26-1-2022

Gráficos y animación en C#.

Enero de 2022

Rafael Alberto Moreno Parra

Otros libros del autor

Libro 19: "Evaluador de expresiones en nueve lenguajes de programación. En C#, C++, Delphi, Java, JavaScript, PHP, Python, TypeScript y Visual Basic .NET". En Colombia 2021. Págs. 158. Libro y código fuente descargable en: https://github.com/ramsoftware/Evaluador3

Libro 18: "C#: Árboles binarios, n-arios, grafos, listas simple y doblemente enlazadas". En Colombia 2021. Págs. 63. Libro y código fuente descargable en: https://github.com/ramsoftware/C-Sharp-Arboles

Libro 17: "C#. Estructuras dinámicas de memoria". En Colombia 2021. Págs. 82. Libro y código fuente descargable en: https://github.com/ramsoftware/CSharpDinamica

Libro 16: "C#. Programación Orientada a Objetos". En Colombia 2020. Págs. 90. Libro y código fuente descargable en: https://github.com/ramsoftware/C-Sharp-POO

Libro 15: "C#. Estructuras básicas de memoria.". En Colombia 2020. Págs. 60. Libro y código fuente descargable en: https://github.com/ramsoftware/EstructuraBasicaMemoriaCSharp

Libro 14: "Iniciando en C#". En Colombia 2020. Págs. 72. Libro y código fuente descargable en: https://github.com/ramsoftware/C-Sharp-Iniciando

Libro 13: "Algoritmos Genéticos". En Colombia 2020. Págs. 62. Libro y código fuente descargable en: https://github.com/ramsoftware/LibroAlgoritmoGenetico2020

Libro 12: "Redes Neuronales. Segunda Edición". En Colombia 2020. Págs. 108. Libro y código fuente descargable en: https://github.com/ramsoftware/LibroRedNeuronal2020

Libro 11: "Capacitándose en JavaScript". En Colombia 2020. Págs. 317. Libro y código fuente descargable en: https://github.com/ramsoftware/JavaScript

Libro 10: "Desarrollo de aplicaciones para Android usando MIT App Inventor 2". En Colombia 2016. Págs. 102. Ubicado en: https://openlibra.com/es/book/desarrollo-de-aplicaciones-para-android-usando-mit-app-inventor-2

Libro 9: "Redes Neuronales. Parte 1.". En Colombia 2016. Págs. 90. Libro descargable en: https://openlibra.com/es/book/redes-neuronales-parte-1

Libro 8: "Segunda parte de uso de algoritmos genéticos para la búsqueda de patrones". En Colombia 2015. Págs. 303. En publicación por la Universidad Libre – Cali.

Libro 7: "Desarrollo de un evaluador de expresiones algebraicas. **Versión 2.0**. C++, C#, Visual Basic .NET, Java, PHP, JavaScript y Object Pascal (Delphi)". En: Colombia 2013. Págs. 308. Ubicado en: https://openlibra.com/es/book/evaluador-de-expresiones-algebraicas-ii

Libro 6: "Un uso de algoritmos genéticos para la búsqueda de patrones". En Colombia 2013. En publicación por la Universidad Libre – Cali.

Libro 5: Desarrollo fácil y paso a paso de aplicaciones para Android usando MIT App Inventor. En Colombia 2013. Págs. 104. Estado: Obsoleto (No hay enlace).

Libro 4: "Desarrollo de un evaluador de expresiones algebraicas. C++, C#, Visual Basic .NET, Java, PHP, JavaScript y Object Pascal (Delphi)". En Colombia 2012. Págs. 308. Ubicado en: https://openlibra.com/es/book/evaluador-de-expresiones-algebraicas

Libro 3: "Simulación: Conceptos y Programación" En Colombia 2012. Págs. 81. Ubicado en: https://openlibra.com/es/book/simulacion-conceptos-y-programacion

Libro 2: "Desarrollo de videojuegos en 2D con Java y Microsoft XNA". En Colombia 2011. Págs. 260. Ubicado en: https://openlibra.com/es/book/desarrollo-de-juegos-en-2d-usando-java-y-microsoft-xna . ISBN: 978-958-8630-45-8

Libro 1: "Desarrollo de gráficos para PC, Web y dispositivos móviles" En Colombia 2009. ed.: Artes Gráficas Del Valle Editores Impresores Ltda. ISBN: 978-958-8308-95-1 v. 1 págs. 317

Artículo: "Programación Genética: La regresión simbólica". Entramado ISSN: 1900-3803 ed.: Universidad Libre Seccional Cali v.3 fasc.1 p.76 - 85, 2007

Página web del autor y canal en YouTube

Investigación sobre Vida Artificial: http://darwin.50webs.com

Canal en YouTube: http://www.youtube.com/user/RafaelMorenoP (dedicado principalmente al desarrollo en C#)

Sitio en GitHub

El código fuente se puede descargar en https://github.com/ramsoftware/

Licencia del software

Todo el software desarrollado aquí tiene licencia LGPL "Lesser General Public License" [1]

Marcas registradas

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

Microsoft ® Windows ® Enlace: http://windows.microsoft.com/en-US/windows/home [2]

Microsoft ® Visual Studio 2022 ® Enlace: https://visualstudio.microsoft.com/es/vs/ [3]

Dedicatoria

A mis padres, a mi hermana….

Y a mi tropa gatuna: Sally, Suini, Vikingo, Grisú, Capuchina, Milú y mis recordadas Tinita, Tammy y Michu.

Contenido

Introducción

En este libro se explica paso a paso como hacer gráficos en C# [4] desde las primitivas gráficas [5] (líneas, círculos, rectángulos, etc..) hasta videojuegos y simulaciones haciendo uso de animaciones.

Qué debemos tener instalado

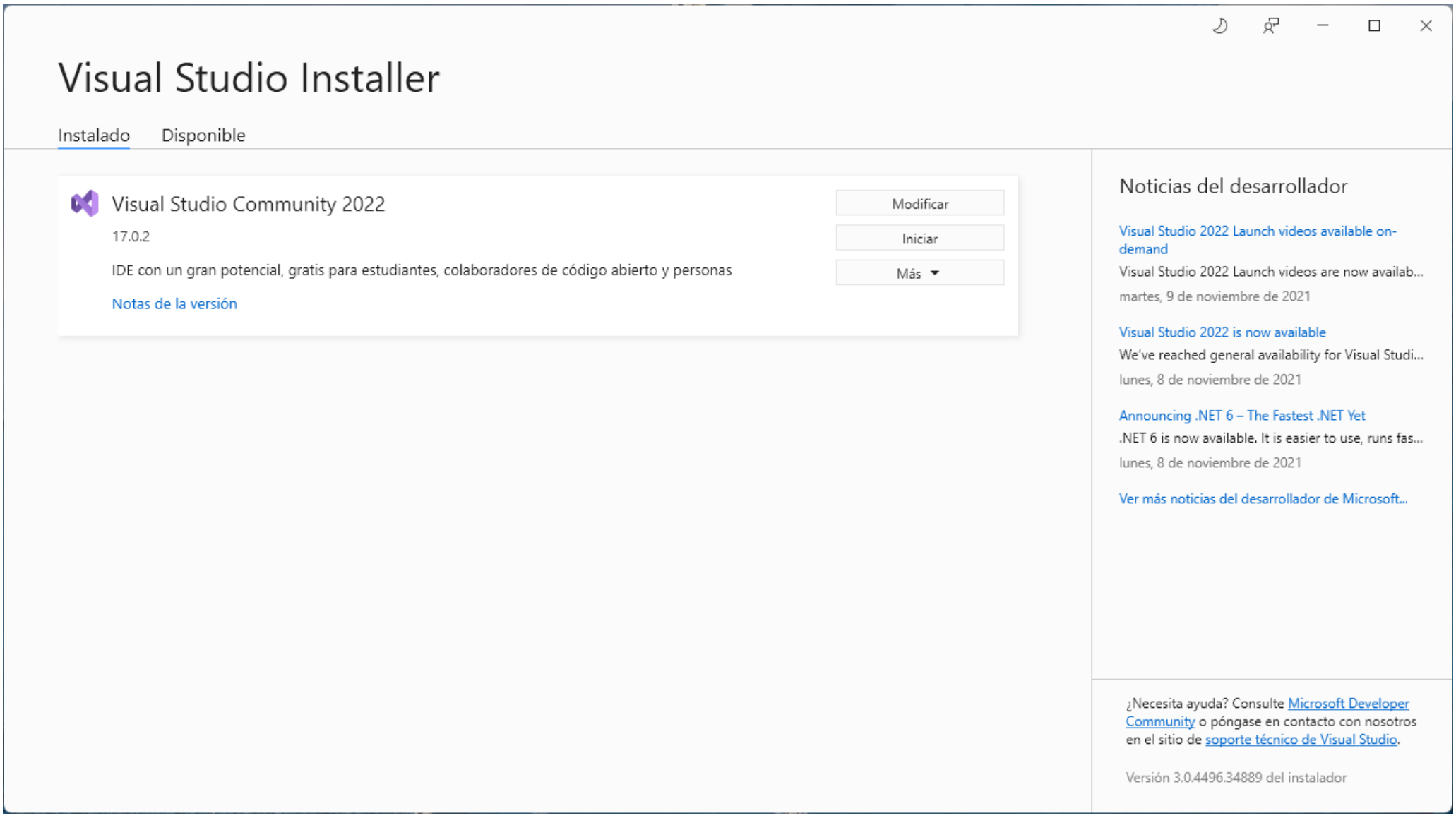


Ilustración 1: En el instalador de Visual Studio Community 2022

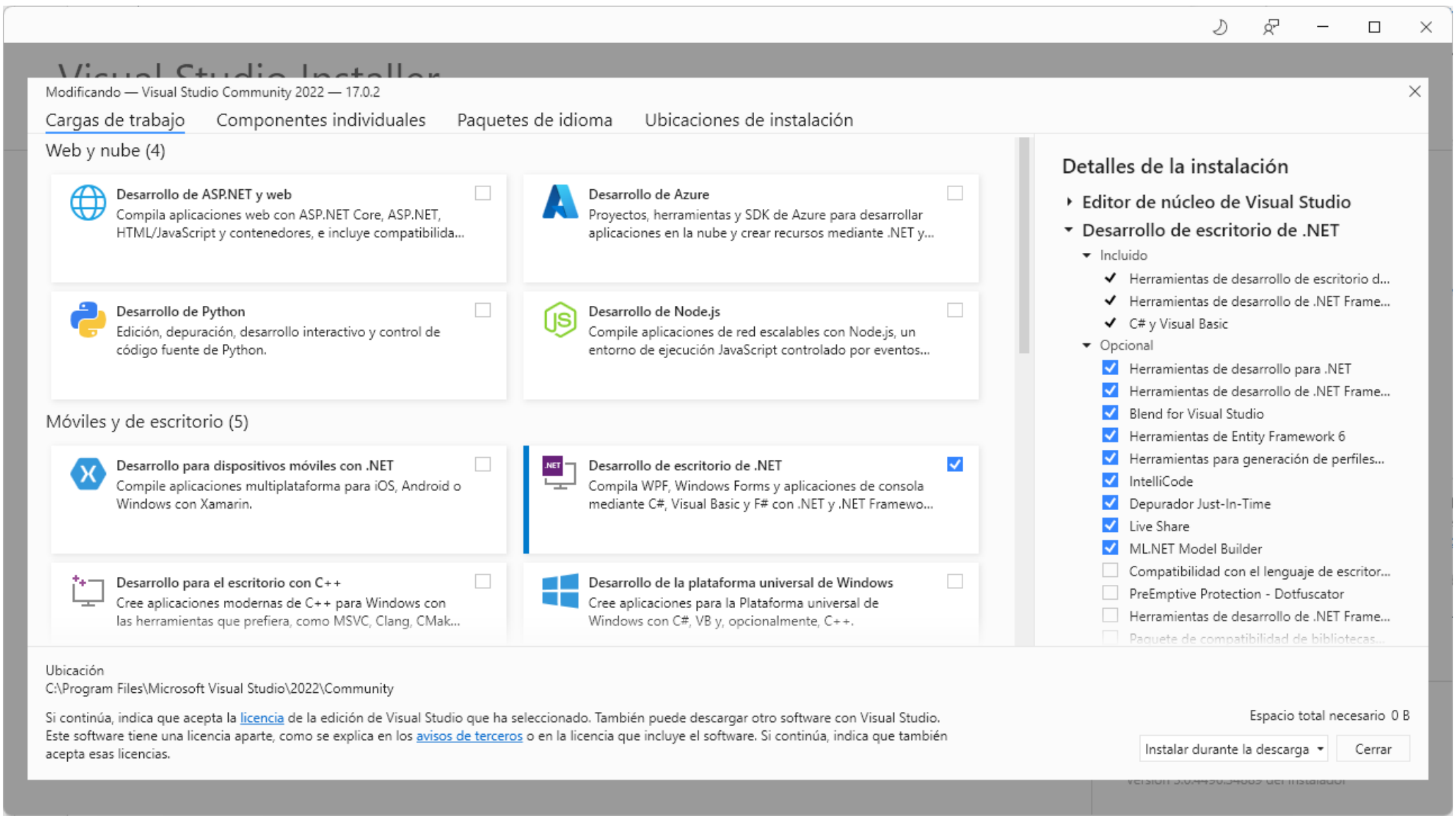


Ilustración 2: Tener marcado "Desarrollo de escritorio de .NET"

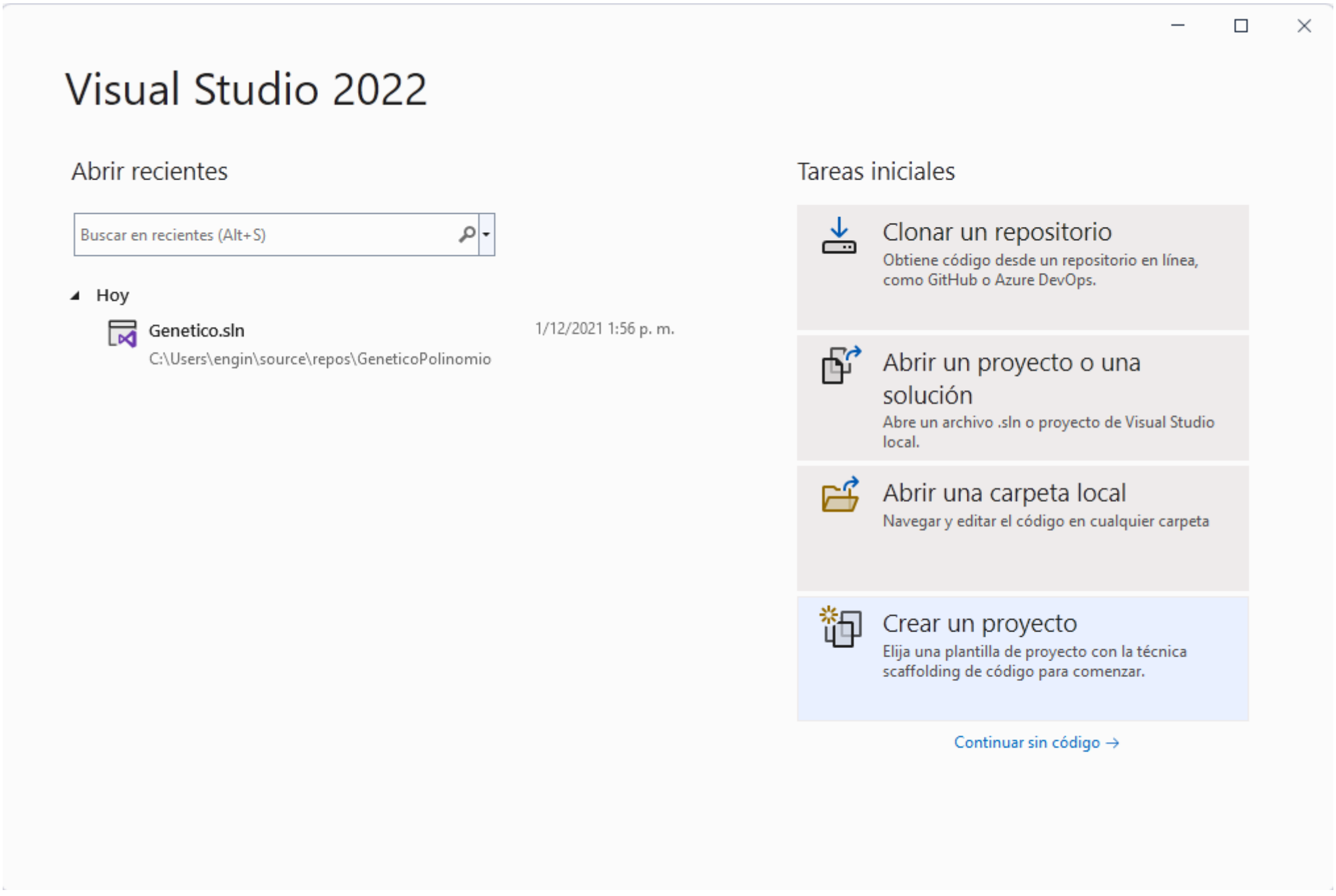


Ilustración 3: Inicia en "Crear un proyecto"

