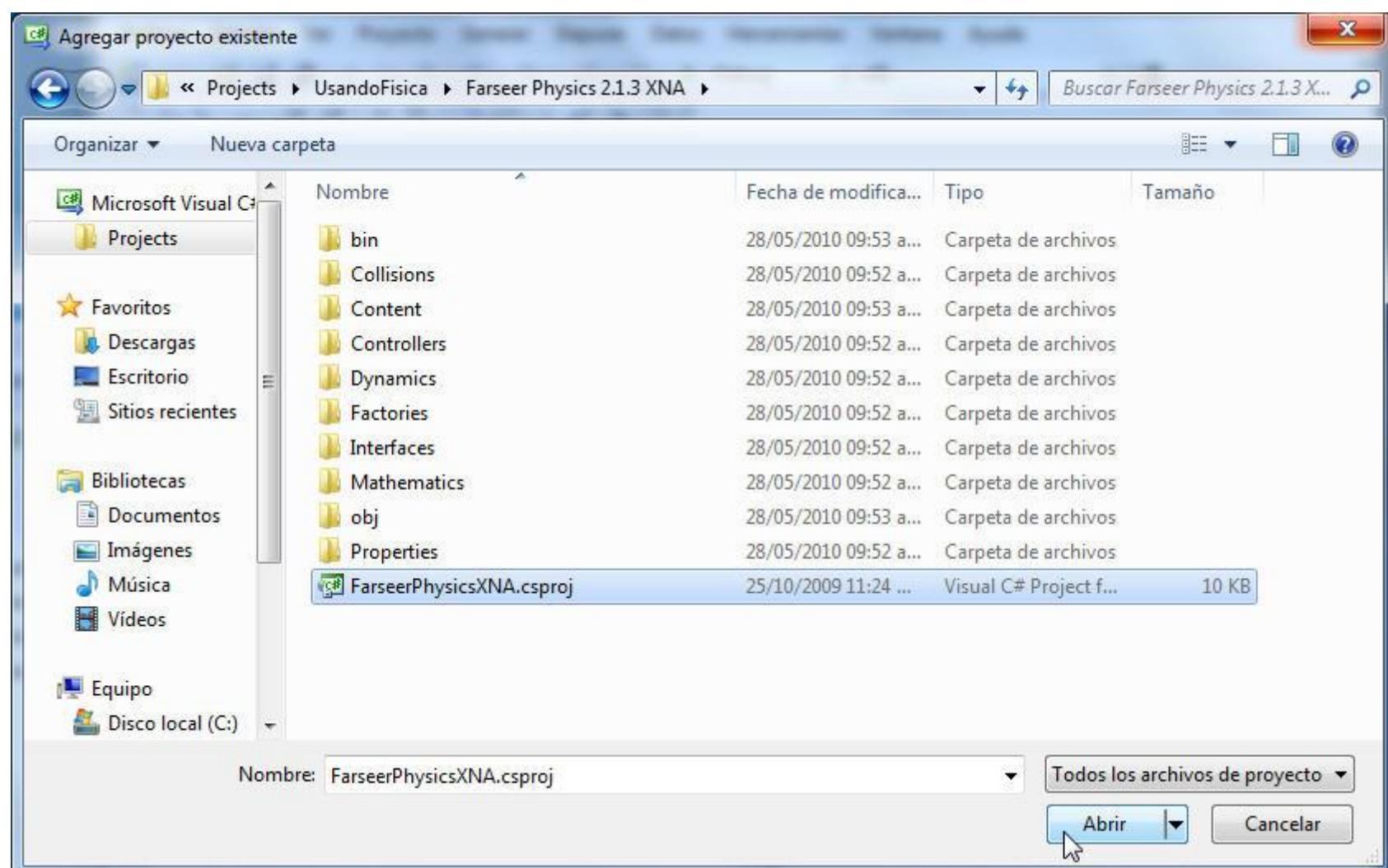


Figura 208: Seleccionamos el archivo de proyecto del motor de física

Quedando así

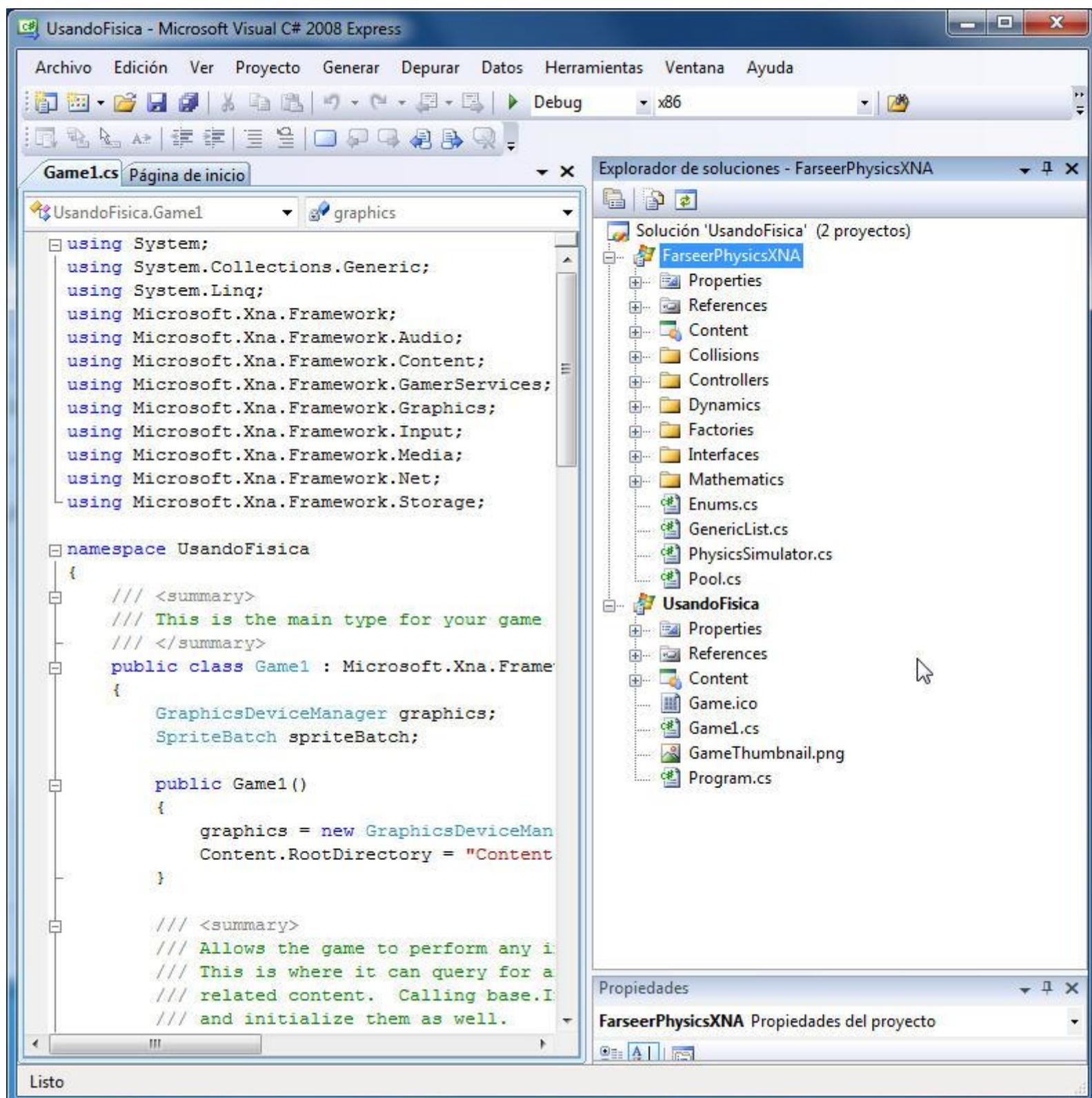


Figura 209: Proyecto cargado dentro de la misma solución del videojuego

Paso 5: Ahora nuestro proyecto de videojuego referencia a "FarseerPhysicsXNA" así:

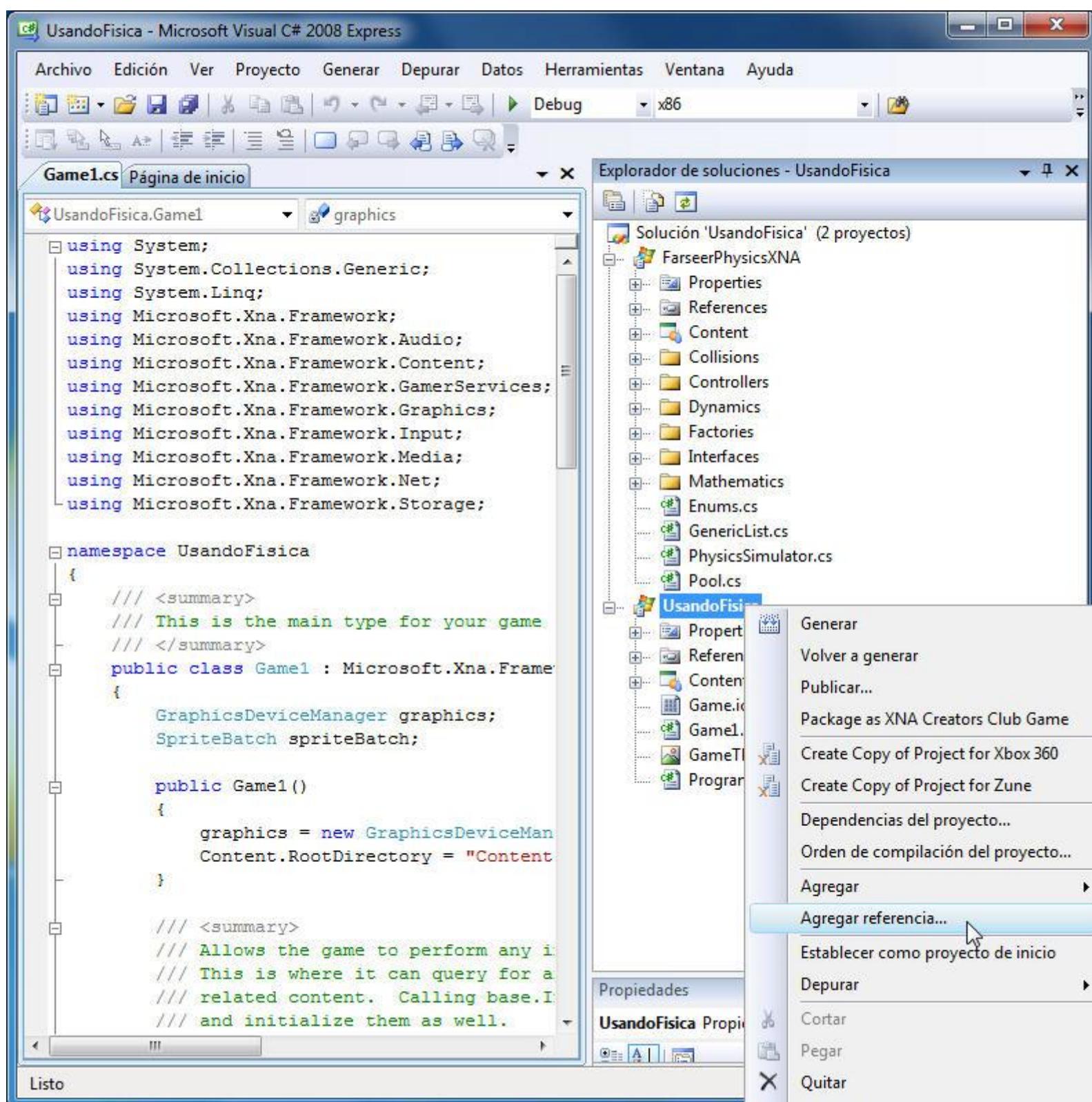


Figura 210: Agregamos referencia

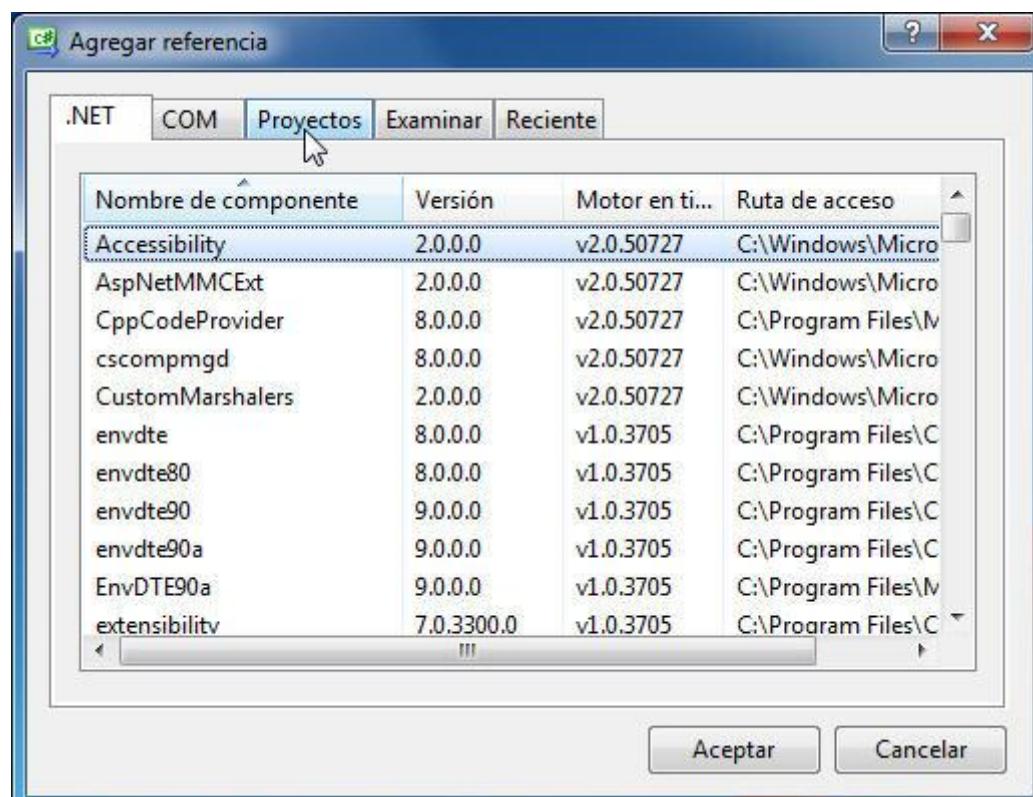


Figura 211: La referencia será el proyecto del motor de física

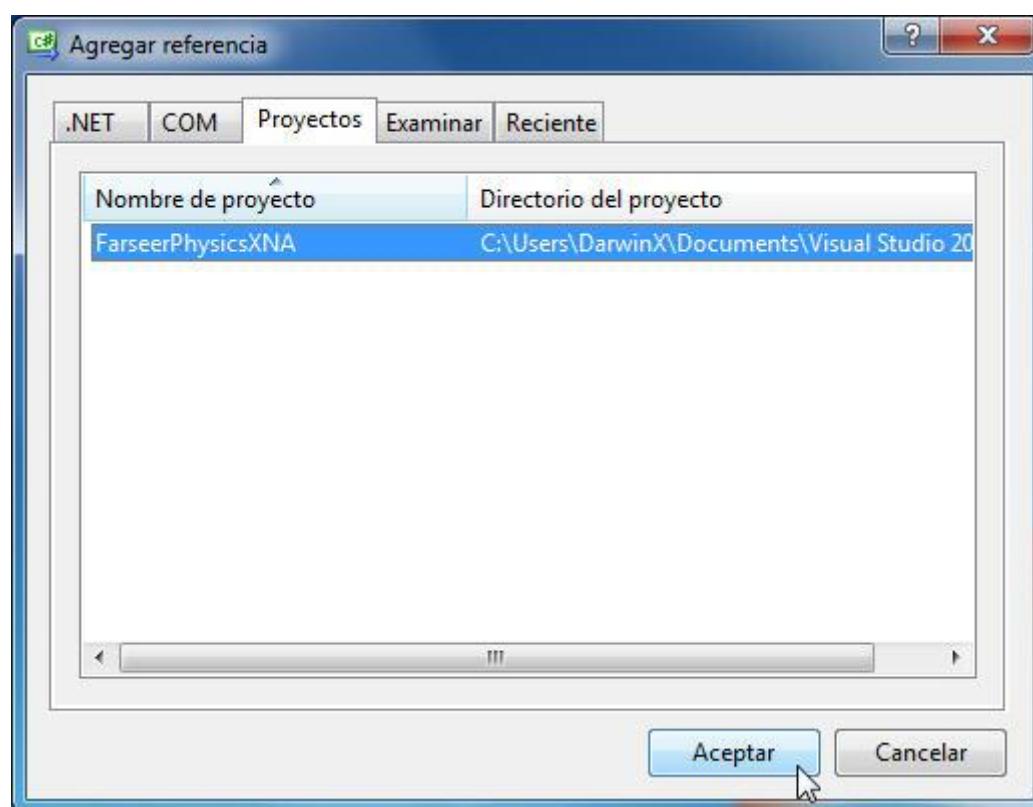


Figura 212: Señalamos el motor de física

Chequeamos que efectivamente nuestro proyecto tiene esa dependencia

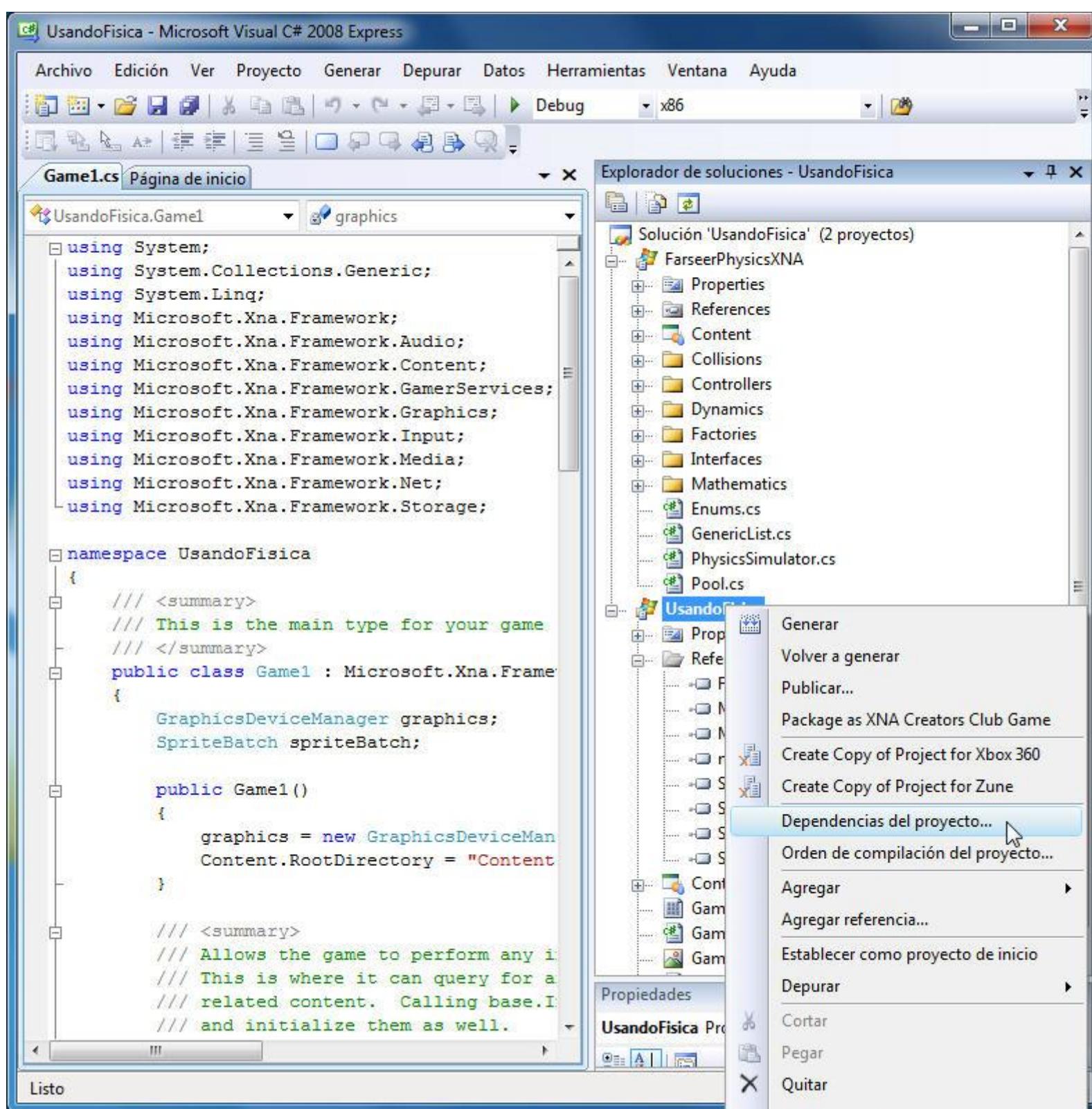
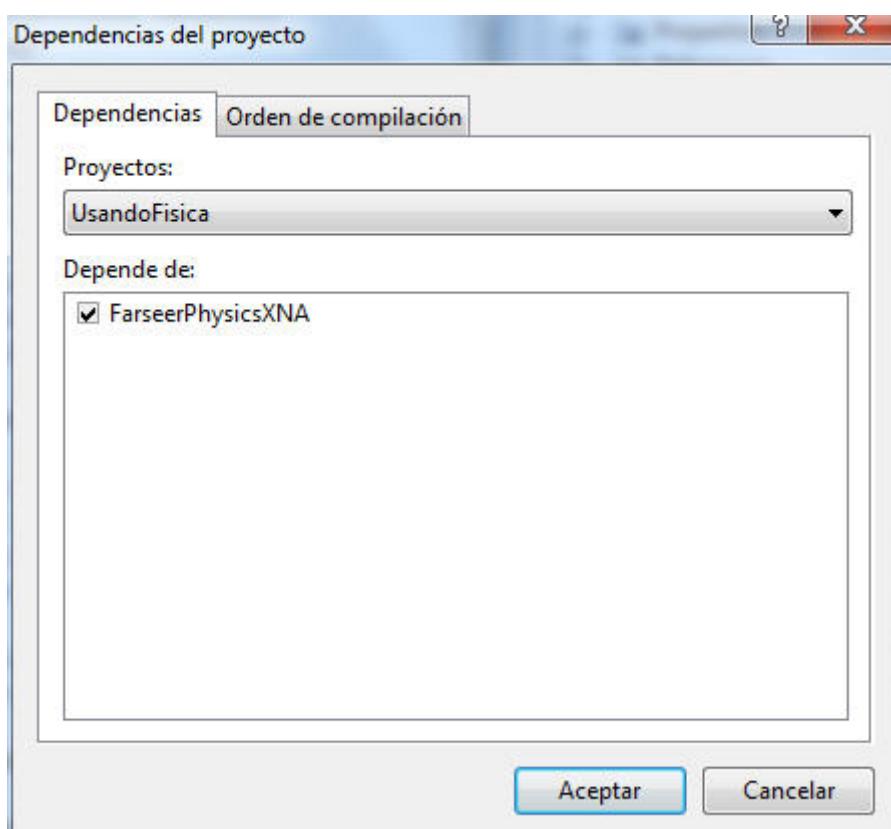
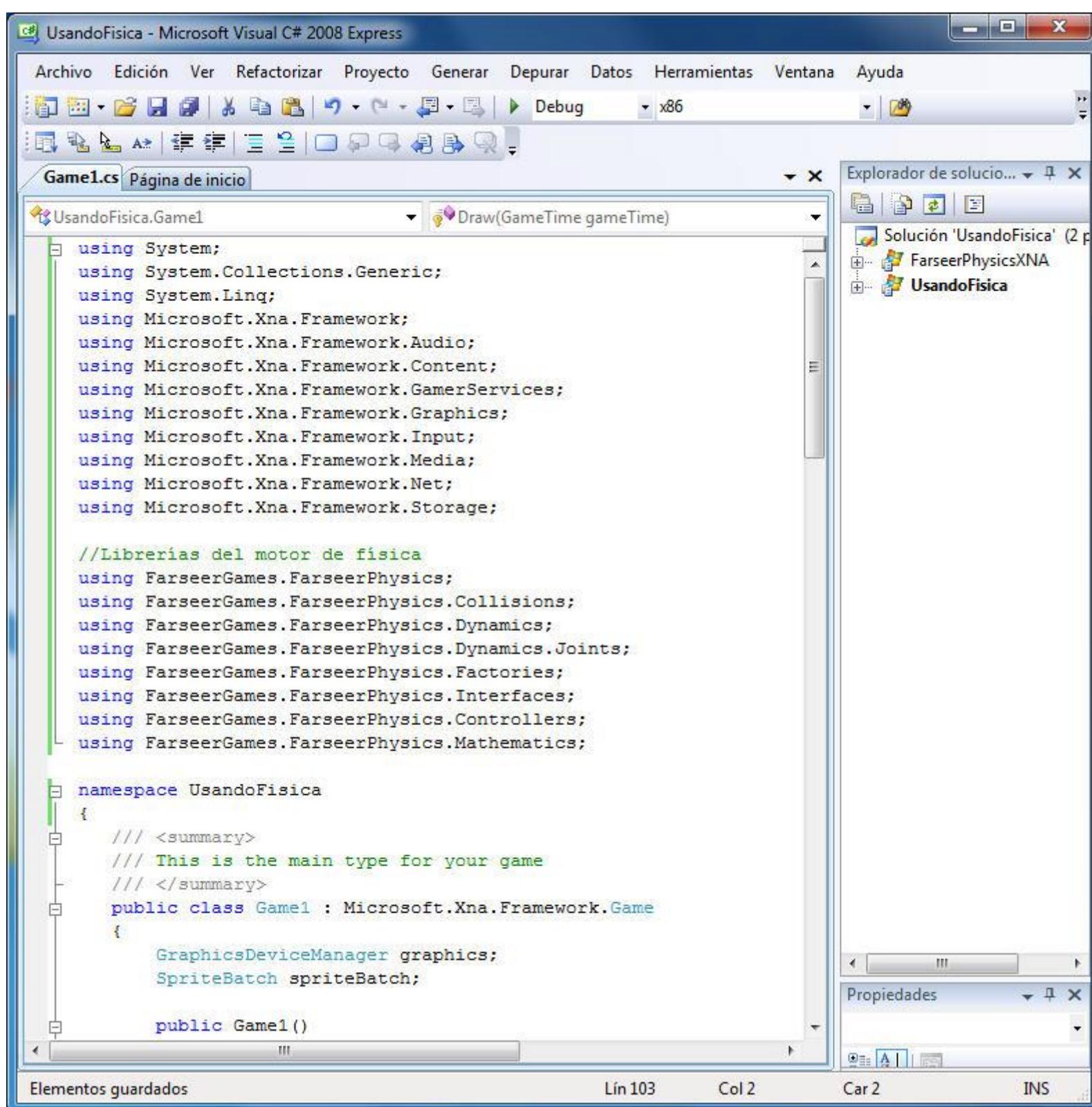