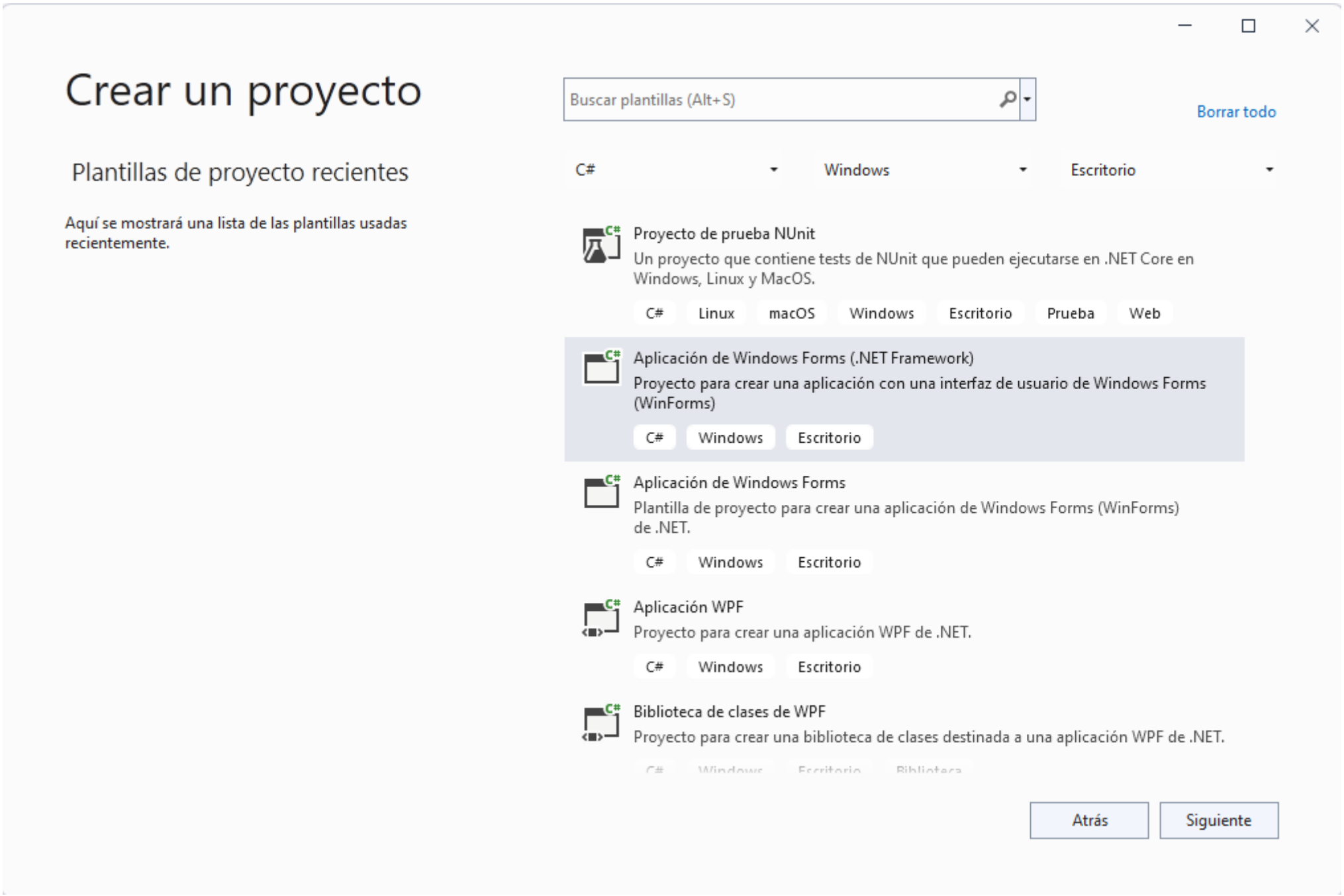


Ilustración 4: Seleccionar "Aplicación de Windows Forms (.NET Framework)"

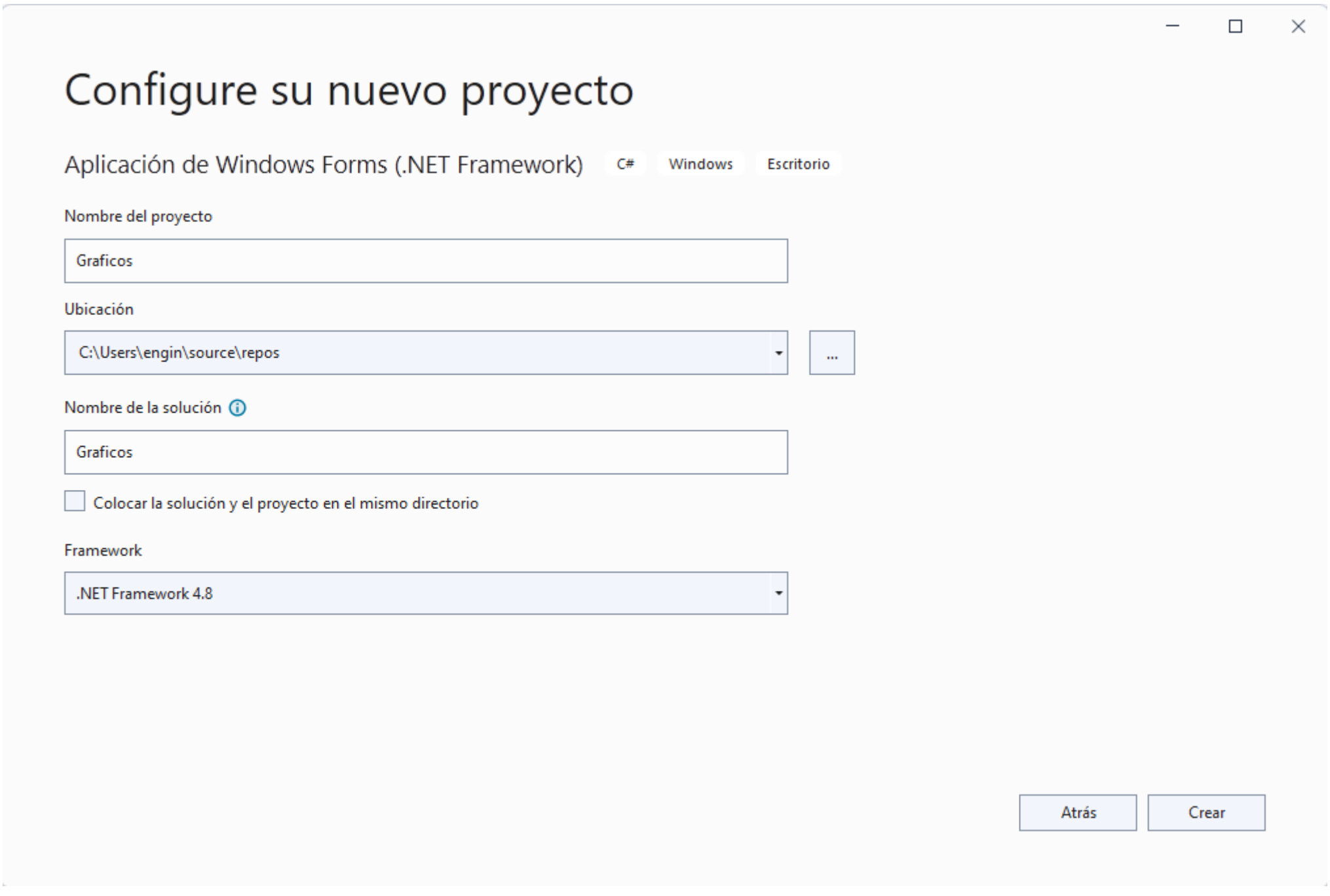


Ilustración 5: Se pone nombre al proyecto y se recomienda seleccionar .NET Framework 4.8

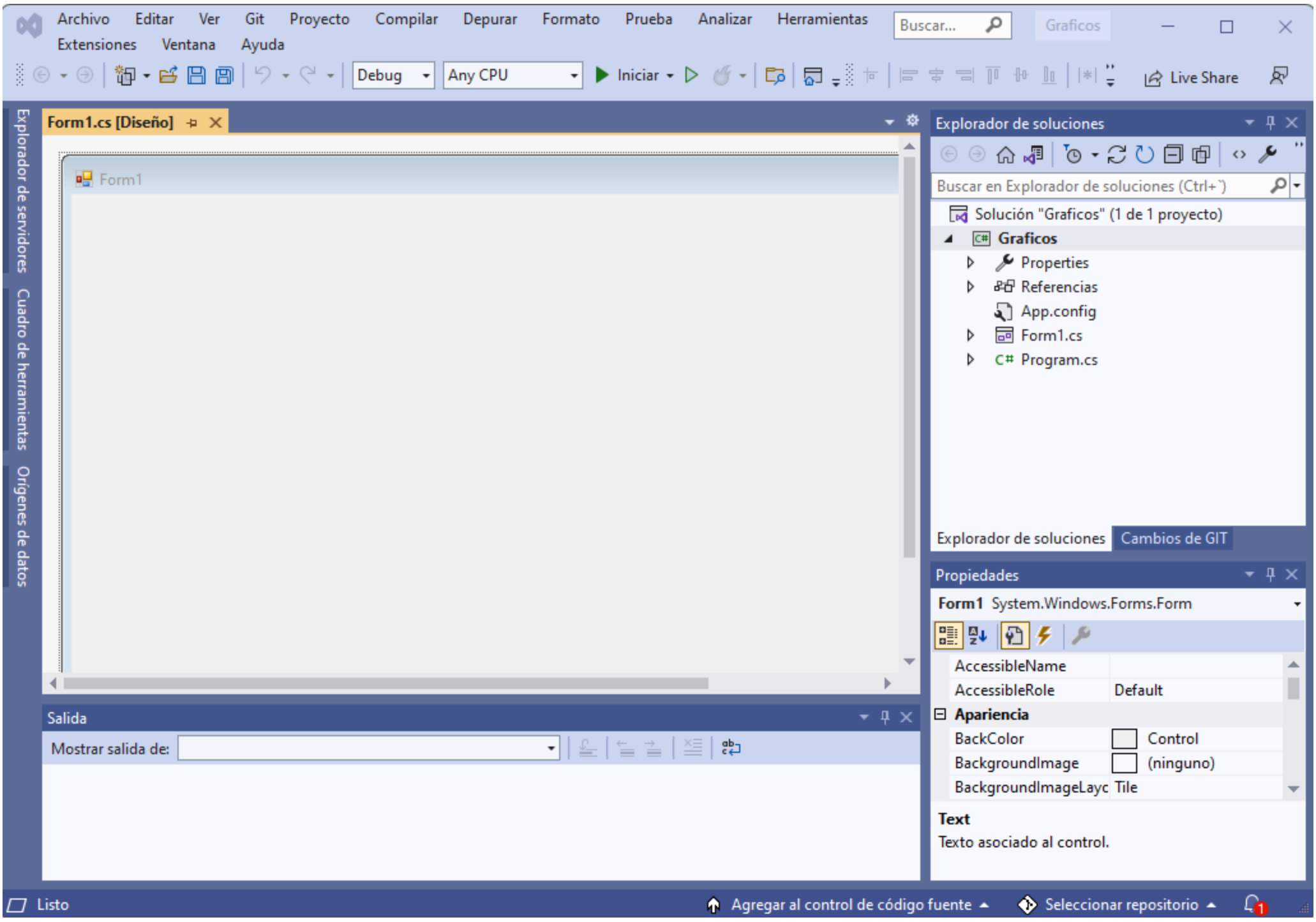


Ilustración 6: Inicio de un proyecto de escritorio

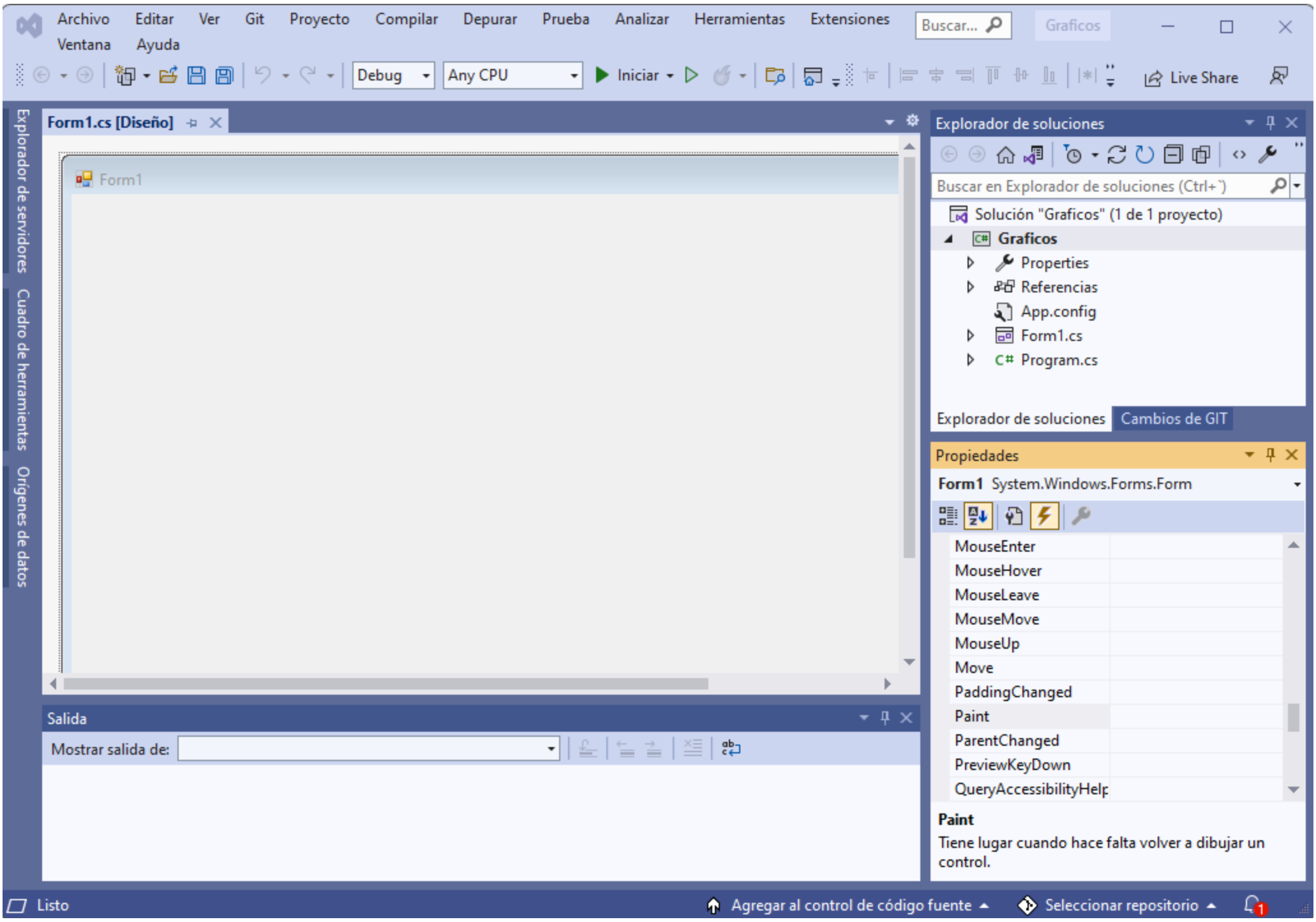


Ilustración 7: Se debe activar el evento "Paint"

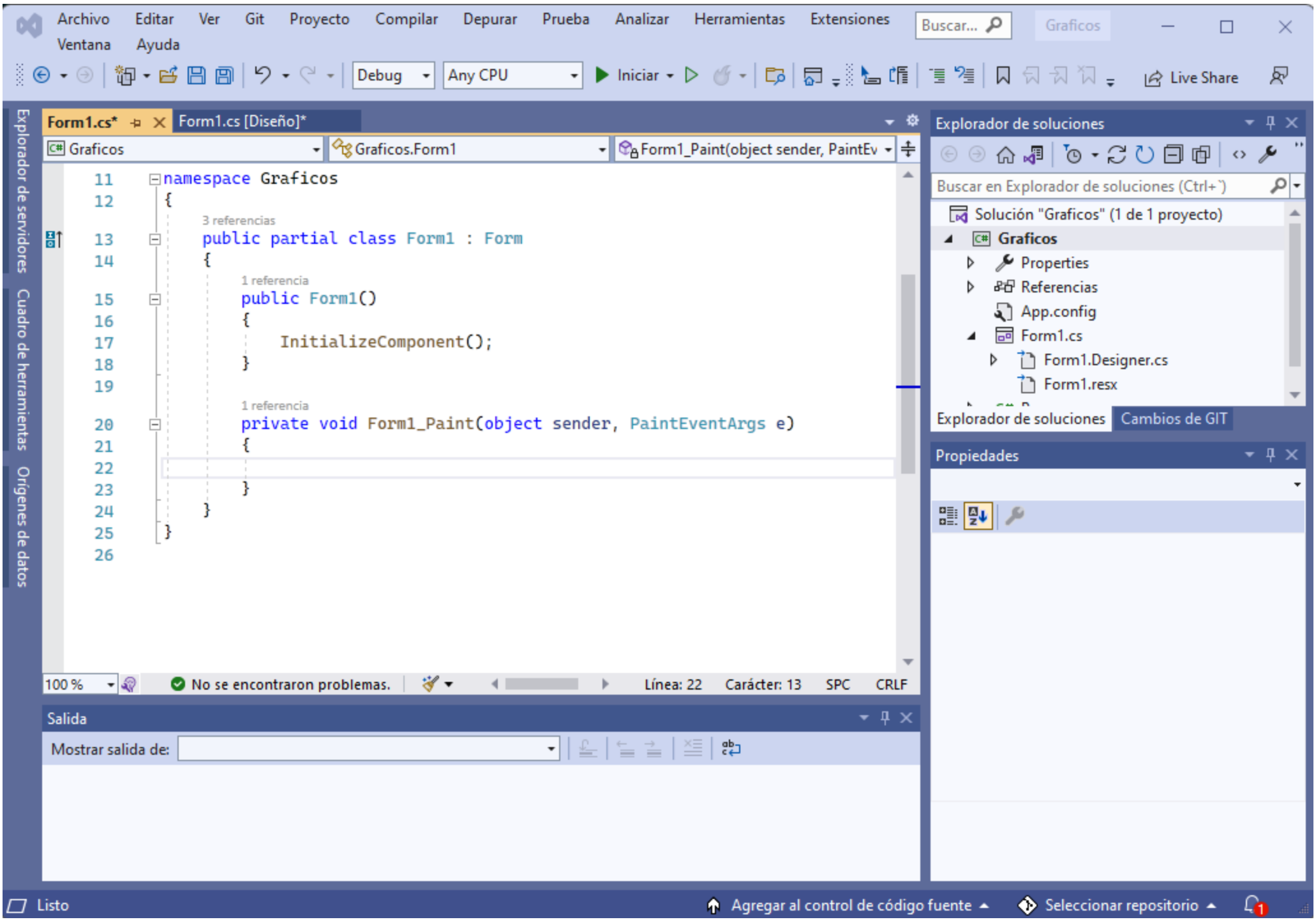


Ilustración 8: El evento "Paint" activado. Allí se escribirá el código de las gráficas.

Primitivas gráficas

Haciendo una línea

El código dentro del evento Paint es el siguiente:

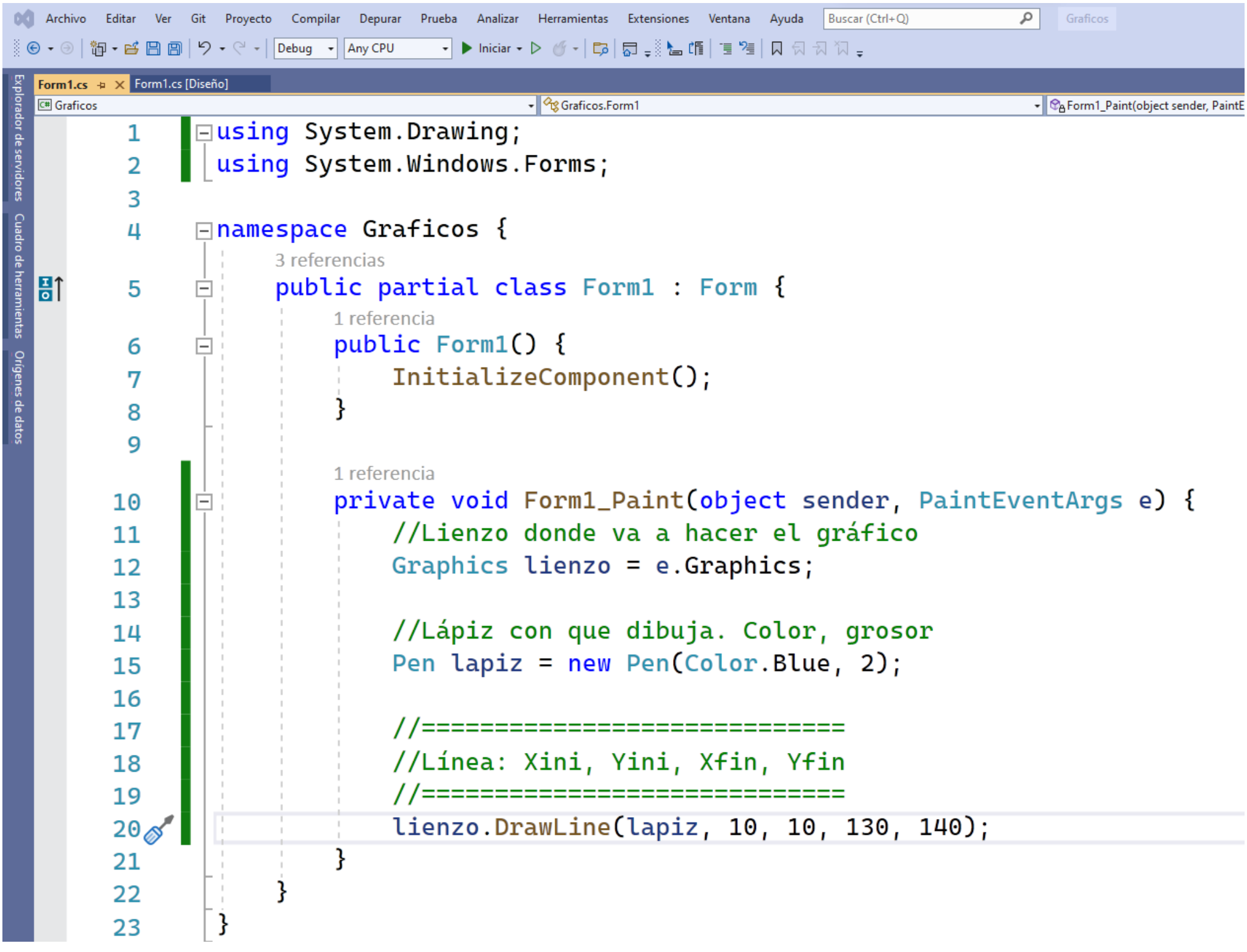


Ilustración 9: Código del Paint

01. Primitivas/01. Linea.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //==============================
            //Línea: Xini, Yini, Xfin, Yfin
            //==============================
            lienzo.DrawLine(lapiz, 10, 10, 130, 140);
        }
    }
}
```

Y este es el resultado al ejecutarlo:

Ilustración 10: Dibujo de una línea

Arco

01. Primitivas/02. Arco.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //=============================================================
            //Arco: Xpos, Ypos, Ancho, Alto, Ángulo Inicial, Ángulo Final
            //=============================================================
            lienzo.DrawArc(lapiz, 10, 90, 160, 170, 0, 270);
        }
    }
}
```

Ilustración 11: Arco

Rectángulo

01. Primitivas/03. Rectángulo.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //===================================
            //Rectángulo: Xpos, Ypos, Ancho, Alto
            //===================================
            lienzo.DrawRectangle(lapiz, 100, 100, 200, 150);
        }
    }
}
```

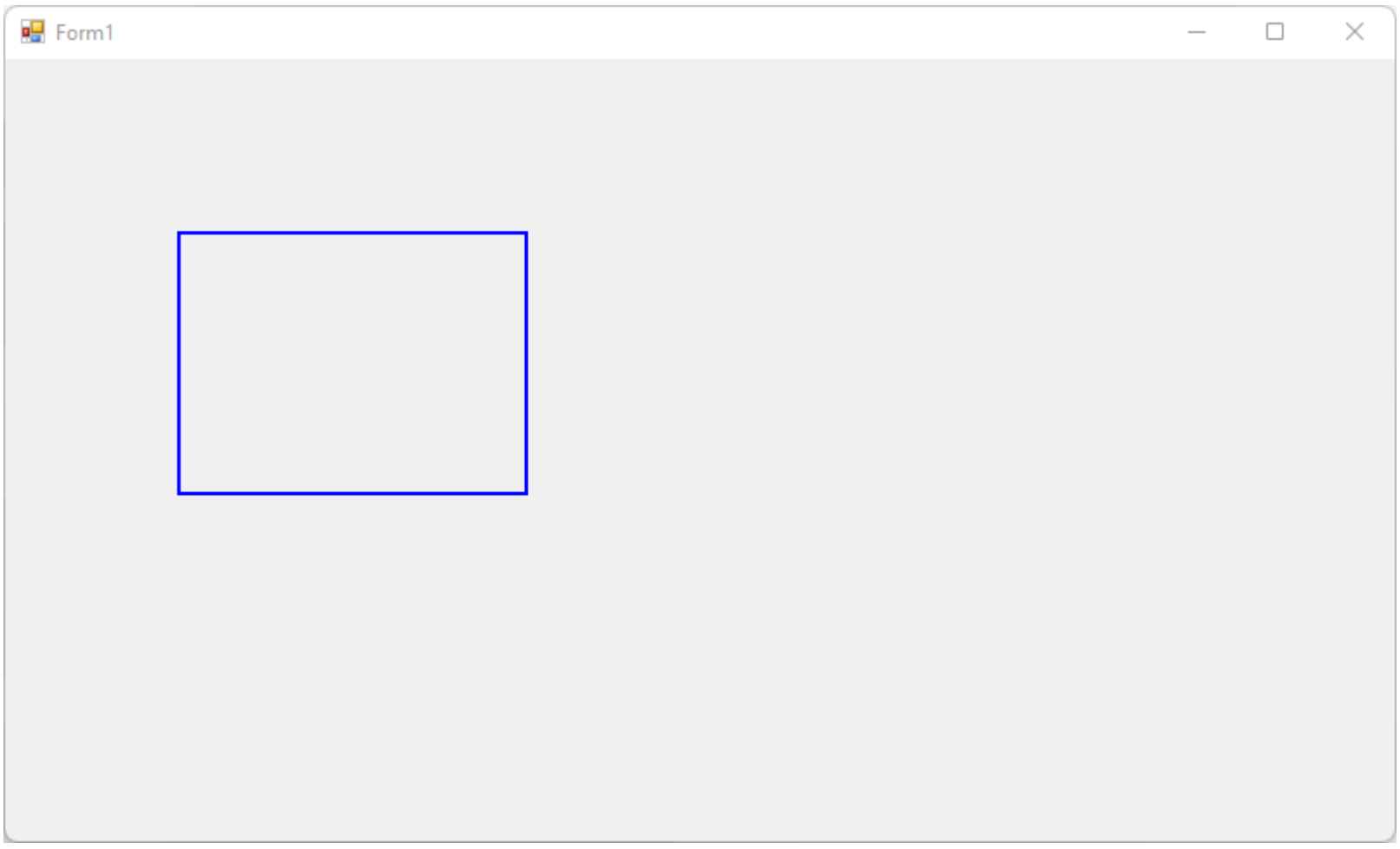


Ilustración 12: Rectángulo

Rectángulo relleno

01. Primitivas/04. Rectángulo Relleno.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Con qué color se rellenan las figuras
            SolidBrush Relleno = new SolidBrush(Color.Red);

            //===================================
            //Rectángulo: Xpos, Ypos, Ancho, Alto
            //===================================
            lienzo.FillRectangle(Relleno, 100, 100, 200, 150);
        }
    }
}
```

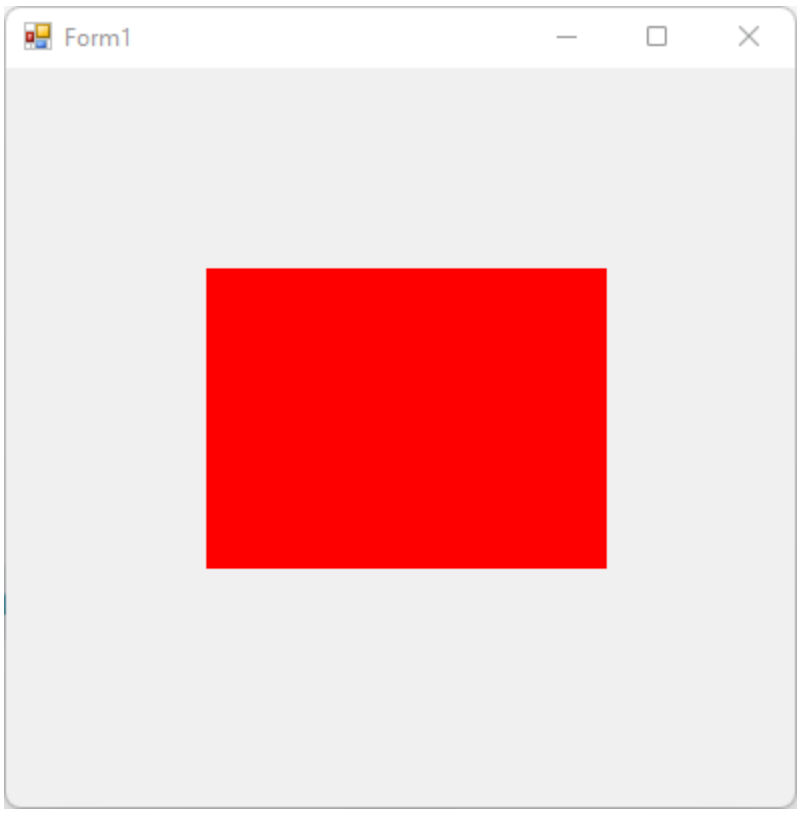


Ilustración 13: Rectángulo relleno

Curva de Bézier

Para más información de curvas Bézier [6]: https://es.wikipedia.org/wiki/Curva_de_B%C3%A9zier

01. Primitivas/05. Bézier.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //=========================================
            //Dibujar una Curva de Bézier
            //=========================================
            Point P0 = new Point(100, 180);
            Point P1 = new Point(200, 10);
            Point P2 = new Point(350, 50);
            Point P3 = new Point(500, 180);
            lienzo.DrawBezier(lapiz, P0, P1, P2, P3);
        }
    }
}
```

Ilustración 14: Curva de Bézier

Puntos y líneas

01. Primitivas/06. Puntos y Líneas.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //Conjunto de puntos
            PointF punto1 = new PointF(150.0F, 150.0F);
            PointF punto2 = new PointF(200.0F, 50.0F);
            PointF punto3 = new PointF(300.0F, 140.0F);
            PointF punto4 = new PointF(400.0F, 200.0F);
            PointF punto5 = new PointF(450.0F, 300.0F);
            PointF punto6 = new PointF(350.0F, 350.0F);
            PointF punto7 = new PointF(300.0F, 150.0F);
            PointF[] Puntos = { punto1, punto2, punto3, punto4, punto5, punto6, punto7 };

            //Dibuja líneas rectas para unir los puntos
            lienzo.DrawLines(lapiz, Puntos);
        }
    }
}
```

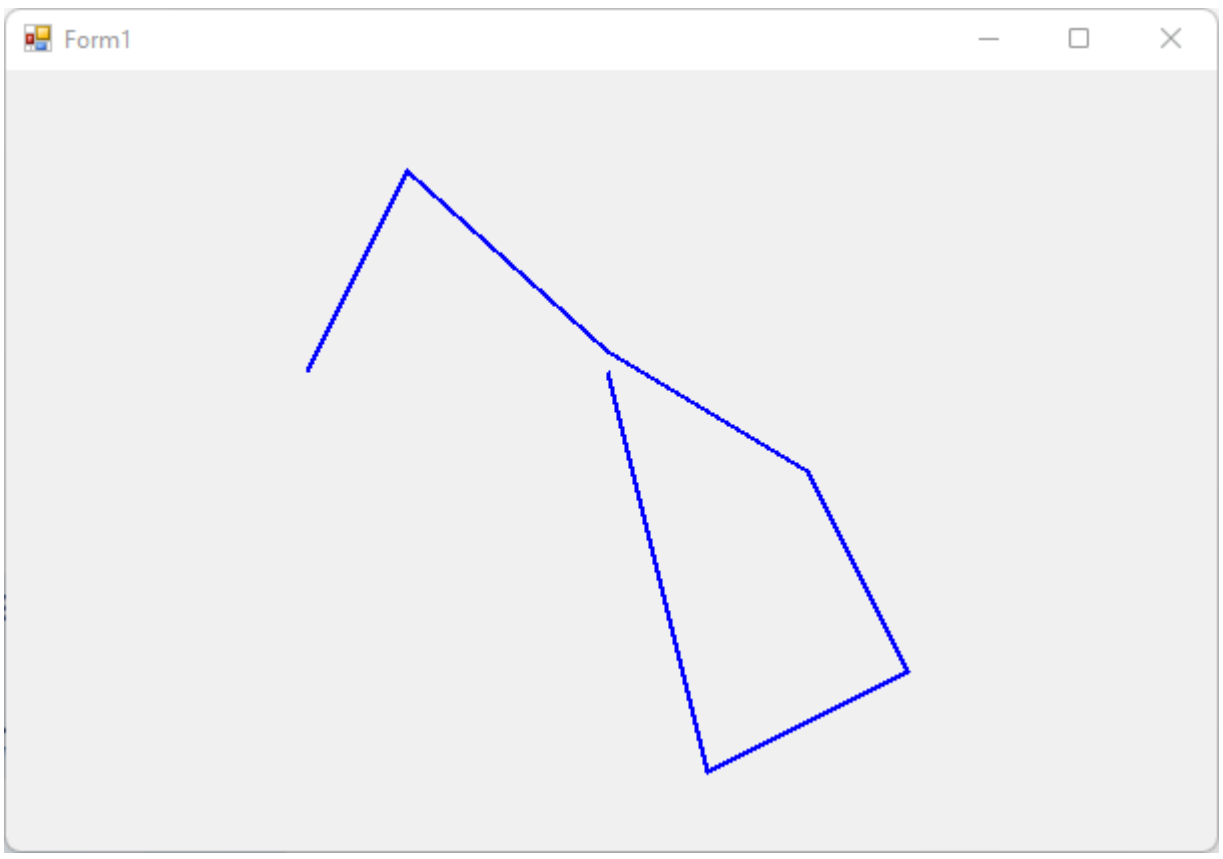


Ilustración 15: Puntos y líneas

Polígono con la serie de puntos

01. Primitivas/07. Polígono.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //Conjunto de puntos
            PointF punto1 = new PointF(150.0F, 150.0F);
            PointF punto2 = new PointF(200.0F, 50.0F);
            PointF punto3 = new PointF(300.0F, 140.0F);
            PointF punto4 = new PointF(400.0F, 200.0F);
            PointF punto5 = new PointF(450.0F, 300.0F);
            PointF punto6 = new PointF(350.0F, 350.0F);
            PointF punto7 = new PointF(300.0F, 150.0F);
            PointF[] Puntos = { punto1, punto2, punto3, punto4, punto5, punto6, punto7 };