Figura 213: Chequeamos que la referencia haya quedado



**Figura 214: Así debe aparecer la referencia**

Paso 6: Ahora si, vamos a nuestro código y lo primero es agregar las librerías del motor de física



**Figura 215: Agregamos las librerías necesarias para usar el motor de física**

```
//Librerías del motor de física
using FarseerGames.FarseerPhysics;
using FarseerGames.FarseerPhysics.Collisions;
using FarseerGames.FarseerPhysics.Dynamics;
using FarseerGames.FarseerPhysics.Dynamics.Joints;
using FarseerGames.FarseerPhysics.Factories;
using FarseerGames.FarseerPhysics.Interfaces;
using FarseerGames.FarseerPhysics.Controllers;
using FarseerGames.FarseerPhysics.Mathematics;
```

Paso 7:Luego las variables de clase

```
//Las texturas de caja1 y caja 2
Texture2D TexturaCaja1;
Texture2D TexturaCaja2;

//Simulador de física que conecta los diferentes cuerpos en un solo contexto físico
PhysicsSimulator SimuladorFisica;

//Cuerpos (usados por el simulador de física)
Body CuerpoCaja1;
Body CuerpoCaja2;

//Geometría de los cuerpos
Geom GeometriaCaja1;
Geom GeometriaCaja2;
```

Paso 8: En el método Initialize debe ir la configuración de los cuerpos y geometría, hay que tener en cuenta el tamaño (ancho y alto) de la imagen de la caja

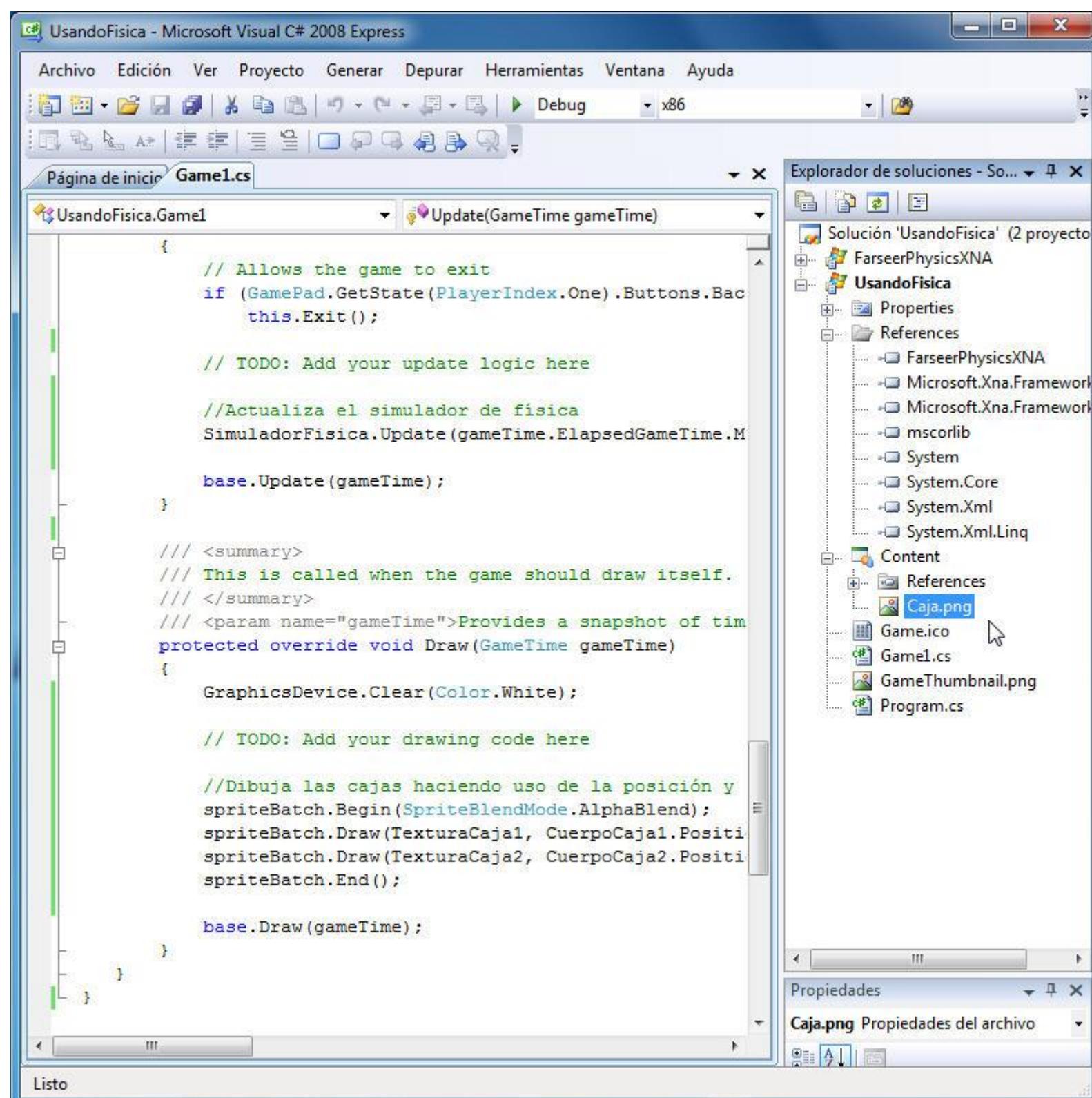


Figura 216: Usamos una textura para la caja en la sección de recursos del videojuego

```
//Configura la tasa de refresco
IsFixedTimeStep = true;
TargetElapsedTIme = new TimeSpan(0, 0, 0, 0, 10); //10ms => 100fps

//Crea el objeto SimuladorFisica y en el constructor se coloca el vector de gravedad
SimuladorFisica = new PhysicsSimulator(new Vector2(0, 200));

//Parámetros de la caja 1
CuerpoCaja1 = BodyFactory.Instance.CreateRectangleBody(SimuladorFisica, 190, 190, 1);
CuerpoCaja1.Position = new Vector2(400, 100);
GeometriaCaja1 = GeomFactory.Instance.CreateRectangleGeom(SimuladorFisica, CuerpoCaja1, 190, 190);

//Parámetros de la caja 2 la cual está inmóvil
CuerpoCaja2 = BodyFactory.Instance.CreateRectangleBody(SimuladorFisica, 190, 190, 1);
CuerpoCaja2.Position = new Vector2(500, 400);
GeometriaCaja2 = GeomFactory.Instance.CreateRectangleGeom(SimuladorFisica, CuerpoCaja2, 190, 190);
CuerpoCaja2.IsStatic = true;
```

Paso 9: En el método LoadContent se llama la textura. En este caso es una imagen de un cuadro amarillo.

```
TexturaCaja1 = Content.Load<Texture2D>("Caja");
TexturaCaja2 = Content.Load<Texture2D>("Caja");
```

Paso 10: En el método Update se actualiza lo que está sucediendo con la física de los diferentes objetos en pantalla

```
//Actualiza el simulador de física
SimuladorFisica.Update(gameTime.ElapsedGameTime.Milliseconds * 0.001f);
```

Paso 11: Finalmente en el método Draw se dibujan las cajas en la posición y giro que el motor de física está calculando.

```
//Dibuja las cajas haciendo uso de la posición y rotación de los cuerpos
spriteBatch.Begin(SpriteBlendMode.AlphaBlend);
spriteBatch.Draw(TexturaCaja1, CuerpoCaja1.Position, null, Color.White, CuerpoCaja1.Rotation, new
Vector2(TexturaCaja1.Width / 2, TexturaCaja1.Height / 2), 1, SpriteEffects.None, 0);
spriteBatch.Draw(TexturaCaja2, CuerpoCaja2.Position, null, Color.White, CuerpoCaja2.Rotation, new
Vector2(TexturaCaja2.Width / 2, TexturaCaja2.Height / 2), 1, SpriteEffects.None, 0);
spriteBatch.End();
```

El código completo de Game1.cs es el siguiente:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

//Librerías del motor de física
using FarseerGames.FarseerPhysics;
using FarseerGames.FarseerPhysics.Collisions;
using FarseerGames.FarseerPhysics.Dynamics;
using FarseerGames.FarseerPhysics.Dynamics.Joints;
using FarseerGames.FarseerPhysics.Factories;
using FarseerGames.FarseerPhysics.Interfaces;
using FarseerGames.FarseerPhysics.Controllers;
using FarseerGames.FarseerPhysics.Mathematics;

namespace UsandoFisica
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        //Las texturas de caja 1 y caja 2
        Texture2D TexturaCaja1;
        Texture2D TexturaCaja2;

        //Simulador de física que conecta los diferentes cuerpos en un solo contexto físico
        PhysicsSimulator SimuladorFisica;

        //Cuerpos (usados por el simulador de física)
        Body CuerpoCaja1;
        Body CuerpoCaja2;

        //Geometría de los cuerpos
        Geom GeometriaCaja1;
        Geom GeometriaCaja2;
```

```

public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";
}

/// <summary>
/// Allows the game to perform any initialization it needs to before starting to run.
/// This is where it can query for any required services and load any non-graphic
/// related content. Calling base.Initialize will enumerate through any components
/// and initialize them as well.
/// </summary>
protected override void Initialize()
{
    // TODO: Add your initialization logic here
    //Configura la tasa de refresco
    IsFixedTimeStep = true;
    TargetElapsedTime = new TimeSpan(0, 0, 0, 0, 10); //10ms => 100fps

    //Crea el objeto SimuladorFisica y en el constructor se coloca el vector de gravedad
    SimuladorFisica = new PhysicsSimulator(new Vector2(0, 200));

    //Parámetros de la caja 1
    CuerpoCaja1 = BodyFactory.Instance.CreateRectangleBody(SimuladorFisica, 190, 190, 1);
    CuerpoCaja1.Position = new Vector2(400, 100);
    GeometriaCaja1 = GeomFactory.Instance.CreateRectangleGeom(SimuladorFisica, CuerpoCaja1, 190, 190);

    //Parámetros de la caja 2 la cual está inmóvil
    CuerpoCaja2 = BodyFactory.Instance.CreateRectangleBody(SimuladorFisica, 190, 190, 1);
    CuerpoCaja2.Position = new Vector2(500, 400);
    GeometriaCaja2 = GeomFactory.Instance.CreateRectangleGeom(SimuladorFisica, CuerpoCaja2, 190, 190);
    CuerpoCaja2.IsStatic = true;

    base.Initialize();
}

/// <summary>
/// LoadContent will be called once per game and is the place to load
/// all of your content.
/// </summary>
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here
    TexturaCaja1 = Content.Load<Texture2D>("Caja");
    TexturaCaja2 = Content.Load<Texture2D>("Caja");
}

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // TODO: Add your update logic here

    //Actualiza el simulador de física
    SimuladorFisica.Update(gameTime.ElapsedGameTime.Milliseconds * 0.001f);

    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here

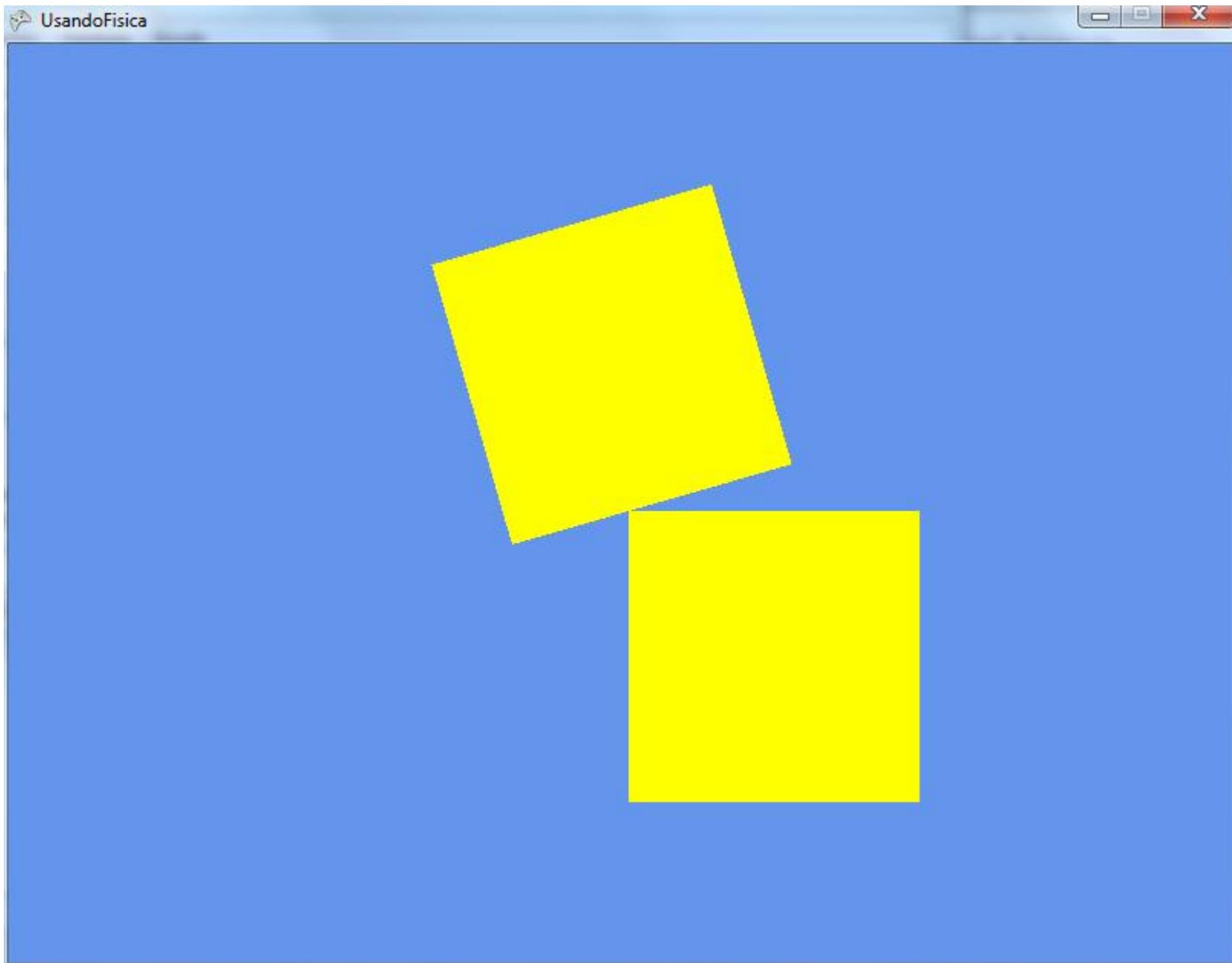
    //Dibuja las cajas haciendo uso de la posición y rotación de los cuerpos
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

    spriteBatch.Draw(TexturaCaja1, CuerpoCaja1.Position, null, Color.White, CuerpoCaja1.Rotation,

```

```
new Vector2(TexturaCaja1.Width / 2, TexturaCaja1.Height / 2), 1, SpriteEffects.None, 0);  
spriteBatch.Draw(TexturaCaja2, CuerpoCaja2.Position, null, Color.White, CuerpoCaja2.Rotation,  
new Vector2(TexturaCaja2.Width / 2, TexturaCaja2.Height / 2), 1, SpriteEffects.None, 0);  
spriteBatch.End();  
  
base.Draw(gameTime);  
}  
}  
}
```

Este es el resultado:



**Figura 217: Ejecutando el ejemplo de la colisión de dos cajas**

## 36. XNA: Usando un Motor de Física. Torque y control por teclado.

En esta lección trataremos del desplazamiento y del giro (torque) de una figura con el simulador de física.

Paso 1: Se genera un nuevo proyecto y se le hace referencia al motor de física como se vio en el punto anterior.

Paso 2: Las librerías que deben usarse en el programa

```
//Librerías del motor de física
using FarseerGames.FarseerPhysics;
using FarseerGames.FarseerPhysics.Collisions;
using FarseerGames.FarseerPhysics.Dynamics;
using FarseerGames.FarseerPhysics.Factories;
```

Paso 3: Se requiere una textura en forma rectangular

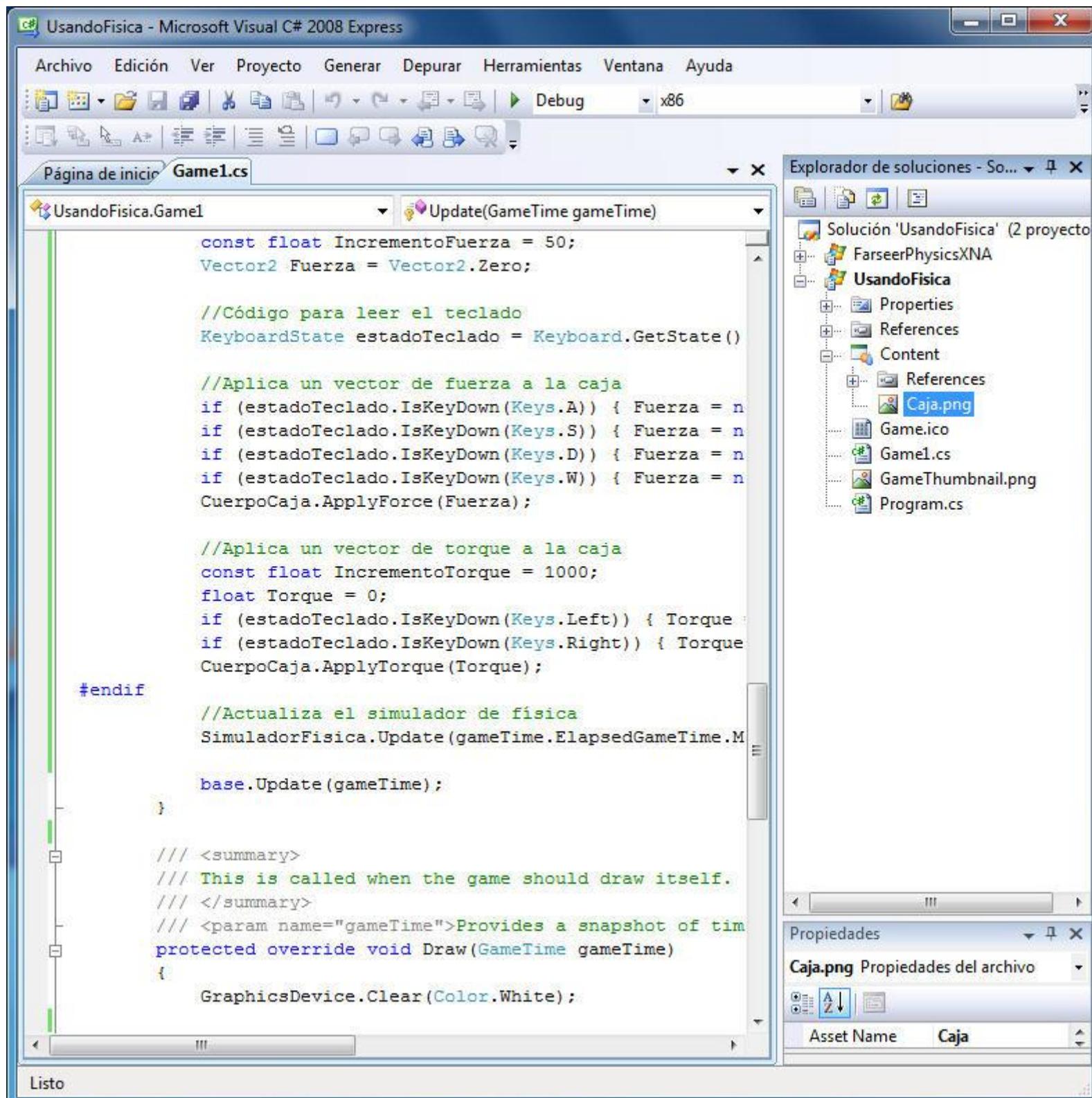


Figura 218: Se hace uso de una textura para la caja

Paso 4: Las variables de clase

```
//Textura del cuadrado
private Texture2D TexturaCaja;

//Simulador de física que conecta los diferentes cuerpos en un solo contexto físico
PhysicsSimulator SimuladorFisica;

//Cuerpo (usado por el simulador de física)
Body CuerpoCaja;
```

Paso 5: Lo que debe ir en el LoadContent

```
//Crea el objeto SimuladorFisica y en el constructor se coloca el vector de gravedad
SimuladorFisica = new PhysicsSimulator(new Vector2(0, 0));

// TODO: use this.Content to load your game content here
TexturaCaja = Content.Load<Texture2D>("Caja");
CuerpoCaja = BodyFactory.Instance.CreateRectangleBody(SimuladorFisica, TexturaCaja.Width, TexturaCaja.Height, 1);
```

```
CuerpoCaja.Position = new Vector2(400, 100);
```

#### Paso 6: Lo que debe ir en Update

```
//Esta es una compilación condicional. El teclado y el mouse solo son leídos si la aplicación ejecuta en un PC
#ifndef !XBOX
const float IncrementoFuerza = 50;
Vector2 Fuerza = Vector2.Zero;

//Código para leer el teclado
KeyboardState estadoTeclado = Keyboard.GetState();

//Aplica un vector de fuerza a la caja
if (estadoTeclado.IsKeyDown(Keys.A)) { Fuerza = new Vector2(-IncrementoFuerza, 0); }
if (estadoTeclado.IsKeyDown(Keys.S)) { Fuerza = new Vector2(0, IncrementoFuerza); }
if (estadoTeclado.IsKeyDown(Keys.D)) { Fuerza = new Vector2(IncrementoFuerza, 0); }
if (estadoTeclado.IsKeyDown(Keys.W)) { Fuerza = new Vector2(0, -IncrementoFuerza); }
CuerpoCaja.ApplyForce(Fuerza);

//Aplica un vector de torque a la caja
const float IncrementoTorque = 1000;
float Torque = 0;
if (estadoTeclado.IsKeyDown(Keys.Left)) { Torque = -IncrementoTorque; }
if (estadoTeclado.IsKeyDown(Keys.Right)) { Torque = IncrementoTorque; }
CuerpoCaja.ApplyTorque(Torque);
#endif
//Actualiza el simulador de física
SimuladorFisica.Update(gameTime.ElapsedGameTime.Milliseconds * 0.001f);
```

#### Paso 7: Lo que debe ir en Draw

```
//Dibuja las cajas haciendo uso de la posición y rotación de los cuerpos dado por el motor de física
spriteBatch.Begin(SpriteBlendMode.AlphaBlend);
spriteBatch.Draw(TexturaCaja, CuerpoCaja.Position, null, Color.White, CuerpoCaja.Rotation,
    new Vector2(TexturaCaja.Width / 2, TexturaCaja.Height / 2), 1, SpriteEffects.None, 0);
spriteBatch.End();
```

Este es el código completo de Game1.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

//Librerías del motor de física
using FarseerGames.FarseerPhysics;
using FarseerGames.FarseerPhysics.Collisions;
using FarseerGames.FarseerPhysics.Dynamics;
using FarseerGames.FarseerPhysics.Factories;

namespace UsandoFisica
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        //Textura del cuadrado
        private Texture2D TexturaCaja;

        //Simulador de física que conecta los diferentes cuerpos en un solo contexto físico
        PhysicsSimulator SimuladorFisica;

        //Cuerpo (usado por el simulador de física)
        Body CuerpoCaja;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
```

```

protected override void Initialize()
{
    // TODO: Add your initialization logic here

    base.Initialize();
}

/// <summary>
/// LoadContent will be called once per game and is the place to load
/// all of your content.
/// </summary>
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);

    // TODO: use this.Content to load your game content here

    //Crea el objeto SimuladorFisica y en el constructor se coloca el vector de gravedad
    SimuladorFisica = new PhysicsSimulator(new Vector2(0, 0));

    // TODO: use this.Content to load your game content here
    TexturaCaja = Content.Load<Texture2D>("Caja");
    CuerpoCaja = BodyFactory.Instance.CreateRectangleBody(SimuladorFisica, TexturaCaja.Width, TexturaCaja.Height, 1);
    CuerpoCaja.Position = new Vector2(400, 200);
}

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // TODO: Add your update logic here
    //Esta es una compilación condicional. El teclado y el mouse solo son leídos si la aplicación ejecuta en un PC
#ifndef !XBOX
    const float IncrementoFuerza = 50;
    Vector2 Fuerza = Vector2.Zero;

    //Código para leer el teclado
    KeyboardState estadoTeclado = Keyboard.GetState();

    //Aplica un vector de fuerza a la caja
    if (estadoTeclado.IsKeyDown(Keys.A)) { Fuerza = new Vector2(-IncrementoFuerza, 0); }
    if (estadoTeclado.IsKeyDown(Keys.S)) { Fuerza = new Vector2(0, IncrementoFuerza); }
    if (estadoTeclado.IsKeyDown(Keys.D)) { Fuerza = new Vector2(IncrementoFuerza, 0); }
    if (estadoTeclado.IsKeyDown(Keys.W)) { Fuerza = new Vector2(0, -IncrementoFuerza); }
    CuerpoCaja.ApplyForce(Fuerza);

    //Aplica un vector de torque a la caja
    const float IncrementoTorque = 1000;
    float Torque = 0;
    if (estadoTeclado.IsKeyDown(Keys.Left)) { Torque = -IncrementoTorque; }
    if (estadoTeclado.IsKeyDown(Keys.Right)) { Torque = IncrementoTorque; }
    CuerpoCaja.ApplyTorque(Torque);
#endif
    //Actualiza el simulador de física
    SimuladorFisica.Update(gameTime.ElapsedGameTime.Milliseconds * 0.001f);

    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.White);

    // TODO: Add your drawing code here

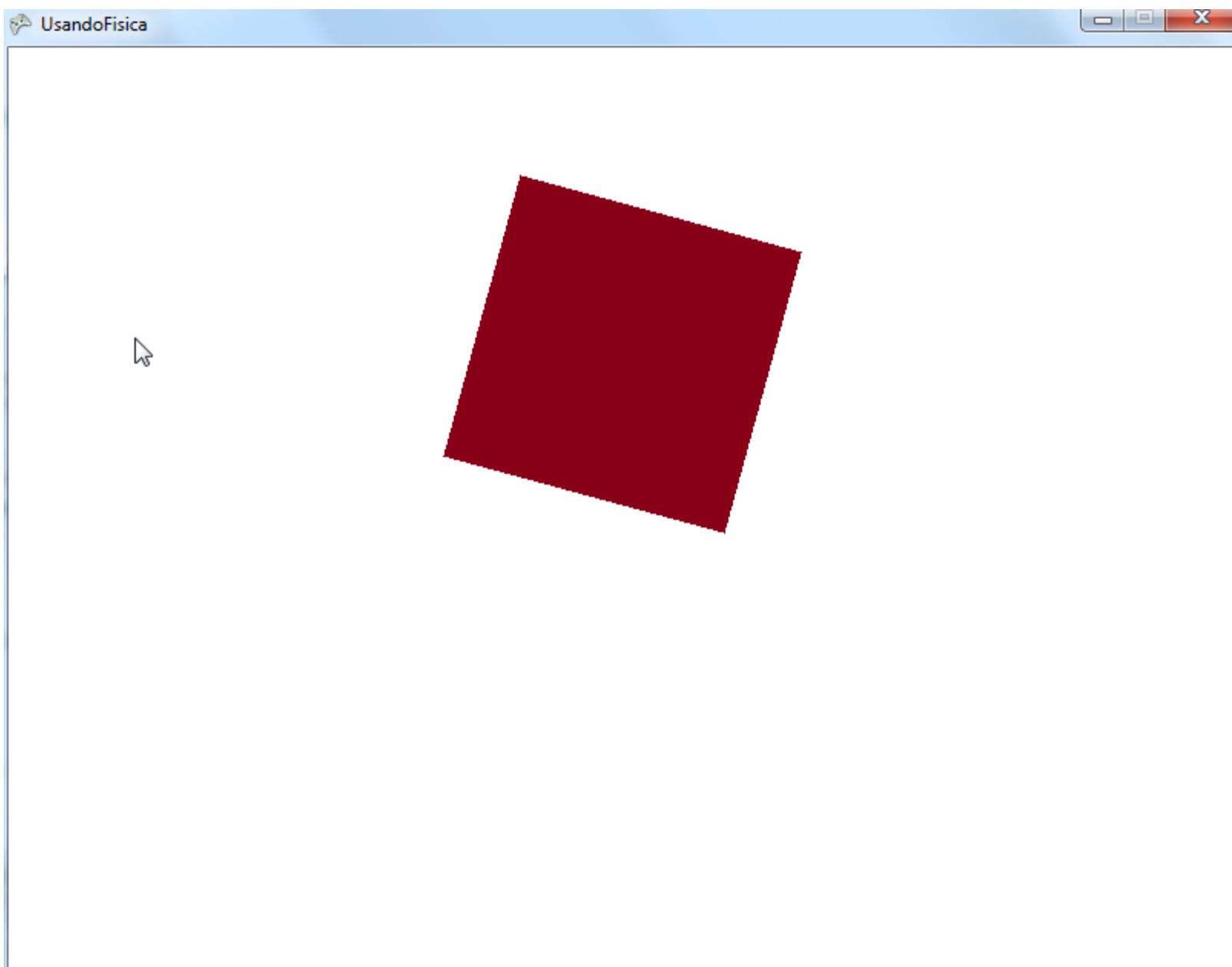
    //Dibuja las cajas haciendo uso de la posición y rotación de los cuerpos dado por el motor de física
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);
    spriteBatch.Draw(TexturaCaja, CuerpoCaja.Position, null, Color.White, CuerpoCaja.Rotation,
        new Vector2(TexturaCaja.Width / 2, TexturaCaja.Height / 2), 1, SpriteEffects.None, 0);
    spriteBatch.End();

    base.Draw(gameTime);
}