            //Dibuja el polígono
            lienzo.DrawPolygon(lapiz, Puntos);
        }
    }
}
```

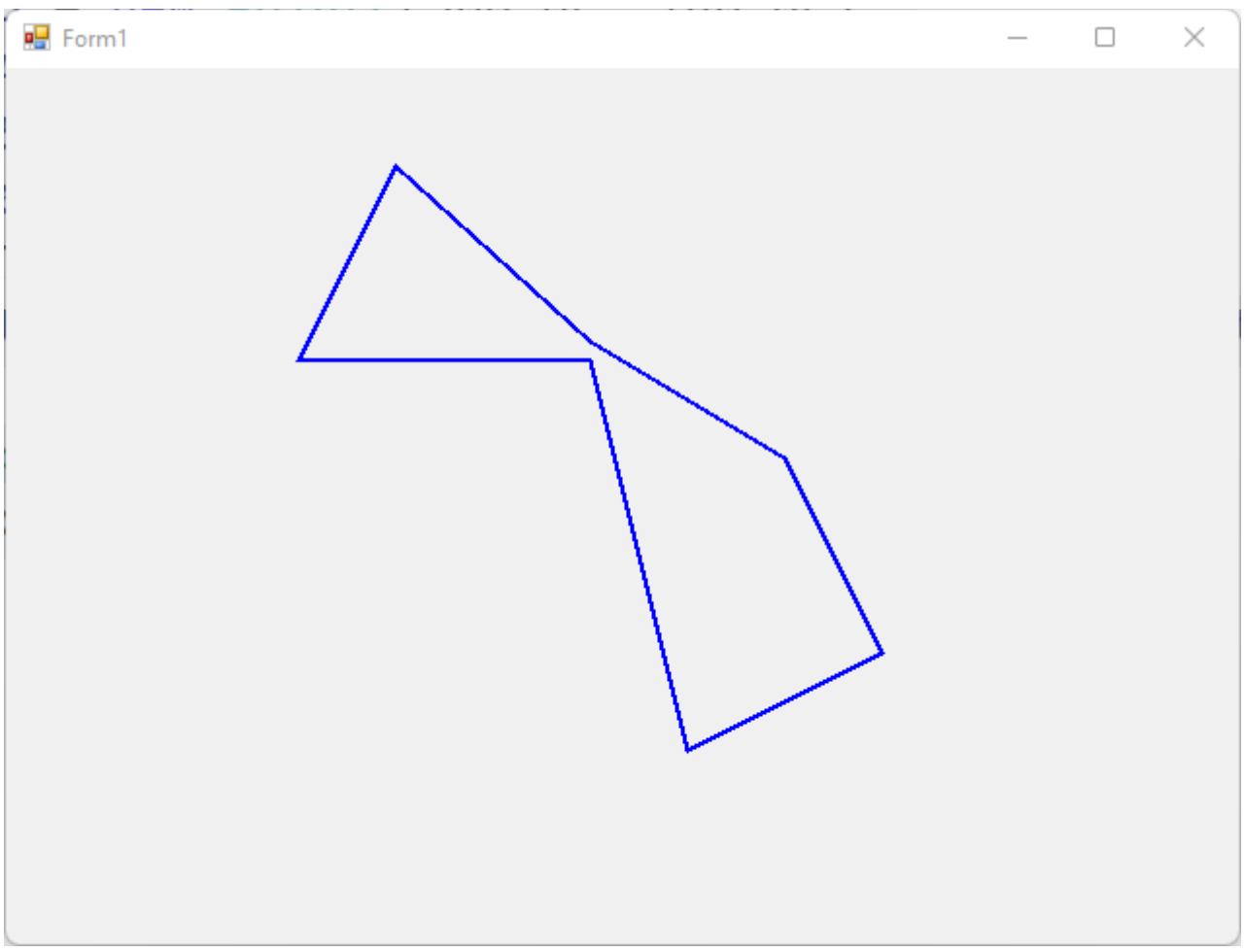


Ilustración 16: Polígono

Polígono relleno

01. Primitivas/08. Polígono Relleno.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Con qué color se rellenan las figuras
            SolidBrush Relleno = new SolidBrush(Color.Red);

            //Conjunto de puntos
            PointF punto1 = new PointF(150.0F, 150.0F);
            PointF punto2 = new PointF(200.0F, 50.0F);
            PointF punto3 = new PointF(300.0F, 140.0F);
            PointF punto4 = new PointF(400.0F, 200.0F);
            PointF punto5 = new PointF(450.0F, 300.0F);
            PointF punto6 = new PointF(350.0F, 350.0F);
            PointF punto7 = new PointF(300.0F, 150.0F);
            PointF[] Puntos = { punto1, punto2, punto3, punto4, punto5, punto6, punto7 };

            //Dibuja el polígono
            lienzo.FillPolygon(Relleno, Puntos);
        }
    }
}
```

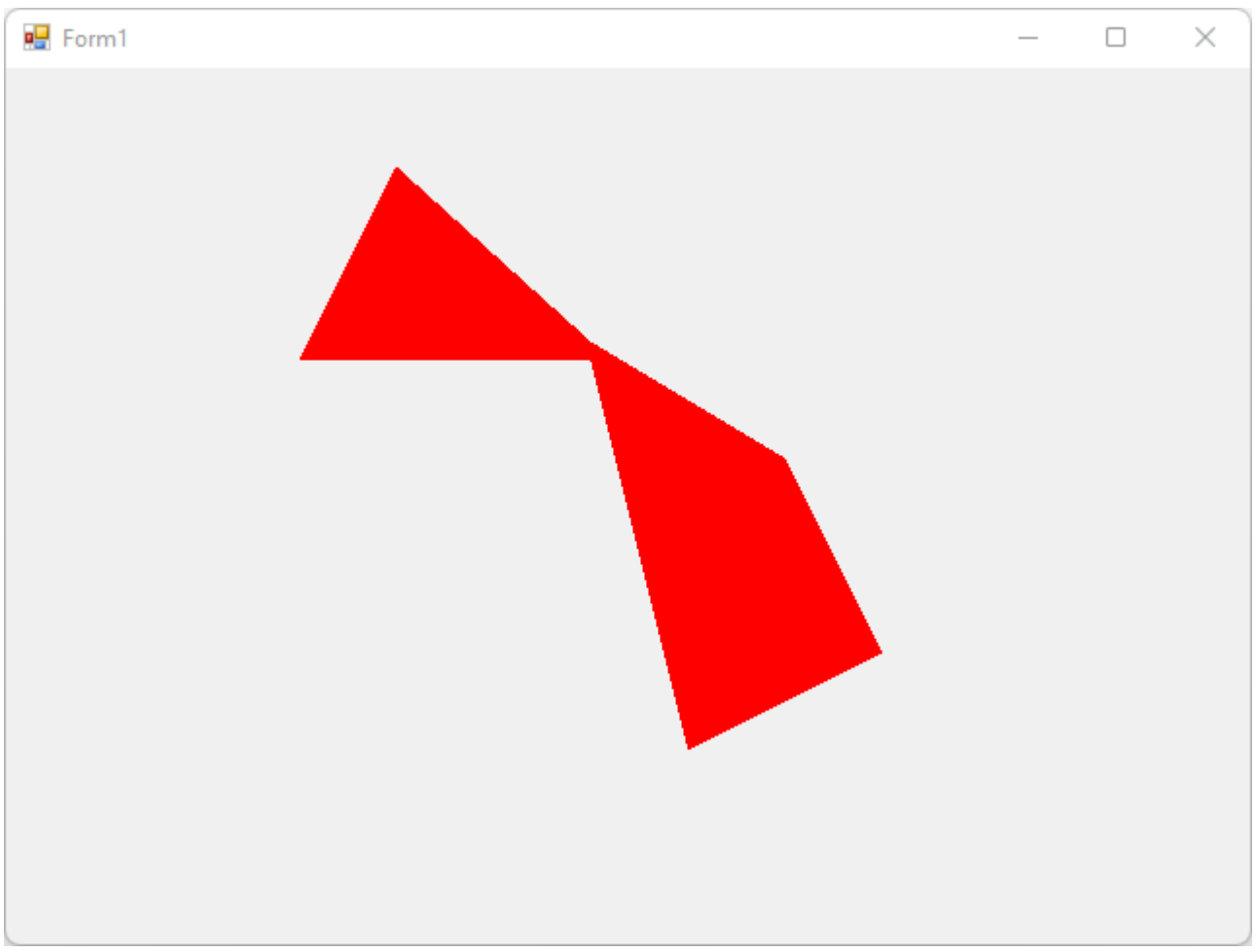


Ilustración 17: Polígono relleno

Curva cerrada a una serie de puntos

01. Primitivas/09. Curva Cerrada.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //Conjunto de Puntos
            PointF punto1 = new PointF(150.0F, 150.0F);
            PointF punto2 = new PointF(200.0F, 50.0F);
            PointF punto3 = new PointF(300.0F, 140.0F);
            PointF punto4 = new PointF(400.0F, 200.0F);
            PointF punto5 = new PointF(450.0F, 300.0F);
            PointF punto6 = new PointF(350.0F, 350.0F);
            PointF punto7 = new PointF(300.0F, 150.0F);
            PointF[] Puntos = { punto1, punto2, punto3, punto4, punto5, punto6, punto7 };

            //Dibuja una curva cerrada que une esos puntos
            lienzo.DrawClosedCurve(lapiz, Puntos);
        }
    }
}
```

Ilustración 18: Curva cerrada a una serie de puntos

Curva rellena cerrada a una serie de puntos

01. Primitivas/10. Curva Cerrada Rellena.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Relleno
            SolidBrush Relleno = new SolidBrush(Color.Gray);

            //Conjunto de Puntos
            PointF punto1 = new PointF(150.0F, 150.0F);
            PointF punto2 = new PointF(200.0F, 50.0F);
            PointF punto3 = new PointF(300.0F, 140.0F);
            PointF punto4 = new PointF(400.0F, 200.0F);
            PointF punto5 = new PointF(450.0F, 300.0F);
            PointF punto6 = new PointF(350.0F, 350.0F);
            PointF punto7 = new PointF(300.0F, 150.0F);
            PointF[] Puntos = { punto1, punto2, punto3, punto4, punto5, punto6, punto7 };

            //Dibuja una curva cerrada rellena que une esos puntos
            lienzo.FillClosedCurve(Relleno, Puntos);
        }
    }
}
```

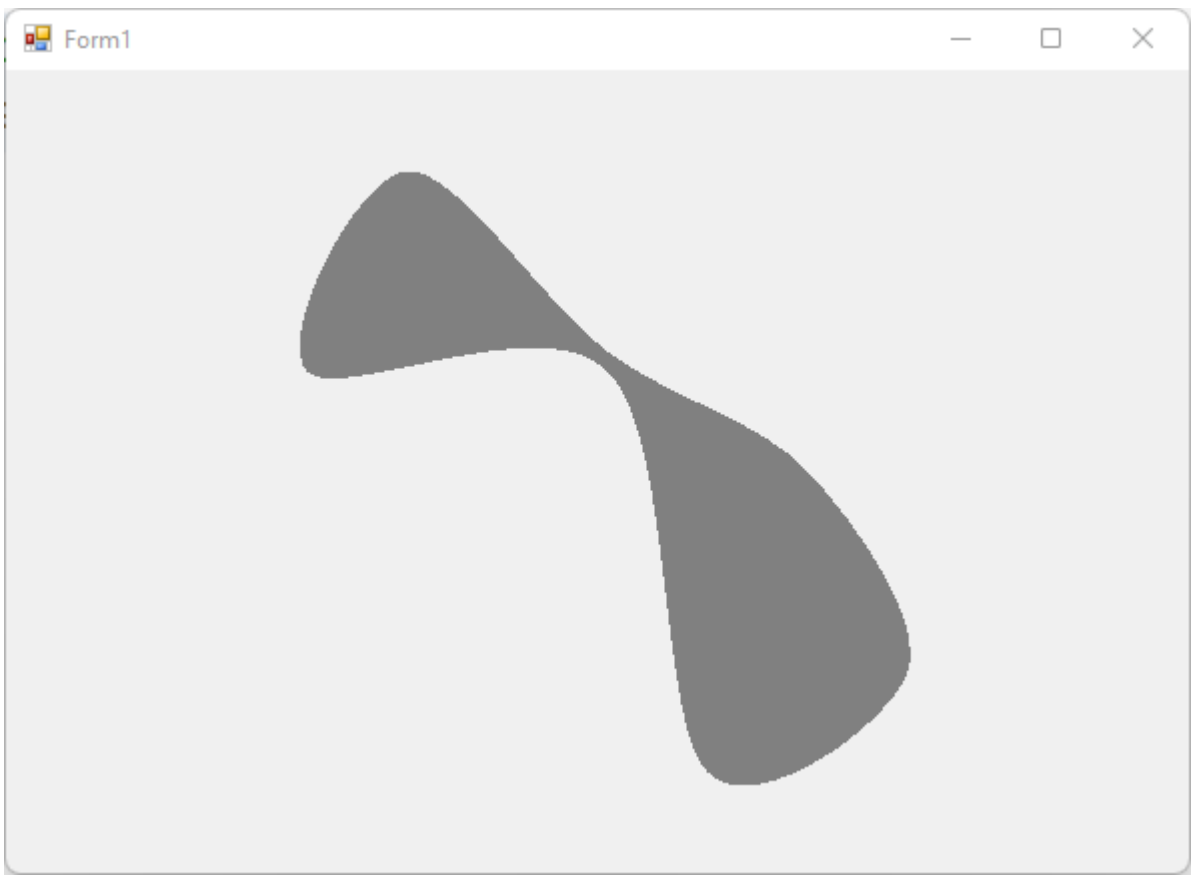


Ilustración 19: Curva rellena cerrada

Curva a una serie de puntos

01. Primitivas/11. Curva a serie de puntos.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //Conjunto de Puntos
            PointF punto1 = new PointF(150.0F, 150.0F);
            PointF punto2 = new PointF(200.0F, 50.0F);
            PointF punto3 = new PointF(300.0F, 140.0F);
            PointF punto4 = new PointF(400.0F, 200.0F);
            PointF punto5 = new PointF(450.0F, 300.0F);
            PointF punto6 = new PointF(350.0F, 350.0F);
            PointF punto7 = new PointF(300.0F, 150.0F);
            PointF[] Puntos = { punto1, punto2, punto3, punto4, punto5, punto6, punto7 };

            //Dibuja la curva que une esos puntos
            lienzo.DrawCurve(lapiz, Puntos);
        }
    }
}
```

Ilustración 20: Curva a una serie de puntos

Elipse

01. Primitivas/12. Elipse.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //================================
            //Elipse: Xpos, Ypos, ancho, alto
            //================================
            lienzo.DrawEllipse(lapiz, 200, 30, 250, 90);
        }
    }
}
```

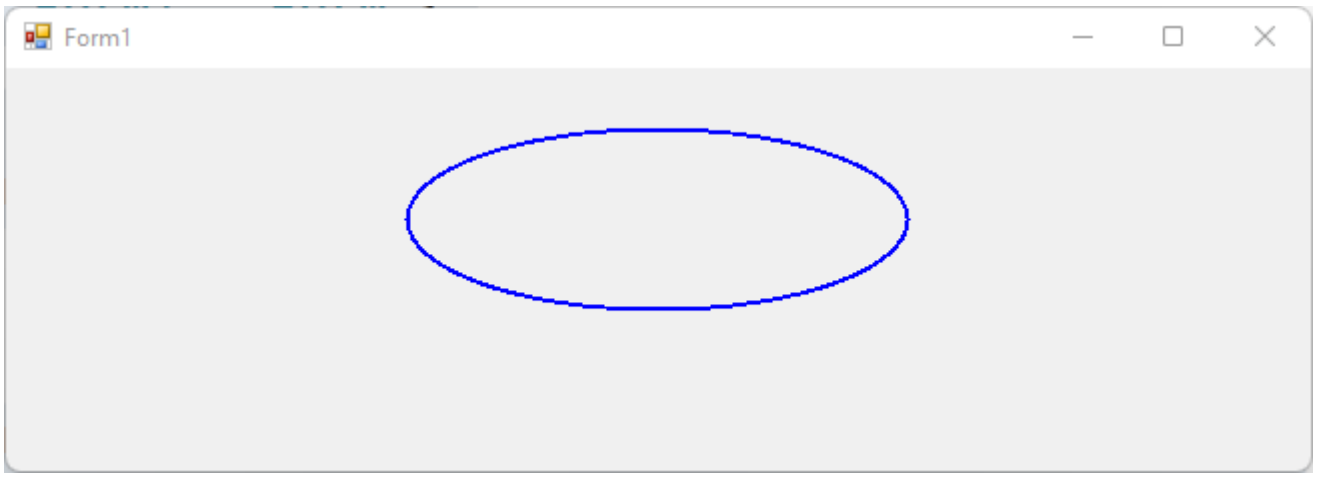


Ilustración 21: Elipse

Elipse rellena

01. Primitivas/13. Elipse Rellena.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Relleno
            SolidBrush Relleno = new SolidBrush(Color.Chocolate);

            //================================
            //Elipse: Xpos, Ypos, ancho, alto
            //================================
            lienzo.FillEllipse(Relleno, 200, 30, 250, 90);
        }
    }
}
```

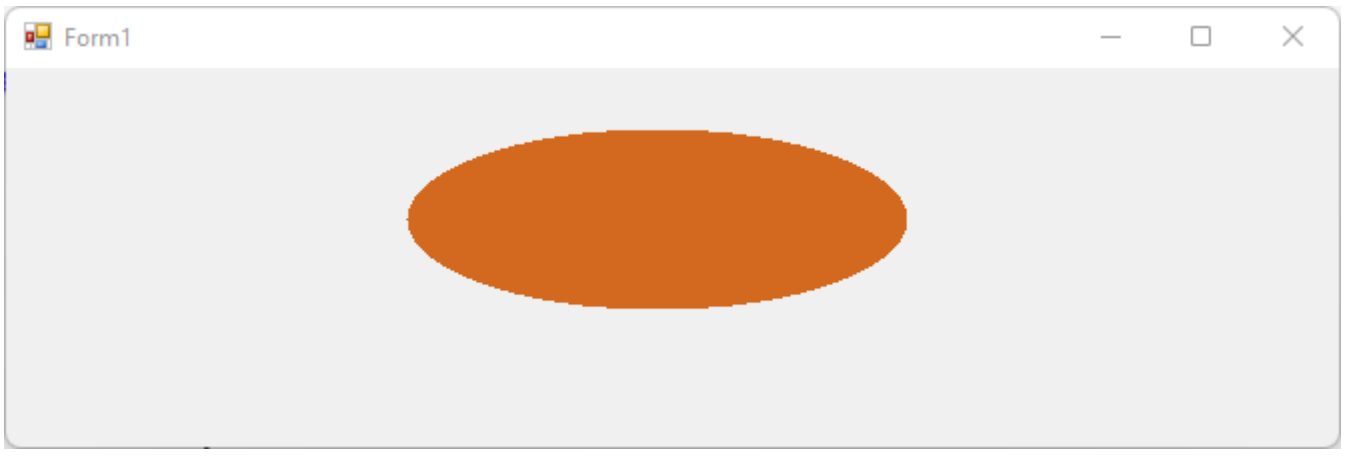


Ilustración 22: Elipse rellena

Dibujar cadenas o letras

01. Primitivas/14. Cadenas y Letras.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //==============
            //Letras
            //==============
            string Cadena = "Esta es una prueba";

            //Fuente y la brocha con que se pinta.
            Font FuenteLetra = new Font("Tahoma", 16);
            SolidBrush BrochaPinta = new SolidBrush(Color.Black);

            //Punto arriba a la izquierda para pintar la cadena
            PointF PuntoCadena = new PointF(100.0F, 140.0F);

            //Formato para dibujar
            StringFormat FormatoDibuja = new StringFormat();
            FormatoDibuja.FormatFlags = StringFormatFlags.DirectionVertical;

            //Dibuja la cadena
            lienzo.DrawString(Cadena, FuenteLetra, BrochaPinta, PuntoCadena, FormatoDibuja);
        }
    }
}
```

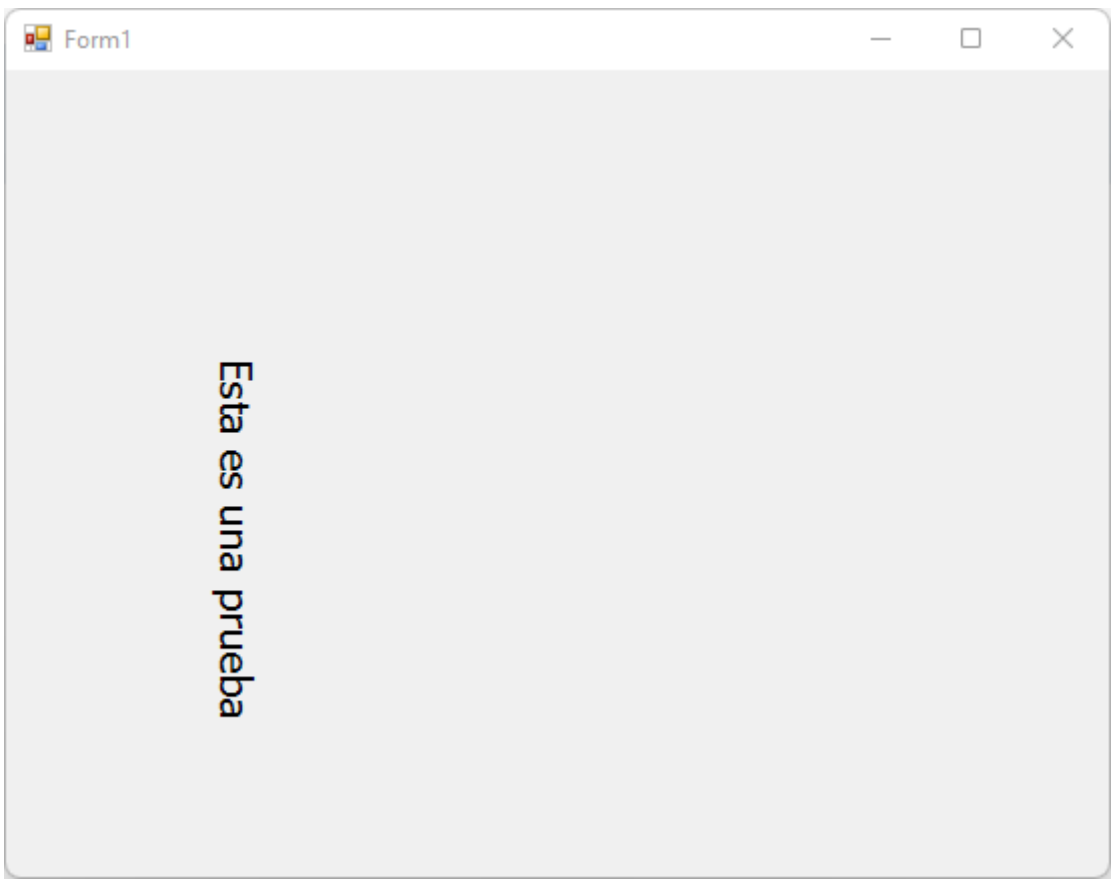


Ilustración 23: Dibujar cadenas

Diagrama de pastel

01. Primitivas/15. Pastel.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Lápiz con que dibuja. Color, grosor
            Pen lapiz = new Pen(Color.Blue, 2);

            //================
            //Dibuja un pastel
            //================

            //Crea un rectángulo para la elipse que dibujará el pastel
            Rectangle rectangulo = new Rectangle(10, 10, 400, 400);

            //Pastel: rectángulo que contiene, ángulo inicial, ángulo de apertura
            lienzo.DrawPie(lapiz, rectangulo, 0F, 45F);
            lienzo.DrawPie(lapiz, rectangulo, 90F, 45F);
            lienzo.DrawPie(lapiz, rectangulo, 150F, 45F);
        }
    }
}
```

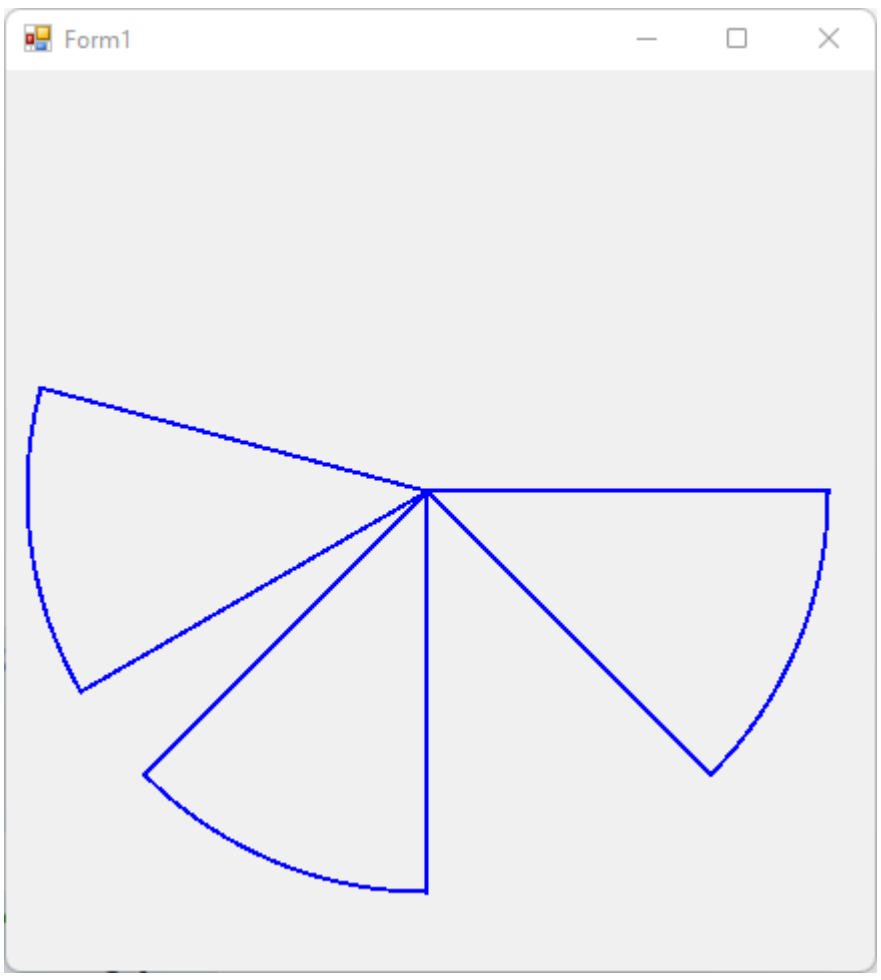


Ilustración 24: Un diagrama de pastel

Diagrama de pastel relleno

01. Primitivas/15. Pastel Relleno.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo donde va a hacer el gráfico
            Graphics lienzo = e.Graphics;

            //Rellenos
            SolidBrush Relleno1 = new SolidBrush(Color.Chocolate);
            SolidBrush Relleno2 = new SolidBrush(Color.Red);
            SolidBrush Relleno3 = new SolidBrush(Color.Blue);

            //================
            //Dibuja un pastel
            //================

            //Crea un rectángulo para la elipse que dibujará el pastel
            Rectangle rectangulo = new Rectangle(10, 10, 400, 400);

            //Pastel: rectángulo que contiene, ángulo inicial, ángulo de apertura
            lienzo.FillPie(Relleno1, rectangulo, 0F, 45F);
            lienzo.FillPie(Relleno2, rectangulo, 90F, 45F);
            lienzo.FillPie(Relleno3, rectangulo, 150F, 45F);
        }
    }
}
```

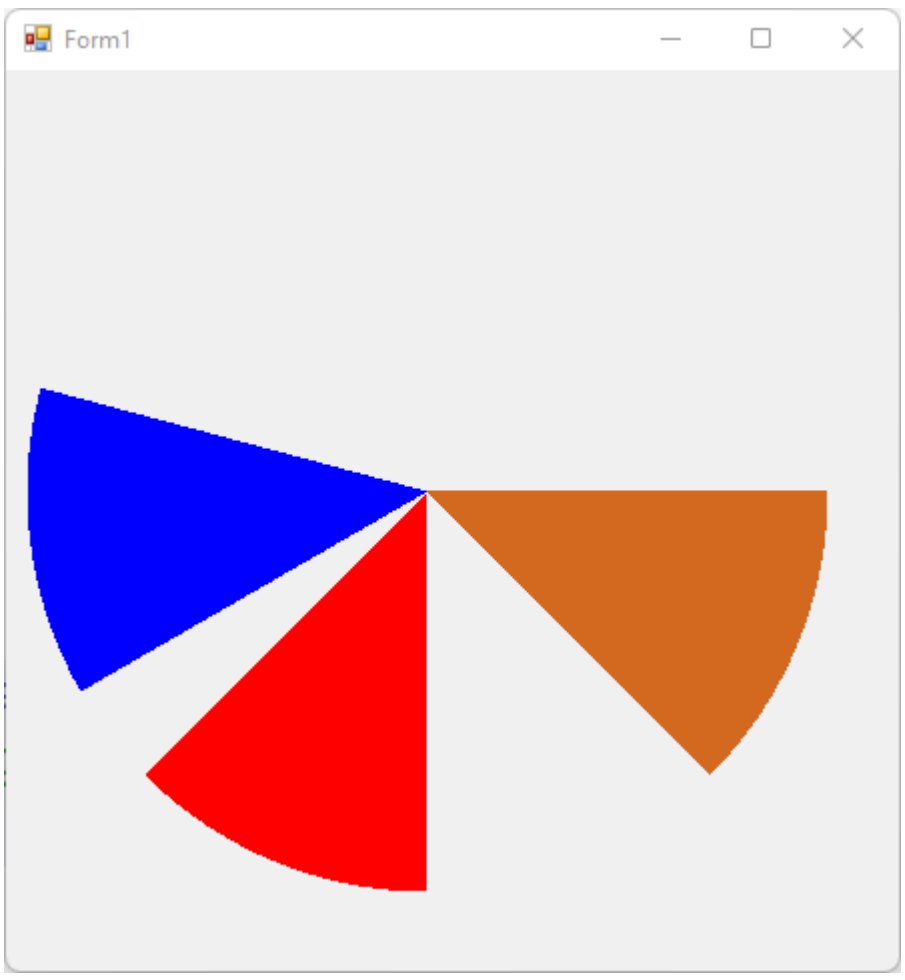


Ilustración 25: Diagrama de pastel con tajadas rellenas de diferentes colores

Gráficos usando ciclos

Líneas horizontales y espaciado

02. Ciclos/01. Líneas Horizontales.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo
            Graphics grafico = e.Graphics;

            //Primer gráfico
            Pen lapiz = new Pen(Color.Blue, 2);
            int contador = 10;
            int incremento = 2;

            do {
                grafico.DrawLine(lapiz, 10, contador, 300, contador);
                incremento++;
                contador += incremento; //Incrementa el espacio entre línea y línea
            } while (contador <= 602);
            contador -= incremento;

            //Segundo gráfico
            Pen lapiz2 = new Pen(Color.Red, 2);
            incremento = 2;
            do {
                grafico.DrawLine(lapiz2, 300, contador, 590, contador);
                incremento++;
                contador -= incremento;
            } while (contador >= 10);
        }
    }
}
```

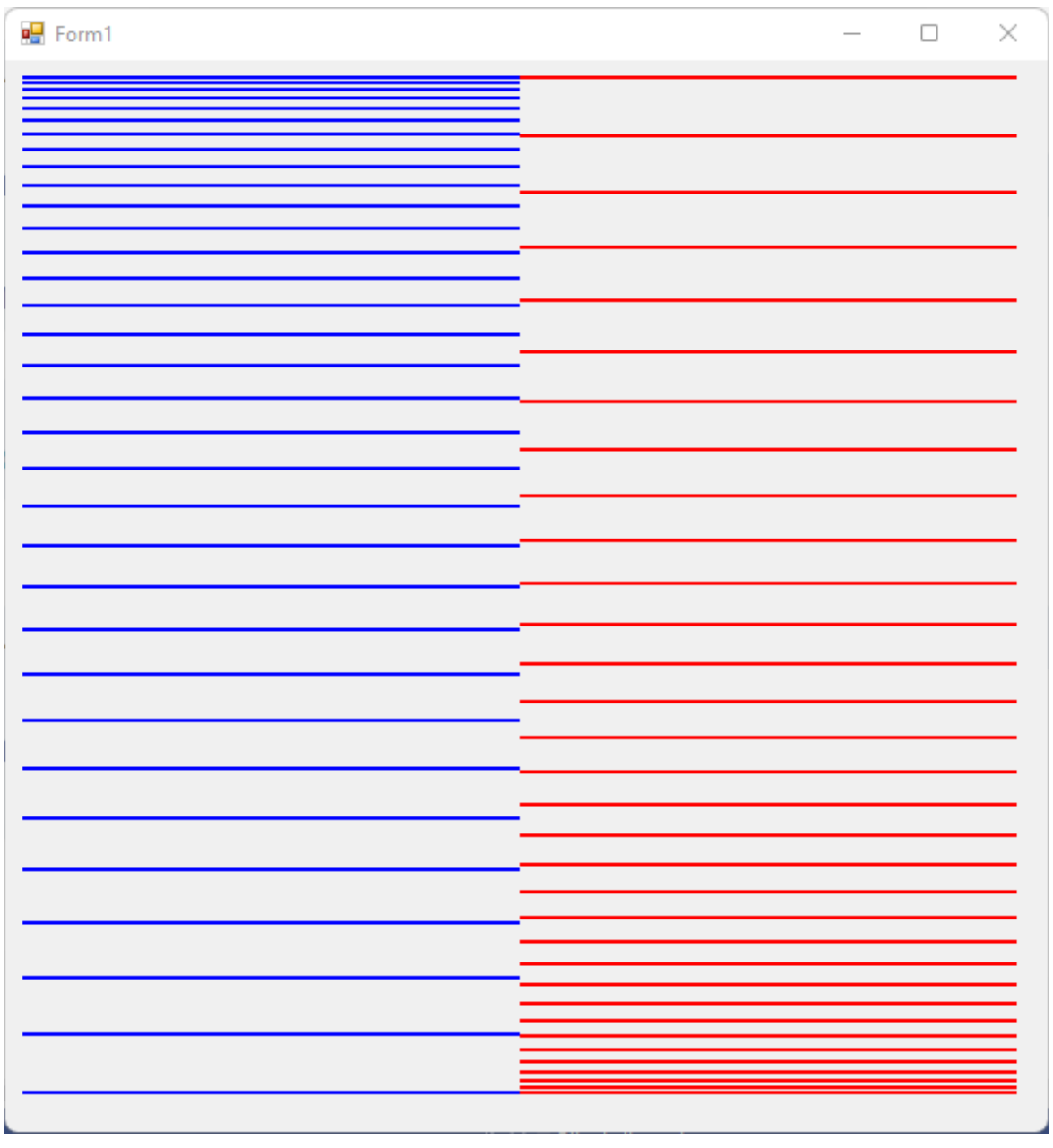


Ilustración 26: Líneas horizontales, variando el espacio

Líneas verticales y espaciado

02. Ciclos/02. Líneas Verticales.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Lienzo
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            int contador = 10;
            int incremento = 2;

            //Primer gráfico
            do {
                grafico.DrawLine(lapiz, contador, 10, contador, 200);
                incremento++;
                contador += incremento; //Incrementa el espacio entre línea y línea
            }
            while (contador <= 602);
            contador -= incremento;

            //Segundo gráfico
            Pen lapiz2 = new Pen(Color.Red, 2);
            incremento = 2;
            do {
                grafico.DrawLine(lapiz2, contador, 200, contador, 380);
                incremento++;
                contador -= incremento;
            }
            while (contador >= 10);
        }
    }
}
```