```

{  
}  
}

Este es el resultado:



**Figura 219: Ejemplo de giro de una caja usando el motor de física**

## 37. XNA: Usando un Motor de Física. Colisión con múltiples objetos rectangulares.

En esta lección trataremos de colisiones entre varios objetos

**Paso 1:** Se genera un nuevo proyecto y se le hace referencia al motor de física como se vio en el capítulo anterior.

**Paso 2:** Las librerías que deben usarse en el programa

```
//Librerías del motor de física
using FarseerGames.FarseerPhysics;
using FarseerGames.FarseerPhysics.Collisions;
using FarseerGames.FarseerPhysics.Dynamics;
using FarseerGames.FarseerPhysics.Factories;
```

**Paso 3:** Se requiere una textura en forma rectangular

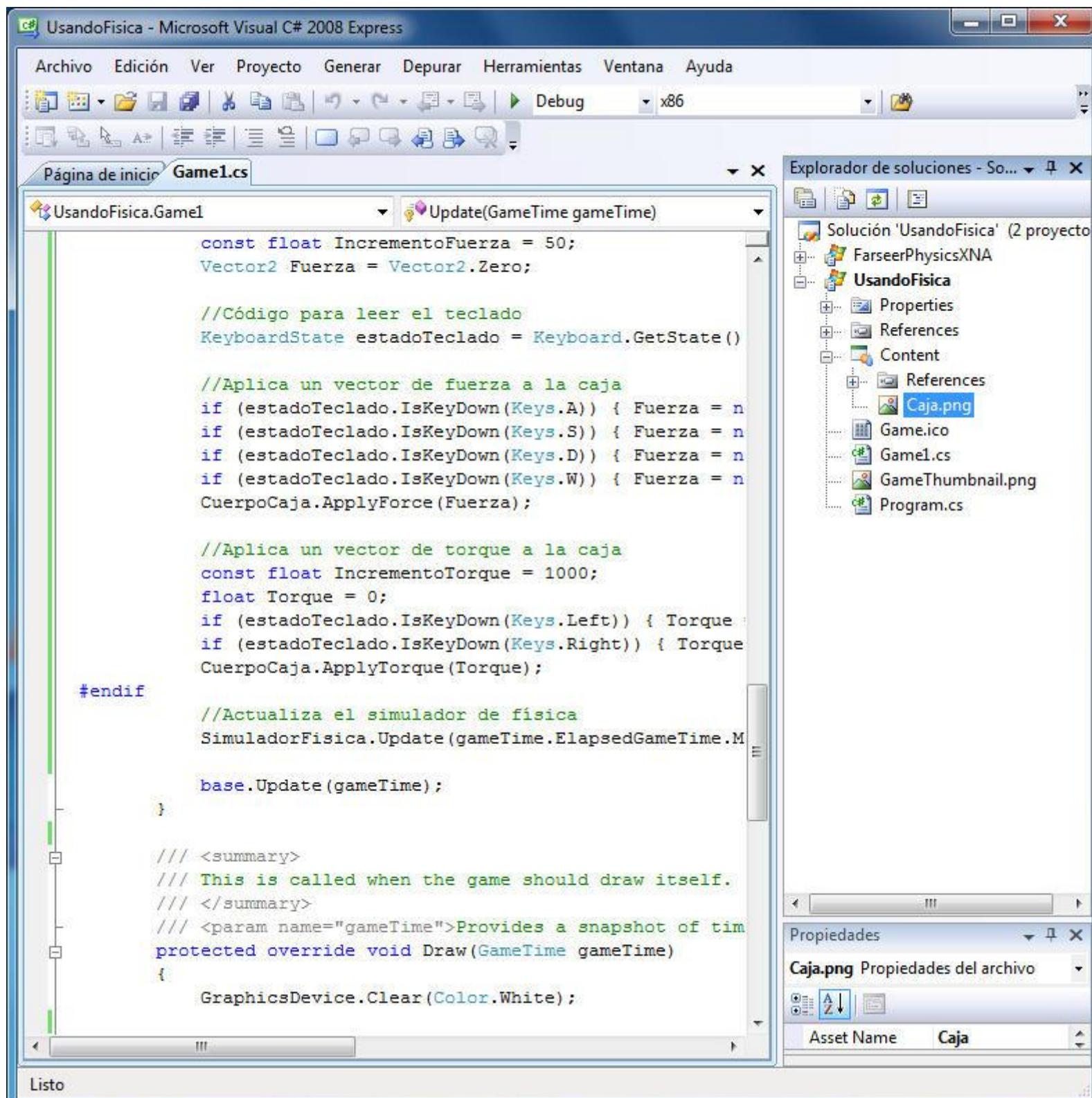


Figura 220: Haciendo uso de una textura para la caja

**Paso 4:** Las variables de clase

```
//Textura de la caja
private Texture2D TexturaCaja;

//Simulador de física que conecta los diferentes cuerpos en un solo contexto físico
PhysicsSimulator SimuladorFisica;

//Cuerpos (usados por el simulador de física)
Body CuerpoCaja;

//Geometría de los cuerpos
Geom GeometriaCaja;

//Cajas a golpear
private ObjetosJuego[] CajasGolpea;
const int MAXCAJASGOLPEA = 5;
```

**Paso 5:** Lo que debe ir en el LoadContent

Rafael Alberto Moreno Parra. <http://darwin.50webs.com>

```
//Crea el objeto SimuladorFisica y en el constructor se coloca el vector de gravedad
SimuladorFisica = new PhysicsSimulator(new Vector2(0, 0));

// TODO: use this.Content to load your game content here
TexturaCaja = Content.Load<Texture2D>("Caja");
CuerpoCaja = BodyFactory.Instance.CreateRectangleBody(SimuladorFisica, TexturaCaja.Width, TexturaCaja.Height, 1);
CuerpoCaja.Position = new Vector2(350, 100);
GeometriaCaja = GeomFactory.Instance.CreateRectangleGeom(SimuladorFisica, CuerpoCaja, TexturaCaja.Width,
TexturaCaja.Height);

//Cargar e inicializar las cajas que se van a golpear
CajasGolpea = new ObjetosJuego[MAXCAJASGOLPEA];
for (int Cont = 0; Cont < MAXCAJASGOLPEA; Cont++)
{
    CajasGolpea[Cont] = new ObjetosJuego(Content.Load<Texture2D>("Caja"));
    CajasGolpea[Cont].CuerpoCajasGolpea = BodyFactory.Instance.CreateRectangleBody(SimuladorFisica,
CajasGolpea[Cont].TexturaCajasGolpea.Width, CajasGolpea[Cont].TexturaCajasGolpea.Height, 1);
    CajasGolpea[Cont].GeometriaCajasGolpea = GeomFactory.Instance.CreateRectangleGeom(SimuladorFisica,
CajasGolpea[Cont].CuerpoCajasGolpea, CajasGolpea[Cont].TexturaCajasGolpea.Width, CajasGolpea[Cont].TexturaCajasGolpea.Height);
    CajasGolpea[Cont].CuerpoCajasGolpea.Position = new Vector2((Cont + 1) * 140, 300);
}
```

#### Paso 6: Lo que debe ir en Update

```
//Esta es una compilación condicional. El teclado y el mouse solo son leídos si la aplicación ejecuta en un PC
#ifndef !XBOX
const float IncrementoFuerza = 50;
Vector2 Fuerza = Vector2.Zero;
//Código para leer el teclado
KeyboardState estadoTeclado = Keyboard.GetState();

//Aplica un vector de fuerza a la caja
if (estadoTeclado.IsKeyDown(Keys.A)) { Fuerza = new Vector2(-IncrementoFuerza, 0); }
if (estadoTeclado.IsKeyDown(Keys.S)) { Fuerza = new Vector2(0, IncrementoFuerza); }
if (estadoTeclado.IsKeyDown(Keys.D)) { Fuerza = new Vector2(IncrementoFuerza, 0); }
if (estadoTeclado.IsKeyDown(Keys.W)) { Fuerza = new Vector2(0, -IncrementoFuerza); }
CuerpoCaja.ApplyForce(Fuerza);

//Aplica un vector de torque a la caja
const float IncrementoTorque = 1000;
float Torque = 0;
if (estadoTeclado.IsKeyDown(Keys.Left)) { Torque = -IncrementoTorque; }
if (estadoTeclado.IsKeyDown(Keys.Right)) { Torque = IncrementoTorque; }
CuerpoCaja.ApplyTorque(Torque);
#endif

//Actualiza el simulador de física
SimuladorFisica.Update(gameTime.ElapsedGameTime.Milliseconds * 0.001f);
```

#### Paso 7: Lo que debe ir en Draw

```
//Dibuja las cajas haciendo uso de la posición y rotación de los cuerpos dado por el motor de física
spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

spriteBatch.Draw(TexturaCaja, CuerpoCaja.Position, null, Color.White, CuerpoCaja.Rotation,
new Vector2(TexturaCaja.Width / 2, TexturaCaja.Height / 2), 1, SpriteEffects.None, 0);

foreach (ObjetosJuego Cajas in CajasGolpea)
    spriteBatch.Draw(Cajas.TexturaCajasGolpea, Cajas.CuerpoCajasGolpea.Position, null, Color.White,
    Cajas.CuerpoCajasGolpea.Rotation,
    new Vector2(Cajas.TexturaCajasGolpea.Width / 2, Cajas.TexturaCajasGolpea.Height / 2), 1, SpriteEffects.None, 0);

spriteBatch.End();
```

Este es el código completo de Game1.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

//Librerías del motor de física
using FarseerGames.FarseerPhysics;
using FarseerGames.FarseerPhysics.Collisions;
using FarseerGames.FarseerPhysics.Dynamics;
using FarseerGames.FarseerPhysicsFactories;

namespace GolpearCajas
{
    /// <summary>
    /// This is the main type for your game
}
```