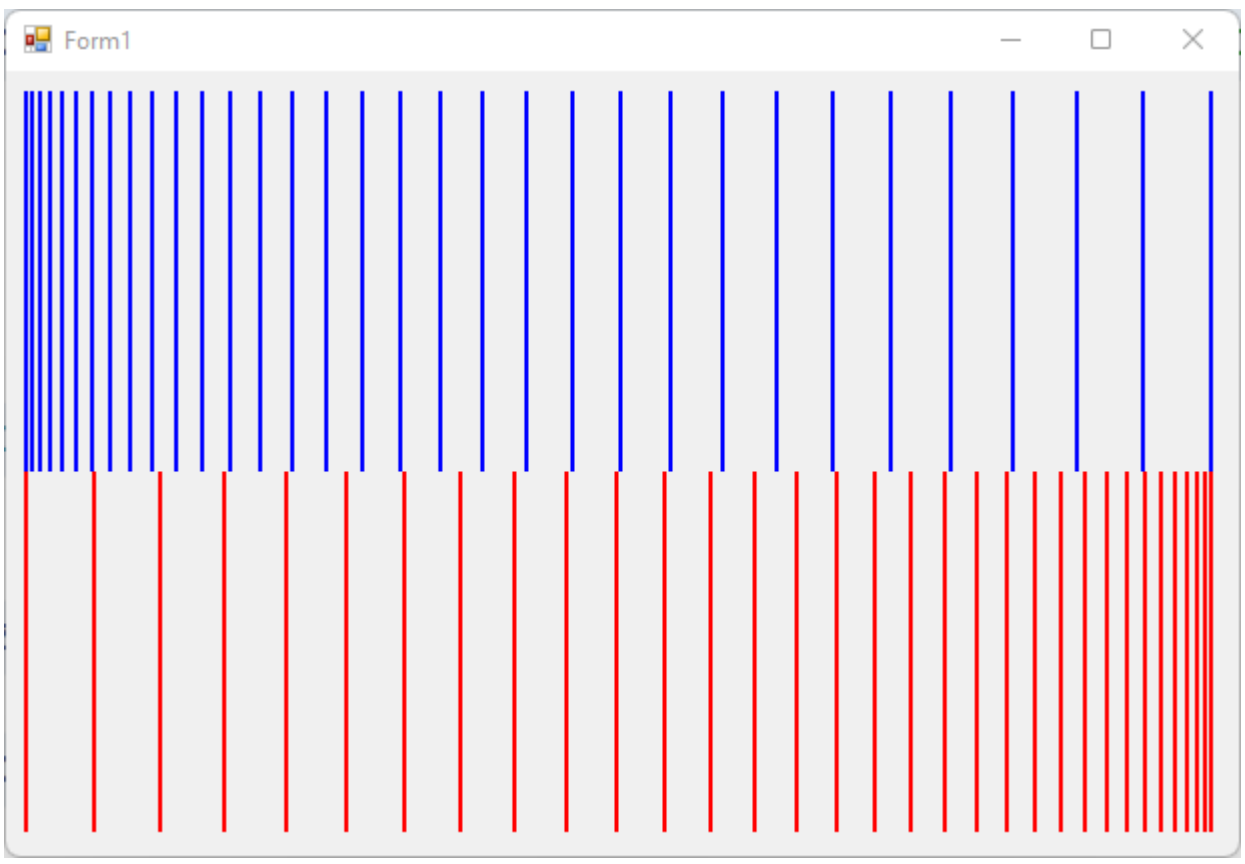


Ilustración 27: Líneas verticales, variando el espaciado

Líneas verticales izquierda y derecha

02. Ciclos/ 03. Líneas Verticales Izquierda Derecha.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            int limite = 500;
            int contador = limite;
            int incremento = 2;

            do {
                grafico.DrawLine(lapiz, contador, 10, contador, 200);
                grafico.DrawLine(lapiz, limite - (contador - limite), 10, limite - (contador - limite),
200);
                grafico.DrawLine(lapiz2, contador, 200, contador, 380);
                grafico.DrawLine(lapiz2, limite - (contador - limite), 200, limite - (contador - limite),
380);
                incremento++;
                contador += incremento;
            }
            while (contador <= 2 * limite);
        }
    }
}
```

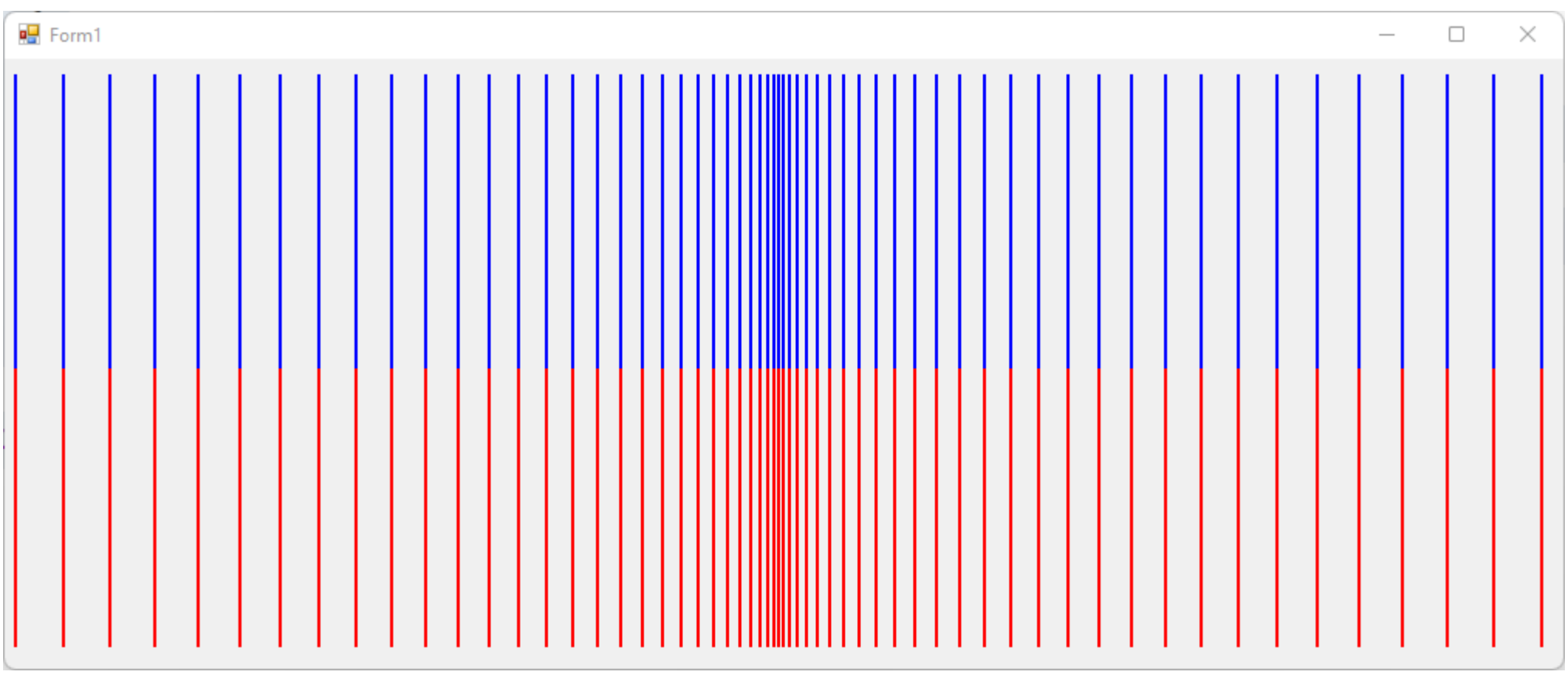


Ilustración 28: Líneas verticales, izquierda y derecha

Rombo

02. Ciclos/ 04. Rombo.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            int xval = 10;
            int yval = 5;

            do {
                grafico.DrawLine(lapiz, xval, 300 - yval, xval, 300 + yval);
                xval += 5;
                yval += 5;
            }
            while (yval < 300);

            do {
                grafico.DrawLine(lapiz2, xval, 300 - yval, xval, 300 + yval);
                xval += 5;
                yval -= 5;
            }
            while (yval > 0);
        }
    }
}
```

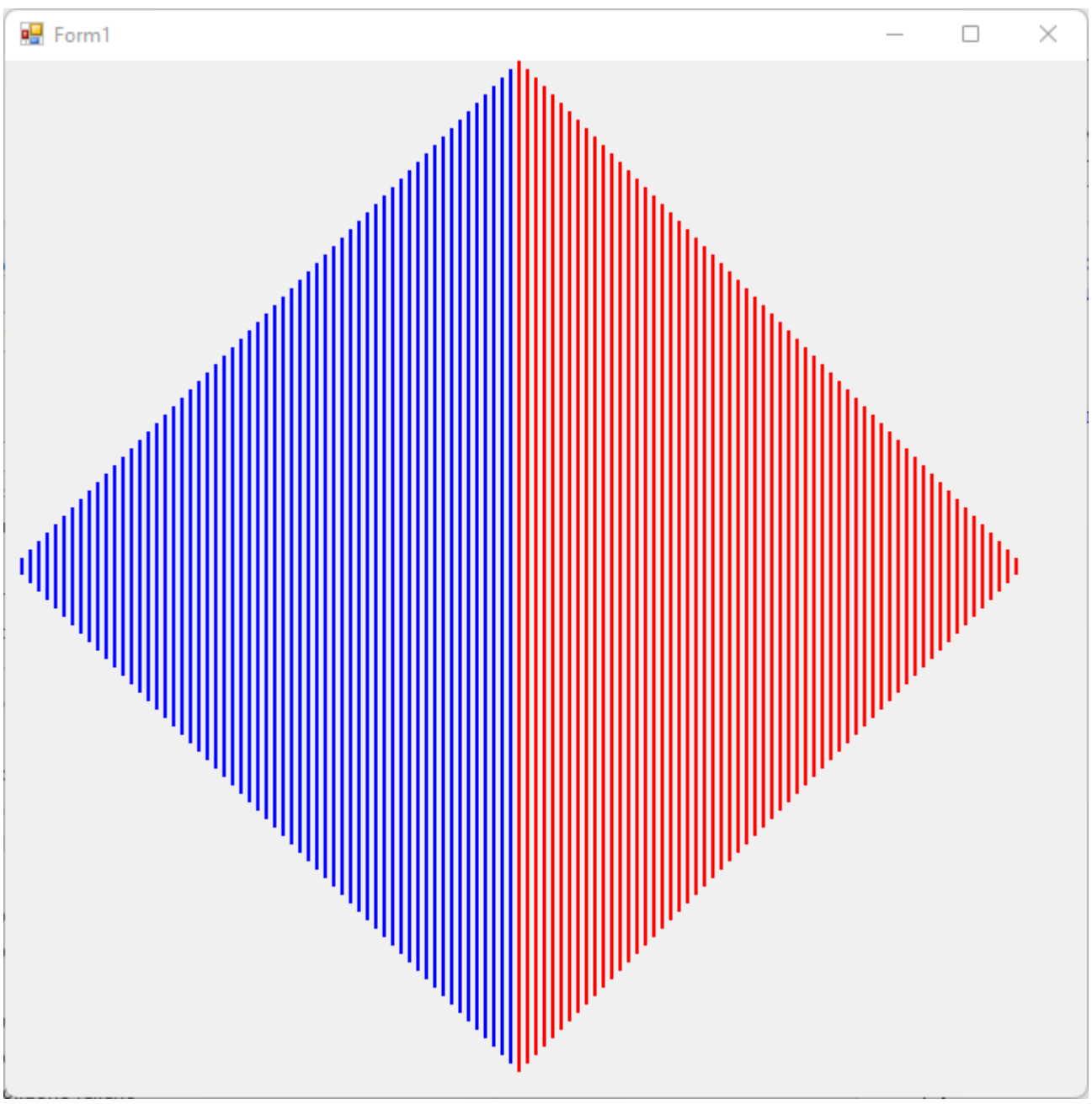


Ilustración 29: Rombo

Esquina superior derecha

02. Ciclos/05. Esquina Superior Derecha.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            int xval = 600;
            int yval = 300;

            do {
                grafico.DrawLine(lapiz, xval, 300 - yval, xval, 300 + yval);
                xval -= 5;
                yval -= 5;
            }
            while (yval >= 0);

            xval = 300;
            yval = 0;
            do {
                grafico.DrawLine(lapiz2, 300 - xval, yval, 300 + xval, yval);
                xval -= 5;
                yval += 5;
            }
            while (yval <= 300);
        }
    }
}
```

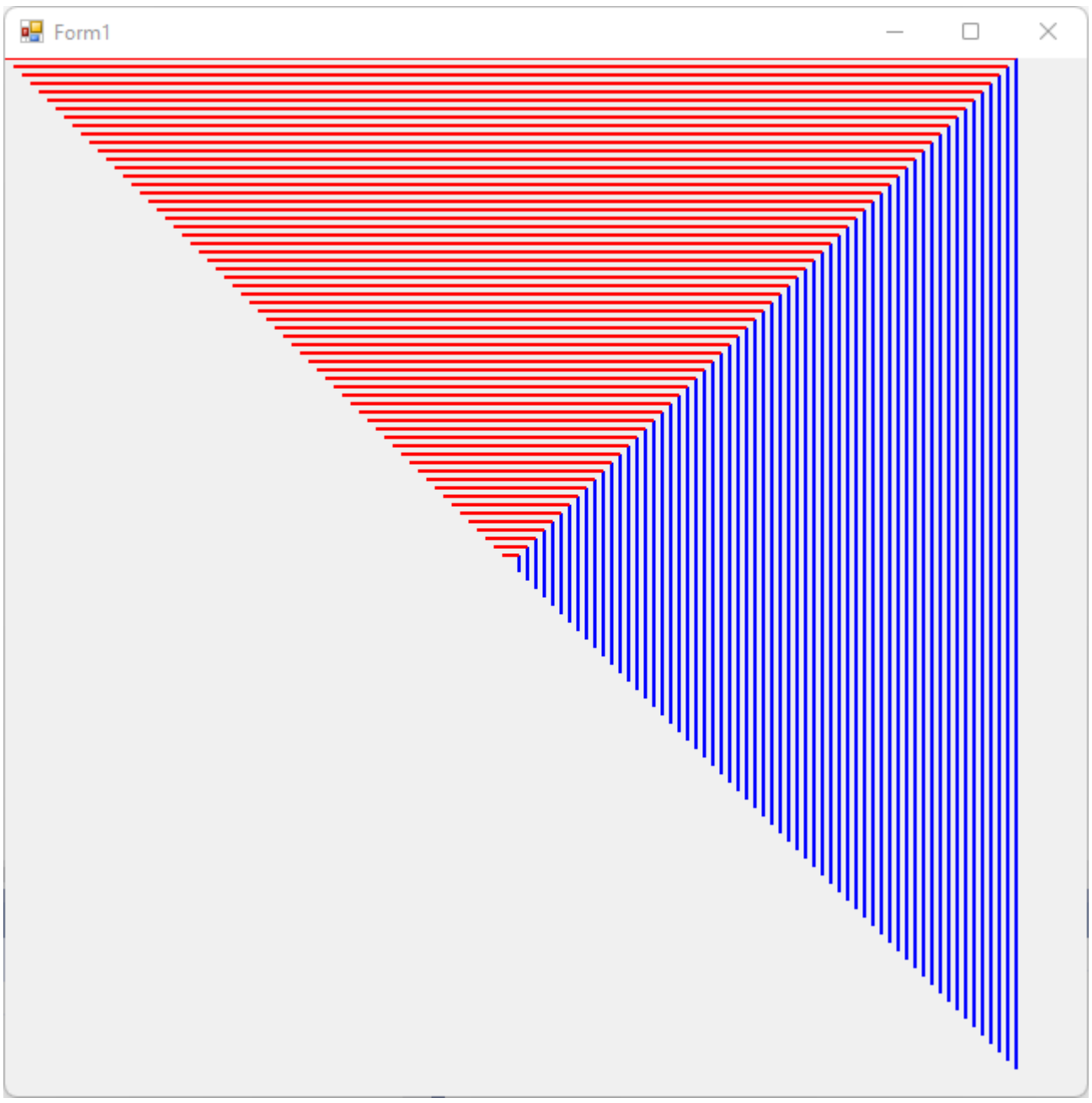


Ilustración 30: Esquina superior derecha

Esquina inferior derecha

02. Ciclos/06. Esquina Inferior Derecha.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            int xval = 600;
            int yval = 300;

            do {
                grafico.DrawLine(lapiz, xval, 300 - yval, xval, 300 + yval);
                xval -= 5;
                yval -= 5;
            }
            while (yval >= 0);

            xval = 300;
            yval = 600;
            do {
                grafico.DrawLine(lapiz2, 300 - xval, yval, 300 + xval, yval);
                xval -= 5;
                yval -= 5;
            }
            while (yval >= 300);
        }
    }
}
```

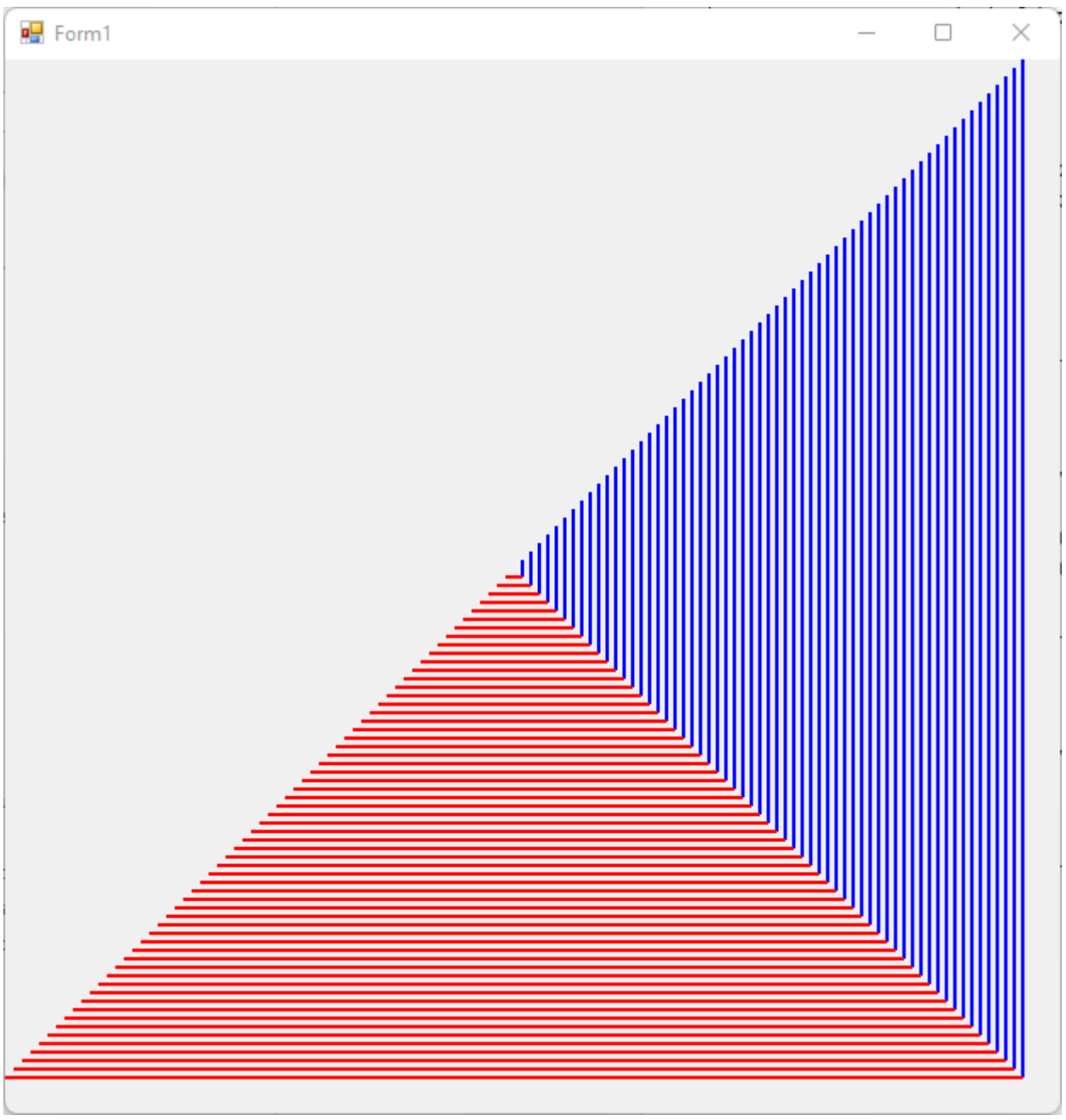


Ilustración 31: Esquina inferior derecha

Esquina inferior izquierda

02. Ciclos/07. Esquina Inferior Izquierda.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            int xval = 0;
            int yval = 300;

            do {
                grafico.DrawLine(lapiz, xval, 300 - yval, xval, 300 + yval);
                xval += 5;
                yval -= 5;
            }
            while (yval >= 0);

            xval = 300;
            yval = 600;
            do {
                grafico.DrawLine(lapiz2, 300 - xval, yval, 300 + xval, yval);
                xval -= 5;
                yval -= 5;
            }
            while (yval >= 300);
        }
    }
}
```

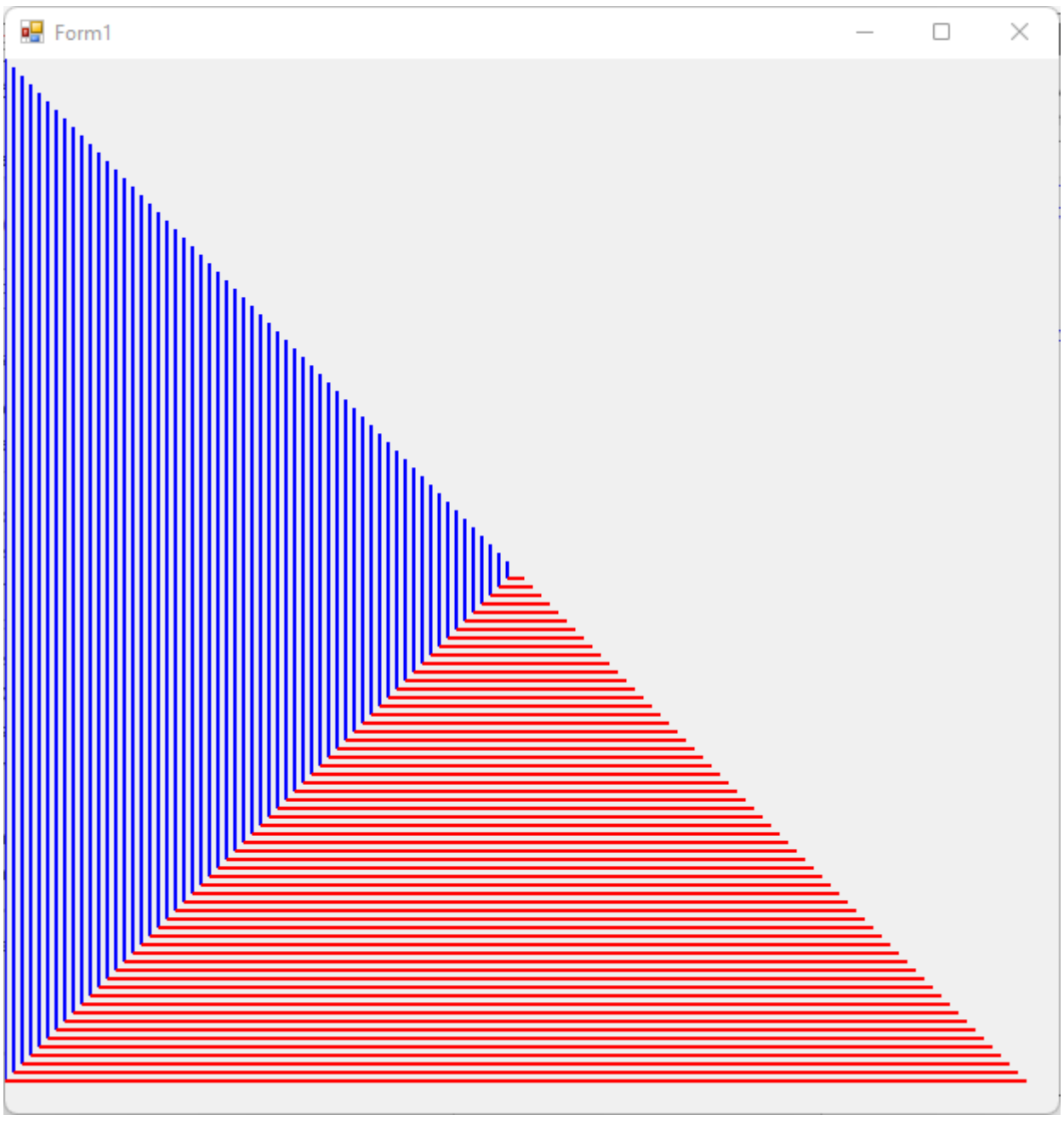


Ilustración 32: Esquina inferior izquierda

Triángulos apuntando a la derecha

02. Ciclos/08. Triángulos apuntando a la derecha.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);

            int pX1, pY1, pX2, pY2, pX3, pY3, constante;
            pX1 = 420;
            pY1 = 360;
            pX2 = 480;
            pY2 = 400;
            pX3 = 420;
            pY3 = 440;
            constante = 20;
            do {
                grafico.DrawLine(lapiz, pX1, pY1, pX2, pY2);
                grafico.DrawLine(lapiz, pX2, pY2, pX3, pY3);
                grafico.DrawLine(lapiz, pX1, pY1, pX3, pY3);
                pX1 -= constante;
                pY1 -= constante;
                pX2 += constante;
                pX3 -= constante;
                pY3 += constante;
            }
            while (pY1 >= 0);
        }
    }
}
```

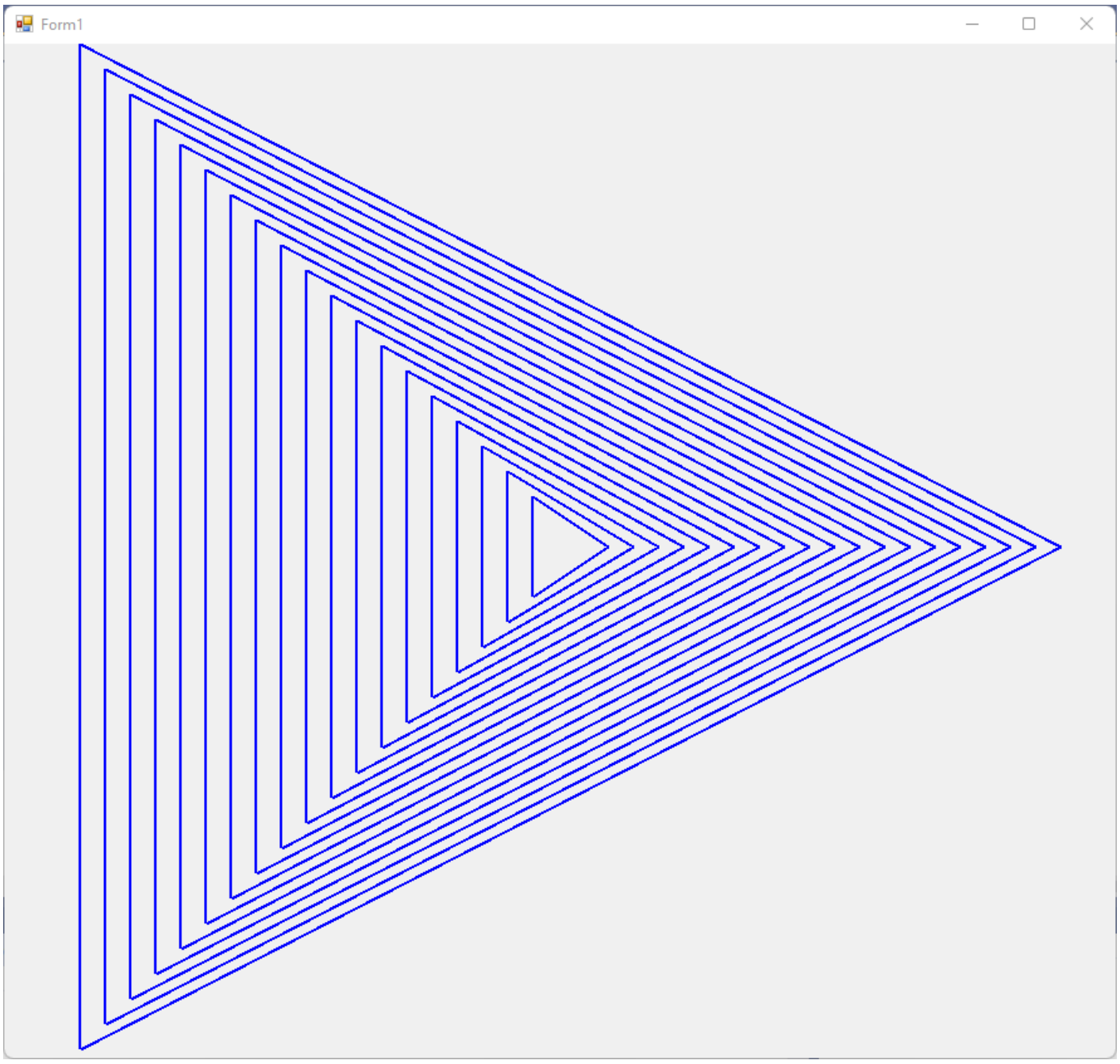


Ilustración 33: Triángulos apuntando a la derecha

Triángulos apuntando a la izquierda

02. Ciclos/09. Triángulos apuntando a la izquierda.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);

            int pX1, pY1, pX2, pY2, pX3, pY3, constante;
            pX1 = 420;
            pY1 = 360;
            pX2 = 360;
            pY2 = 400;
            pX3 = 420;
            pY3 = 440;
            constante = 20;
            do {
                grafico.DrawLine(lapiz, pX1, pY1, pX2, pY2);
                grafico.DrawLine(lapiz, pX2, pY2, pX3, pY3);
                grafico.DrawLine(lapiz, pX1, pY1, pX3, pY3);
                pX1 += constante;
                pY1 -= constante;
                pX2 -= constante;
                pX3 += constante;
                pY3 += constante;
            }
            while (pY1 >= 0);
        }
    }
}
```

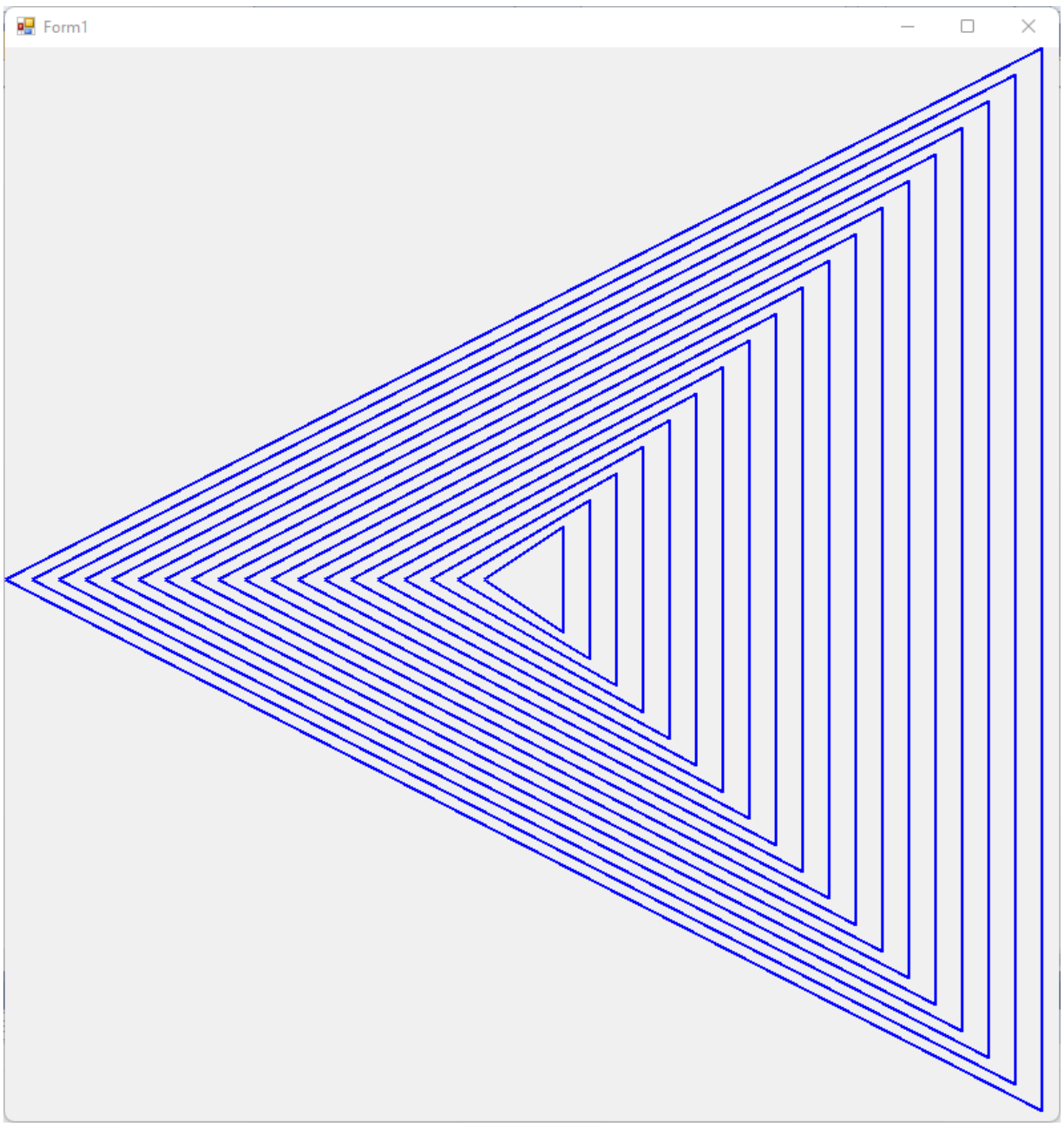


Ilustración 34: Triángulos apuntando a la izquierda

Triángulos azul - rojo apuntando a la izquierda

02. Ciclos/10. Triángulos azul rojo apuntando a la izquierda.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz;
            Pen lapizRojo = new Pen(Color.Red, 2);
            Pen lapizAzul = new Pen(Color.Blue, 2);

            int pX1, pY1, pX2, pY2, pX3, pY3, constante, cambia;
            pX1 = 220;
            pY1 = 360;
            pX2 = 160;
            pY2 = 400;
            pX3 = 220;
            pY3 = 440;
            constante = 30;
            cambia = 1;
            do {
                if (cambia == 1) {
                    cambia = 2;
                    lapiz = lapizAzul;
                }
                else {
                    cambia = 1;
                    lapiz = lapizRojo;
                }

                grafico.DrawLine(lapiz, pX1, pY1, pX2, pY2);
                grafico.DrawLine(lapiz, pX2, pY2, pX3, pY3);
                grafico.DrawLine(lapiz, pX1, pY1, pX3, pY3);
                pX1 += constante;
                pY1 -= constante;
                pX2 += constante;
                pX3 += constante;
                pY3 += constante;
            }
            while (pY1 >= 0);
        }
    }
}
```