```

/// </summary>
public class Game1 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;

    //Textura de la caja
    private Texture2D TexturaCaja;

    //Simulador de física que conecta los diferentes cuerpos en un solo contexto físico
    PhysicsSimulator SimuladorFisica;

    //Cuerpos (usados por el simulador de física)
    Body CuerpoCaja;

    //Geometría de los cuerpos
    Geom GeometriaCaja;

    //Cajas a golpear
    private ObjetosJuego[] CajasGolpea;
    const int MAXCAJASGOLPEA = 5;

    public Game1()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
    }

    /// <summary>
    /// Allows the game to perform any initialization it needs to before starting to run.
    /// This is where it can query for any required services and load any non-graphic
    /// related content. Calling base.Initialize will enumerate through any components
    /// and initialize them as well.
    /// </summary>
    protected override void Initialize()
    {
        // TODO: Add your initialization logic here

        base.Initialize();
    }

    /// <summary>
    /// LoadContent will be called once per game and is the place to load
    /// all of your content.
    /// </summary>
    protected override void LoadContent()
    {
        // Create a new SpriteBatch, which can be used to draw textures.
        spriteBatch = new SpriteBatch(GraphicsDevice);

        // TODO: use this.Content to load your game content here

        //Crea el objeto SimuladorFisica y en el constructor se coloca el vector de gravedad
        SimuladorFisica = new PhysicsSimulator(new Vector2(0, 0));

        // TODO: use this.Content to load your game content here
        TexturaCaja = Content.Load<Texture2D>("Caja");
        CuerpoCaja = BodyFactory.Instance.CreateRectangleBody(SimuladorFisica, TexturaCaja.Width, TexturaCaja.Height, 1);
        CuerpoCaja.Position = new Vector2(370, 100);
        GeometriaCaja = GeomFactory.Instance.CreateRectangleGeom(SimuladorFisica, CuerpoCaja, TexturaCaja.Width,
TexturaCaja.Height);

        //Cargar e inicializar las cajas que se van a golpear
        CajasGolpea = new ObjetosJuego[MAXCAJASGOLPEA];
        for (int Cont = 0; Cont < MAXCAJASGOLPEA; Cont++)
        {
            CajasGolpea[Cont] = new ObjetosJuego(Content.Load<Texture2D>("Caja"));
            CajasGolpea[Cont].CuerpoCajasGolpea = BodyFactory.Instance.CreateRectangleBody(SimuladorFisica,
CajasGolpea[Cont].TexturaCajasGolpea.Width, CajasGolpea[Cont].TexturaCajasGolpea.Height, 1);
            CajasGolpea[Cont].GeometriaCajasGolpea = GeomFactory.Instance.CreateRectangleGeom(SimuladorFisica,
CajasGolpea[Cont].CuerpoCajasGolpea, CajasGolpea[Cont].TexturaCajasGolpea.Width, CajasGolpea[Cont].TexturaCajasGolpea.Height);
            CajasGolpea[Cont].CuerpoCajasGolpea.Position = new Vector2((Cont + 1) * 105, 300);
        }
    }

    /// <summary>
    /// UnloadContent will be called once per game and is the place to unload
    /// all content.
    /// </summary>
    protected override void UnloadContent()
    {
        // TODO: Unload any non ContentManager content here
    }

    /// <summary>
    /// Allows the game to run logic such as updating the world,
    /// checking for collisions, gathering input, and playing audio.
    /// </summary>
    /// <param name="gameTime">Provides a snapshot of timing values.</param>
    protected override void Update(GameTime gameTime)
    {
        // Allows the game to exit
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)

```

```

        this.Exit();

        // TODO: Add your update logic here
        //Esta es una compilación condicional. El teclado y el mouse solo son leídos si la aplicación ejecuta en un PC

#if !XBOX
    const float IncrementoFuerza = 50;
    Vector2 Fuerza = Vector2.Zero;
    //Código para leer el teclado
    KeyboardState estadoTeclado = Keyboard.GetState();

    //Aplica un vector de fuerza a la caja
    if (estadoTeclado.IsKeyDown(Keys.A)) { Fuerza = new Vector2(-IncrementoFuerza, 0); }
    if (estadoTeclado.IsKeyDown(Keys.S)) { Fuerza = new Vector2(0, IncrementoFuerza); }
    if (estadoTeclado.IsKeyDown(Keys.D)) { Fuerza = new Vector2(IncrementoFuerza, 0); }
    if (estadoTeclado.IsKeyDown(Keys.W)) { Fuerza = new Vector2(0, -IncrementoFuerza); }
    CuerpoCaja.ApplyForce(Fuerza);

    //Aplica un vector de torque a la caja
    const float IncrementoTorque = 1000;
    float Torque = 0;
    if (estadoTeclado.IsKeyDown(Keys.Left)) { Torque = -IncrementoTorque; }
    if (estadoTeclado.IsKeyDown(Keys.Right)) { Torque = IncrementoTorque; }
    CuerpoCaja.ApplyTorque(Torque);
#endif

    //Actualiza el simulador de física
    SimuladorFisica.Update(gameTime.ElapsedGameTime.Milliseconds * 0.001f);

    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.White);

    // TODO: Add your drawing code here

    //Dibuja las cajas haciendo uso de la posición y rotación de los cuerpos dado por el motor de física
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

    spriteBatch.Draw(TexturaCaja, CuerpoCaja.Position, null, Color.White, CuerpoCaja.Rotation,
        new Vector2(TexturaCaja.Width / 2, TexturaCaja.Height / 2), 1, SpriteEffects.None, 0);

    foreach (ObjetosJuego Cajas in CajasGolpea)
        spriteBatch.Draw(Cajas.TexturaCajasGolpea, Cajas.CuerpoCajasGolpea.Position, null, Color.White,
Cajas.CuerpoCajasGolpea.Rotation,
            new Vector2(Cajas.TexturaCajasGolpea.Width / 2, Cajas.TexturaCajasGolpea.Height / 2), 1, SpriteEffects.None,
0);

    spriteBatch.End();

    base.Draw(gameTime);
}
}

```

No olvidemos poner la clase ObjetosJuego.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

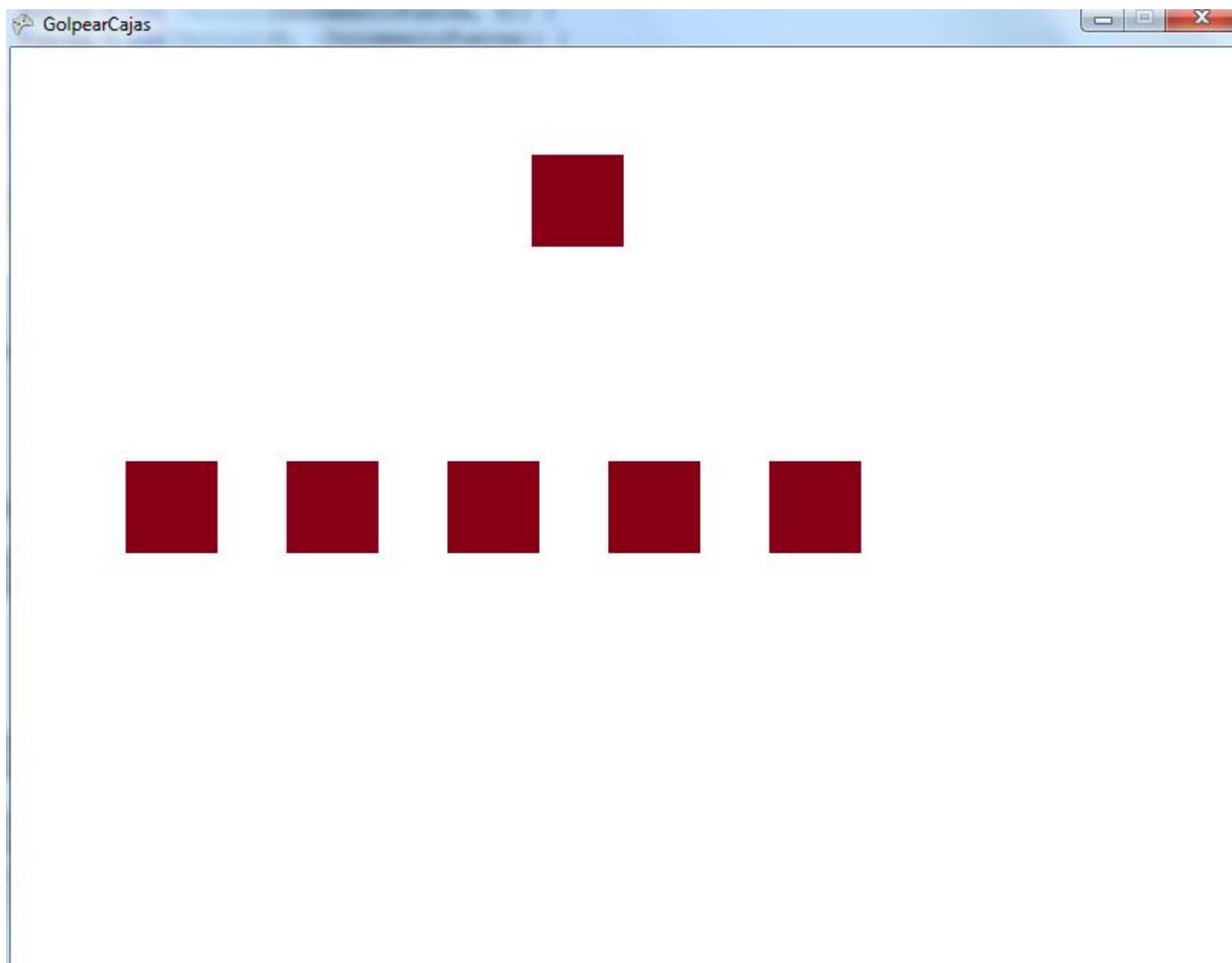
//Librerías del motor de física
using FarseerGames.FarseerPhysics;
using FarseerGames.FarseerPhysics.Collisions;
using FarseerGames.FarseerPhysics.Dynamics;
using FarseerGames.FarseerPhysicsFactories;

namespace GolpearCajas
{
    class ObjetosJuego
    {
        public Texture2D TexturaCajasGolpea; //La textura (o imagen)
        public Body CuerpoCajasGolpea;
        public Geom GeometriaCajasGolpea;

        //Constructor
        public ObjetosJuego(Texture2D textura)
        {
            TexturaCajasGolpea = textura; //Recibe la textura por parámetro
        }
    }
}

```

Este es el resultado:



**Figura 221: Animación de como una caja colisiona con otras**

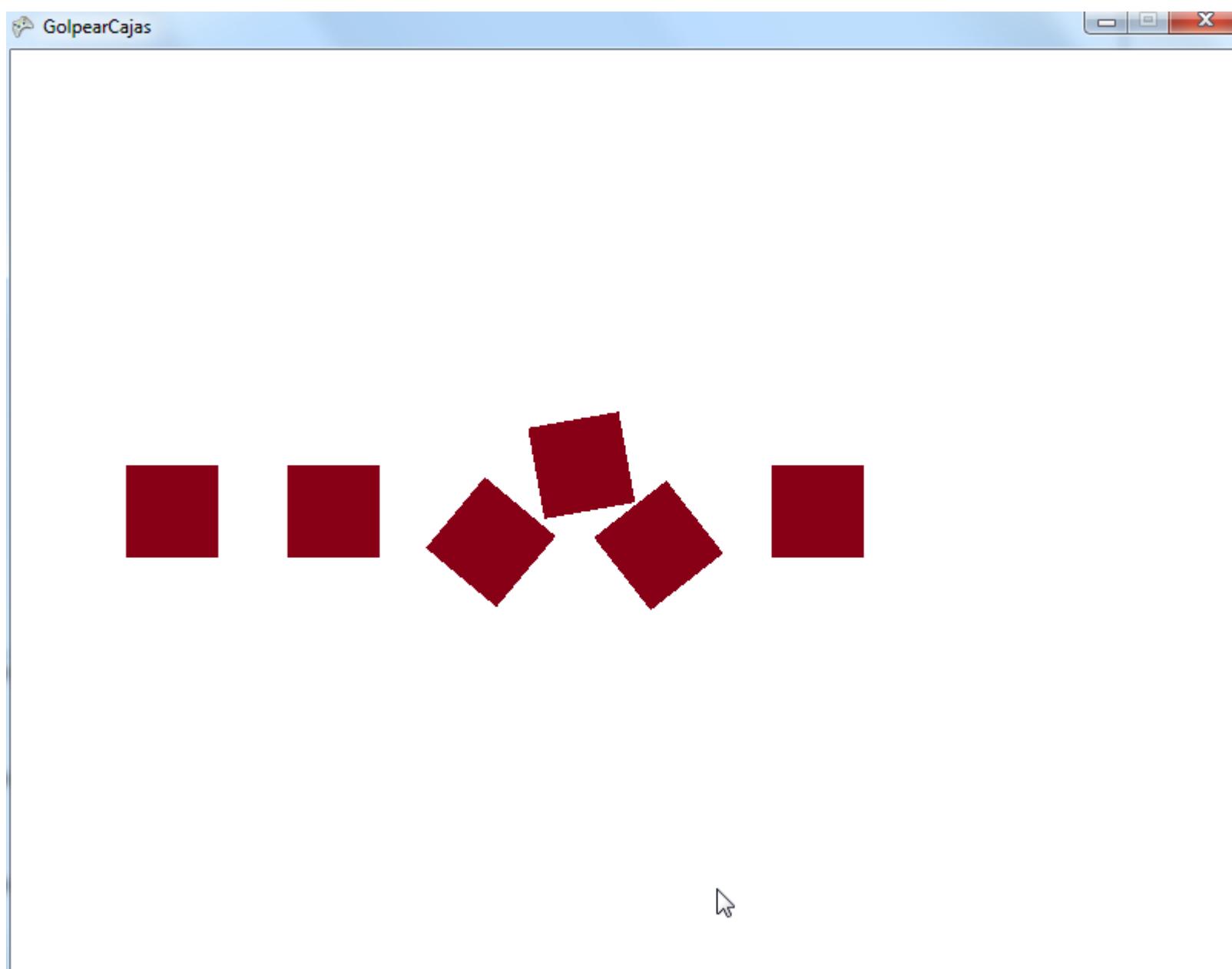


Figura 222: Animación de como una caja colisiona con otras

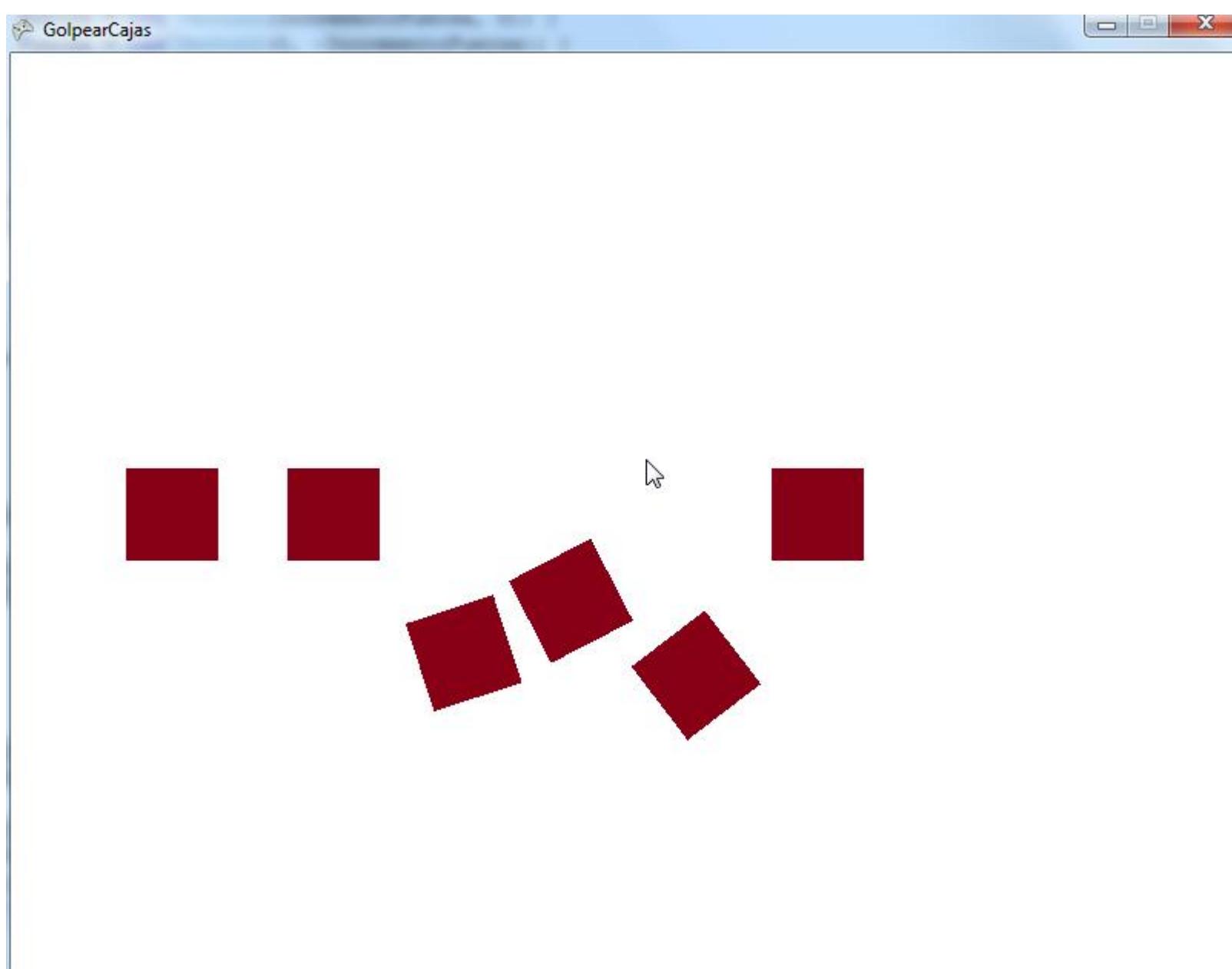


Figura 223: Animación de como una caja colisiona con otras

## 38. XNA: Usando un Motor de Física. Colisión entre objeto rectangular y circular.

En esta lección trataremos de colisiones entre un objeto rectangular y un objeto circular.

**Paso 1:** Se genera un nuevo proyecto y se le hace referencia al motor de física como se vio en el capítulo anterior.

**Paso 2:** Las librerías que deben usarse en el programa

```
//Librerías del motor de física
using FarseerGames.FarseerPhysics;
using FarseerGames.FarseerPhysics.Collisions;
using FarseerGames.FarseerPhysics.Dynamics;
using FarseerGames.FarseerPhysics.Factories;
```

**Paso 3:** Se requiere una textura en forma rectangular

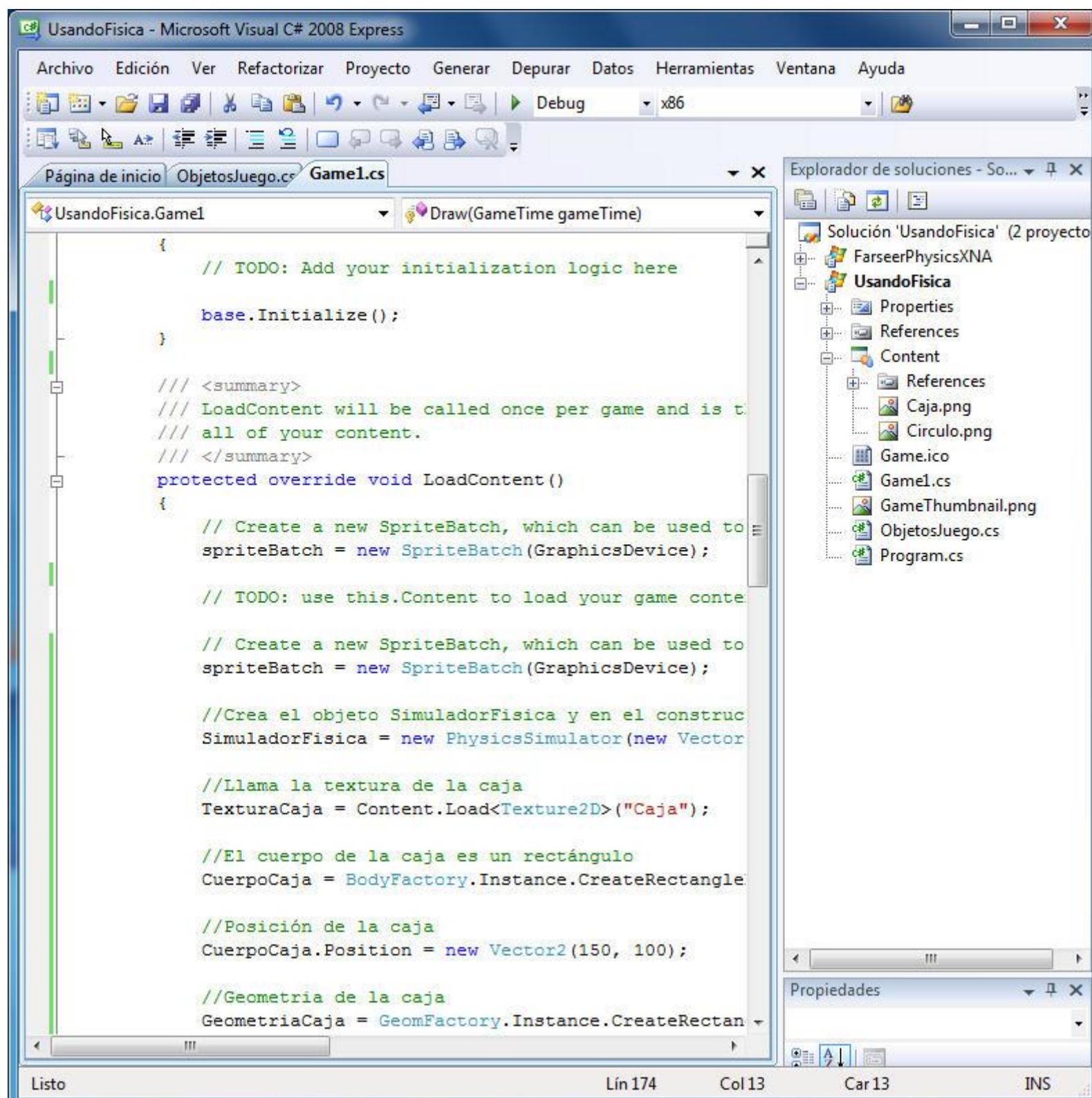


Figura 224: Dos texturas para representar el círculo y la caja

**Paso 4:** Las variables de clase

```
//Textura caja
private Texture2D TexturaCaja;

//Textura del círculo
private Texture2D TexturaCirculo;

//Simulador de física que conecta los diferentes cuerpos en un solo contexto físico
PhysicsSimulator SimuladorFisica;

//Cuerpos (usados por el simulador de física)
Body CuerpoCaja;
Body CuerpoCirculo;

//Geometría de los cuerpos
Geom GeometriaCaja;
Geom GeometriaCirculo;
```

**Paso 5:** Lo que debe ir en el LoadContent

```
//Crea el objeto SimuladorFisica y en el constructor se coloca el vector de gravedad
SimuladorFisica = new PhysicsSimulator(new Vector2(0, 0));

//Llama la textura de la caja
TexturaCaja = Content.Load<Texture2D>("Caja");

//El cuerpo de la caja es un rectángulo
CuerpoCaja = BodyFactory.Instance.CreateRectangleBody(SimuladorFisica, TexturaCaja.Width, TexturaCaja.Height, 1);

//Posición de la caja
CuerpoCaja.Position = new Vector2(150, 100);

//Geometria de la caja
GeometriaCaja = GeomFactory.Instance.CreateRectangleGeom(SimuladorFisica, CuerpoCaja, TexturaCaja.Width,
TexturaCaja.Height);

//Llama la textura del círculo
TexturaCirculo = Content.Load<Texture2D>("Circulo");

//El cuerpo del círculo
CuerpoCirculo = BodyFactory.Instance.CreateCircleBody(SimuladorFisica, TexturaCirculo.Width / 2, 1);

//Posición del círculo
CuerpoCirculo.Position = new Vector2(450, 100);

//Geometría del círculo. Se ponen 12 ejes.
GeometriaCirculo = GeomFactory.Instance.CreateCircleGeom(SimuladorFisica, CuerpoCirculo, TexturaCirculo.Width / 2, 12);
```

**Paso 6:** Lo que debe ir en Update

```
//Esta es una compilación condicional. El teclado y el mouse solo son leídos si la aplicación ejecuta en un PC
#ifndef !XBOX
const float IncrementoFuerza = 50;
Vector2 Fuerza = Vector2.Zero;

//Código para leer el teclado
KeyboardState estadoTeclado = Keyboard.GetState();

//Aplica un vector de fuerza a la caja
if (estadoTeclado.IsKeyDown(Keys.A)) { Fuerza = new Vector2(-IncrementoFuerza, 0); }
if (estadoTeclado.IsKeyDown(Keys.S)) { Fuerza = new Vector2(0, IncrementoFuerza); }
if (estadoTeclado.IsKeyDown(Keys.D)) { Fuerza = new Vector2(IncrementoFuerza, 0); }
if (estadoTeclado.IsKeyDown(Keys.W)) { Fuerza = new Vector2(0, -IncrementoFuerza); }
CuerpoCaja.ApplyForce(Fuerza);

//Aplica un vector de torque a la caja
const float IncrementoTorque = 1000;
float Torque = 0;
if (estadoTeclado.IsKeyDown(Keys.Left)) { Torque = -IncrementoTorque; }
if (estadoTeclado.IsKeyDown(Keys.Right)) { Torque = IncrementoTorque; }
CuerpoCaja.ApplyTorque(Torque);
#endif

//Actualiza el simulador de física
SimuladorFisica.Update(gameTime.ElapsedGameTime.Milliseconds * 0.001f);
```

**Paso 7:** Lo que debe ir en Draw

```
//Dibuja la caja y el círculo haciendo uso de la posición y rotación de los cuerpos dado por el motor de física
spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

spriteBatch.Draw(TexturaCaja, CuerpoCaja.Position, null, Color.White, CuerpoCaja.Rotation,
new Vector2(TexturaCaja.Width / 2, TexturaCaja.Height / 2), 1, SpriteEffects.None, 0);

spriteBatch.Draw(TexturaCirculo, CuerpoCirculo.Position, null, Color.White, CuerpoCirculo.Rotation,
new Vector2(TexturaCirculo.Width / 2, TexturaCirculo.Height / 2), 1, SpriteEffects.None, 0);

spriteBatch.End();
```

Este es el código completo de Game1.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

//Librerías del motor de física
using FarseerGames.FarseerPhysics;
```

```

using FarseerGames.FarseerPhysics.Collisions;
using FarseerGames.FarseerPhysics.Dynamics;
using FarseerGames.FarseerPhysics.Factories;

namespace UsandoFisica
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        //Textura caja
        private Texture2D TexturaCaja;

        //Textura del círculo
        private Texture2D TexturaCirculo;

        //Simulador de física que conecta los diferentes cuerpos en un solo contexto físico
        PhysicsSimulator SimuladorFisica;

        //Cuerpos (usados por el simulador de física)
        Body CuerpoCaja;
        Body CuerpoCirculo;

        //Geometría de los cuerpos
        Geom GeometriaCaja;
        Geom GeometriaCirculo;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content. Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // TODO: use this.Content to load your game content here

            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            //Crea el objeto SimuladorFisica y en el constructor se coloca el vector de gravedad
            SimuladorFisica = new PhysicsSimulator(new Vector2(0, 0));

            //Llama la textura de la caja
            TexturaCaja = Content.Load<Texture2D>("Caja");

            //El cuerpo de la caja es un rectángulo
            CuerpoCaja = BodyFactory.Instance.CreateRectangleBody(SimuladorFisica, TexturaCaja.Width, TexturaCaja.Height, 1);

            //Posición de la caja
            CuerpoCaja.Position = new Vector2(150, 100);

            //Geometria de la caja
            GeometriaCaja = GeomFactory.Instance.CreateRectangleGeom(SimuladorFisica, CuerpoCaja, TexturaCaja.Width, TexturaCaja.Height);

            //Llama la textura del círculo
            TexturaCirculo = Content.Load<Texture2D>("Circulo");

            //El cuerpo del círculo
            CuerpoCirculo = BodyFactory.Instance.CreateCircleBody(SimuladorFisica, TexturaCirculo.Width / 2, 1);

            //Posición del círculo
            CuerpoCirculo.Position = new Vector2(450, 100);

            //Geometría del círculo. Se ponen 12 ejes.
            GeometriaCirculo = GeomFactory.Instance.CreateCircleGeom(SimuladorFisica, CuerpoCirculo, TexturaCirculo.Width / 2, 12);
        }
    }
}

```

```

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();

    // TODO: Add your update logic here

    //Esta es una compilación condicional. El teclado y el mouse solo son leídos si la aplicación ejecuta en un PC
#if !XBOX
    const float IncrementoFuerza = 50;
    Vector2 Fuerza = Vector2.Zero;

    //Código para leer el teclado
    KeyboardState estadoTeclado = Keyboard.GetState();

    //Aplica un vector de fuerza a la caja
    if (estadoTeclado.IsKeyDown(Keys.A)) { Fuerza = new Vector2(-IncrementoFuerza, 0); }
    if (estadoTeclado.IsKeyDown(Keys.S)) { Fuerza = new Vector2(0, IncrementoFuerza); }
    if (estadoTeclado.IsKeyDown(Keys.D)) { Fuerza = new Vector2(IncrementoFuerza, 0); }
    if (estadoTeclado.IsKeyDown(Keys.W)) { Fuerza = new Vector2(0, -IncrementoFuerza); }
    CuerpoCaja.ApplyForce(Fuerza);

    //Aplica un vector de torque a la caja
    const float IncrementoTorque = 1000;
    float Torque = 0;
    if (estadoTeclado.IsKeyDown(Keys.Left)) { Torque = -IncrementoTorque; }
    if (estadoTeclado.IsKeyDown(Keys.Right)) { Torque = IncrementoTorque; }
    CuerpoCaja.ApplyTorque(Torque);
#endif

    //Actualiza el simulador de física
    SimuladorFisica.Update(gameTime.ElapsedGameTime.Milliseconds * 0.001f);

    base.Update(gameTime);
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.White);

    // TODO: Add your drawing code here
    //Dibuja la caja y el círculo haciendo uso de la posición y rotación de los cuerpos dado por el motor de física
    spriteBatch.Begin(SpriteBlendMode.AlphaBlend);

    spriteBatch.Draw(TexturaCaja, CuerpoCaja.Position, null, Color.White, CuerpoCaja.Rotation,
        new Vector2(TexturaCaja.Width / 2, TexturaCaja.Height / 2), 1, SpriteEffects.None, 0);

    spriteBatch.Draw(TexturaCirculo, CuerpoCirculo.Position, null, Color.White, CuerpoCirculo.Rotation,
        new Vector2(TexturaCirculo.Width / 2, TexturaCirculo.Height / 2), 1, SpriteEffects.None, 0);

    spriteBatch.End();

    base.Draw(gameTime);
}
}

```

Este es el resultado

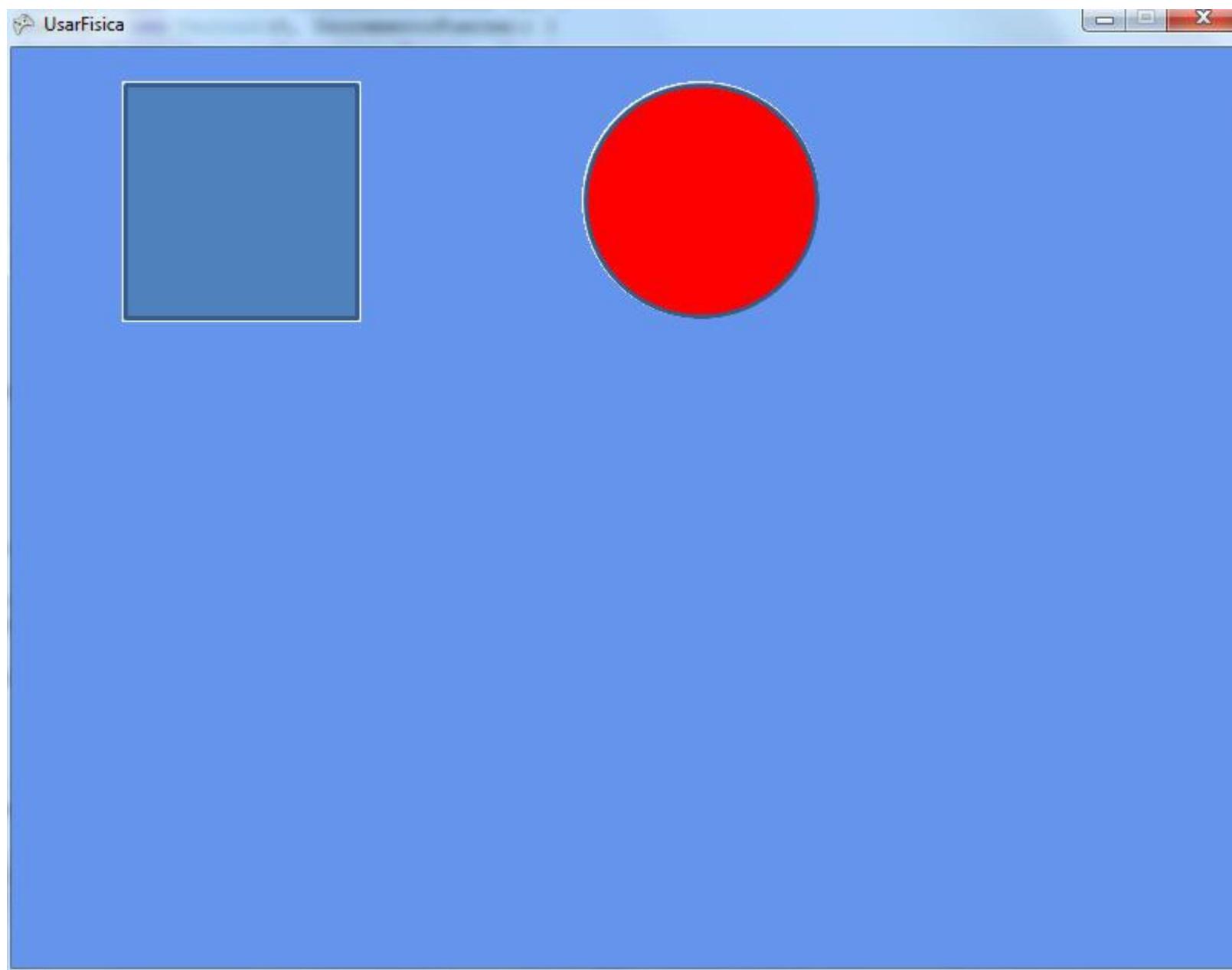


Figura 225: Ejemplo de colisión entre un círculo y una caja

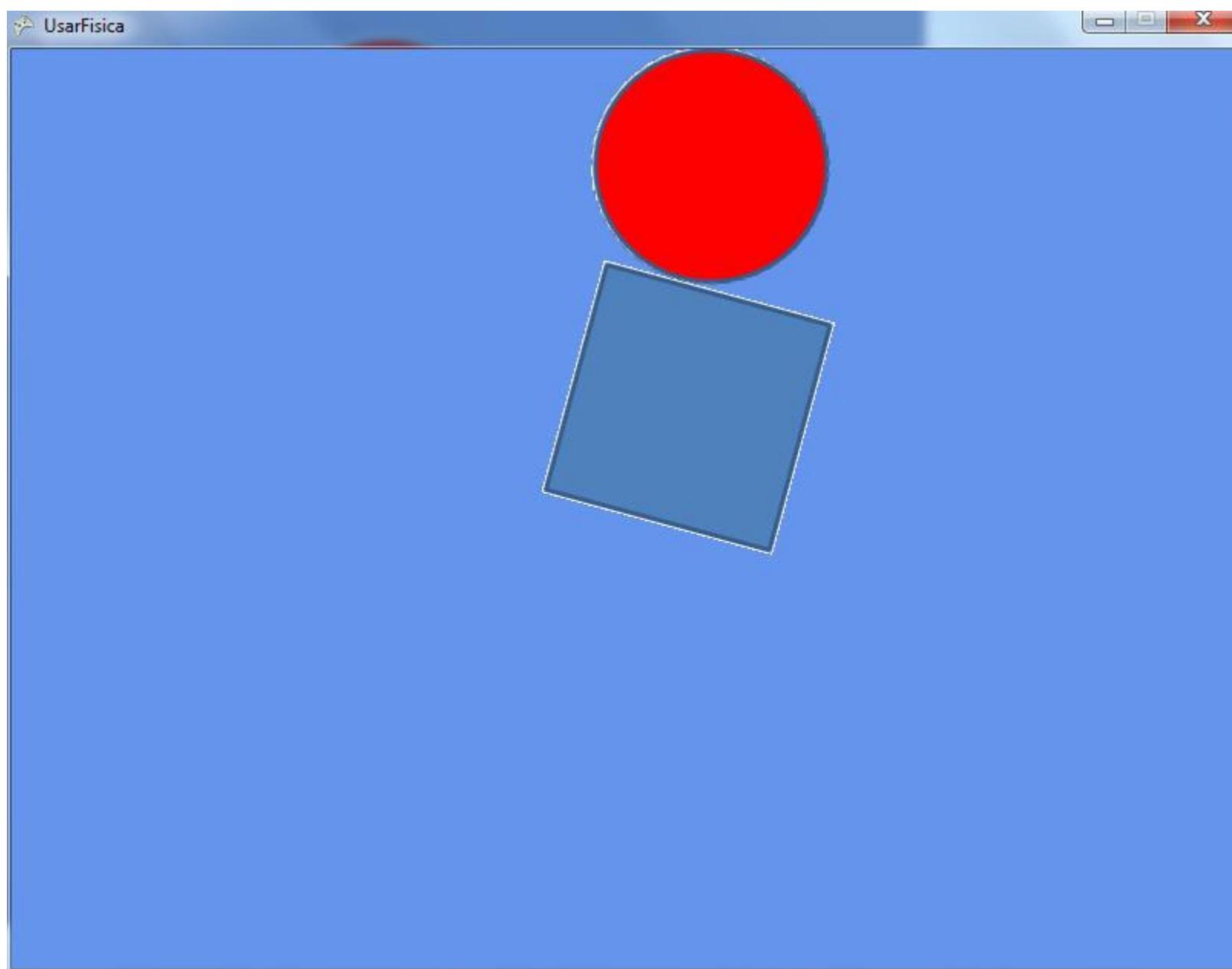


Figura 226: Ejemplo de colisión entre un círculo y una caja

## Bibliografía

- Brackeen David, Barker Bret and Vanhelstuwé Laurence** Developing Games in Java [Book]. - [s.l.] : New Riders Publishing, 2003.
- Cawood Stephen and McGee Pat** Microsoft XNA Game Studio Creator's Guide [Book]. - [s.l.] : McGrawHill, 2007.
- Davison Andrew** Killer Game Programming in Java [Book]. - [s.l.] : O'Reilly, 2005.
- Harris Andy** Microsoft C# Programming for the absolute beginner [Book]. - [s.l.] : Course Technology PTR, 2002.
- Klawonn Frank** Introduction to Computer Graphics. Using Java 2D y 3D [Book]. - London : Springer, 2008.
- Reed Aaron** Learning XNA 3.0 [Book]. - [s.l.] : O'REILLY, 2008.