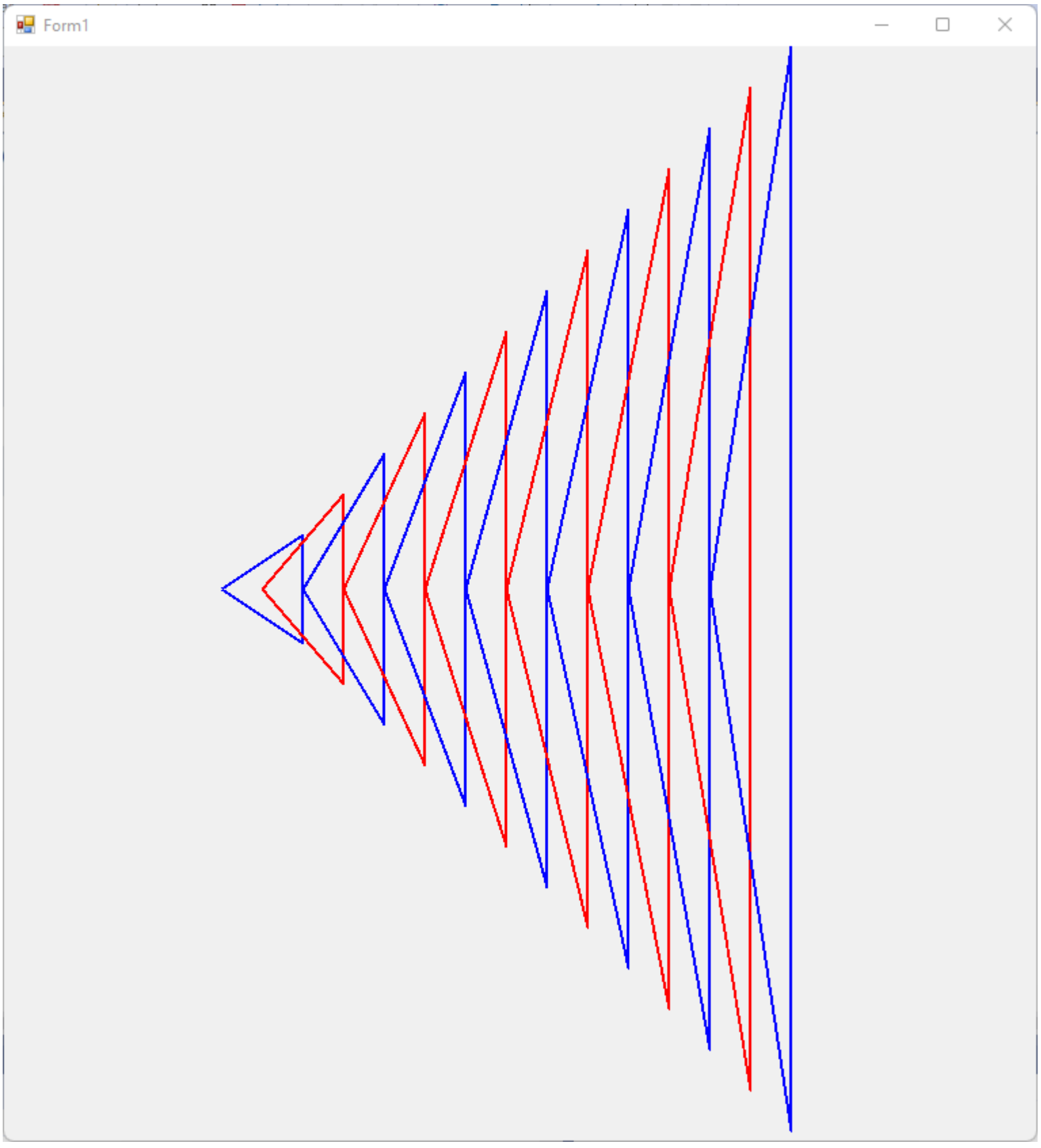


Ilustración 35: Triángulos azul-rojo apuntando a la izquierda

Cuadrados concéntricos

02. Ciclos/11. Cuadrados concéntricos.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapizA = new Pen(Color.Blue, 2);
            Pen lapizB = new Pen(Color.Red, 2);
            Pen lapizC = new Pen(Color.Chocolate, 2);
            Pen lapizD = new Pen(Color.DarkGreen, 2);

            int variar, centro;
            variar = 0;
            centro = 380;

            for (var cont = 1; cont <= 30; cont += 1) {
                grafico.DrawLine(lapizA, centro - variar, centro - variar, centro - variar + variar * 2,
centro - variar);
                grafico.DrawLine(lapizB, centro - variar, centro - variar, centro - variar, centro -
variar + variar * 2);
                grafico.DrawLine(lapizC, centro - variar + variar * 2, centro - variar, centro - variar +
variar * 2, centro - variar + variar * 2);
                grafico.DrawLine(lapizD, centro - variar, centro - variar + variar * 2, centro - variar +
variar * 2, centro - variar + variar * 2);
                variar = variar + 10;
            }
        }
    }
}
```

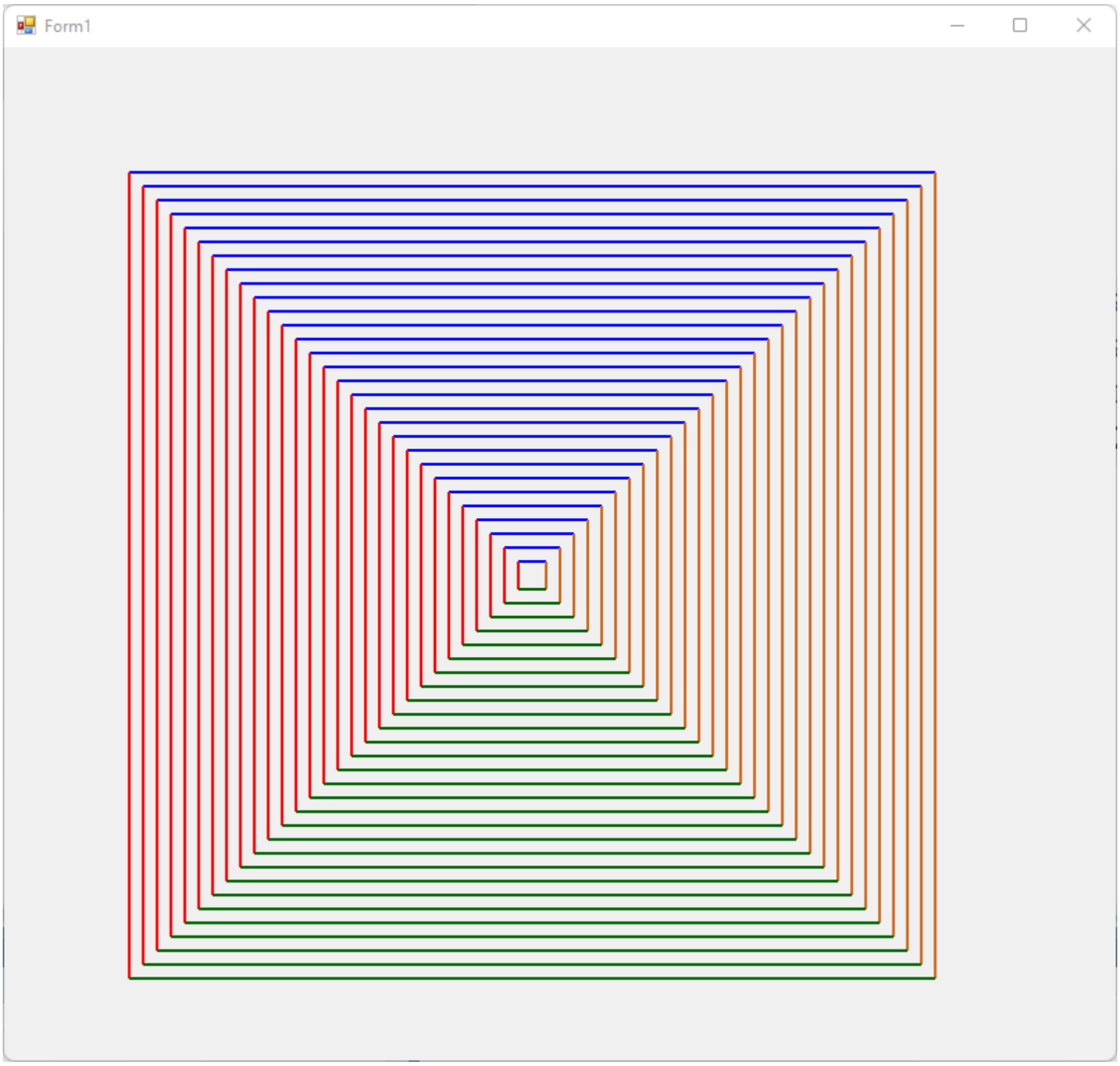


Ilustración 36: Cuadrados concéntricos

Alternar A

02. Ciclos/12. Alternar A.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            int variar = 0;

            for (var cont = 1; cont <= 18; cont += 1) {
                grafico.DrawLine(lapiz, 380 - variar, 380 - variar, 380 + variar, 380 - variar);
                grafico.DrawLine(lapiz, 380 - variar, 380 - variar, 380 - variar, 380 + variar);
                grafico.DrawLine(lapiz, 380 + variar, 380 - variar, 380 + variar, 380 + variar);
                variar += 10;

                grafico.DrawLine(lapiz2, 380 - variar, 380 + variar, 380 + variar, 380 + variar);
                grafico.DrawLine(lapiz2, 380 - variar, 380 - variar, 380 - variar, 380 + variar);
                grafico.DrawLine(lapiz2, 380 + variar, 380 - variar, 380 + variar, 380 + variar);
                variar += 10;
            }
        }
    }
}
```

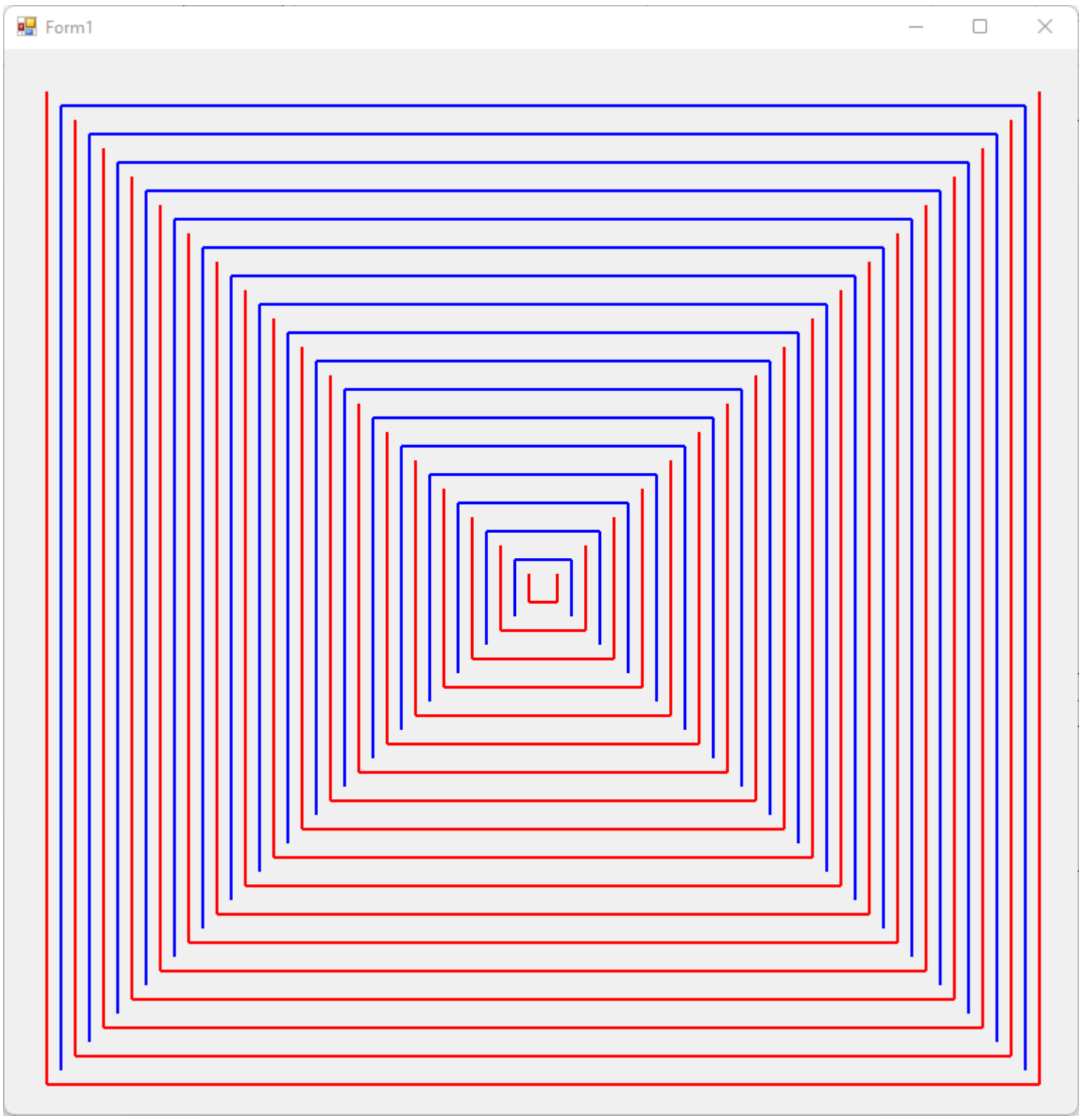


Ilustración 37: Alternando

Alternar B

02. Ciclos/13. Alternar B.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            int variar = 0;

            for (var cont = 1; cont <= 18; cont += 1) {
                grafico.DrawLine(lapiz, 380 - variar, 380 - variar, 380 + variar, 380 - variar);
                grafico.DrawLine(lapiz, 380 - variar, 380 - variar, 380 - variar, 380 + variar);
                variar += 10;

                grafico.DrawLine(lapiz2, 380 - variar, 380 + variar, 380 + variar, 380 + variar);
                grafico.DrawLine(lapiz2, 380 - variar, 380 - variar, 380 - variar, 380 + variar);
                variar += 10;
            }
        }
    }
}
```

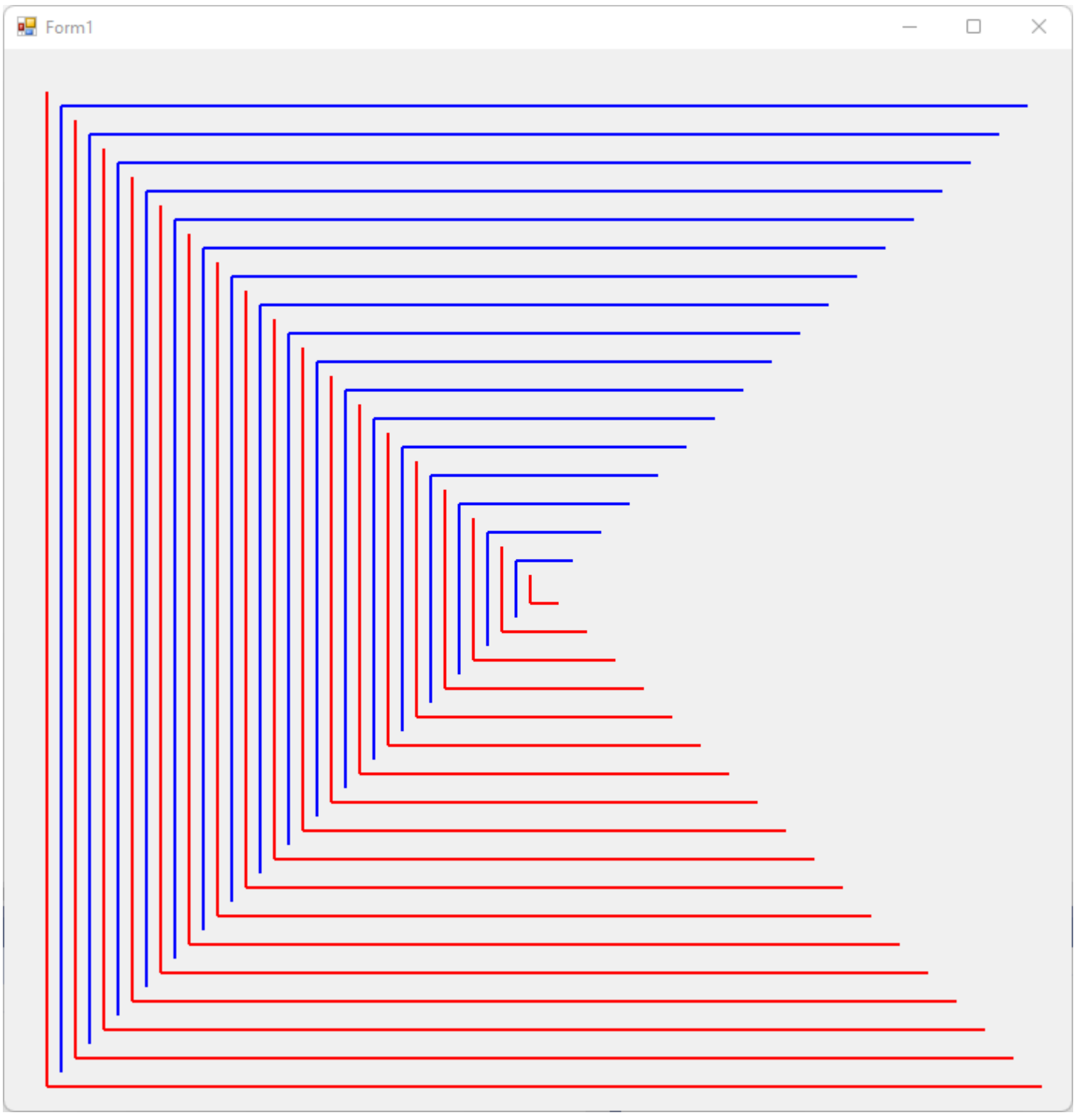


Ilustración 38: Alternar B

Cruce

02. Ciclos/14. Cruce.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);
            Pen lapiz2 = new Pen(Color.Red, 2);
            Pen lapiz3 = new Pen(Color.Green, 2);
            Pen lapiz4 = new Pen(Color.Violet, 2);
            int cont = 0;

            do {
                grafico.DrawLine(lapiz, cont, 600, 600, cont);
                grafico.DrawLine(lapiz2, 1200 - cont, 600, 600, cont);
                grafico.DrawLine(lapiz3, 1200 - cont, 0, 600, 600 - cont);
                grafico.DrawLine(lapiz4, cont, 0, 600, 600 - cont);
                cont += 20;
            }
            while (cont <= 600);
        }
    }
}
```

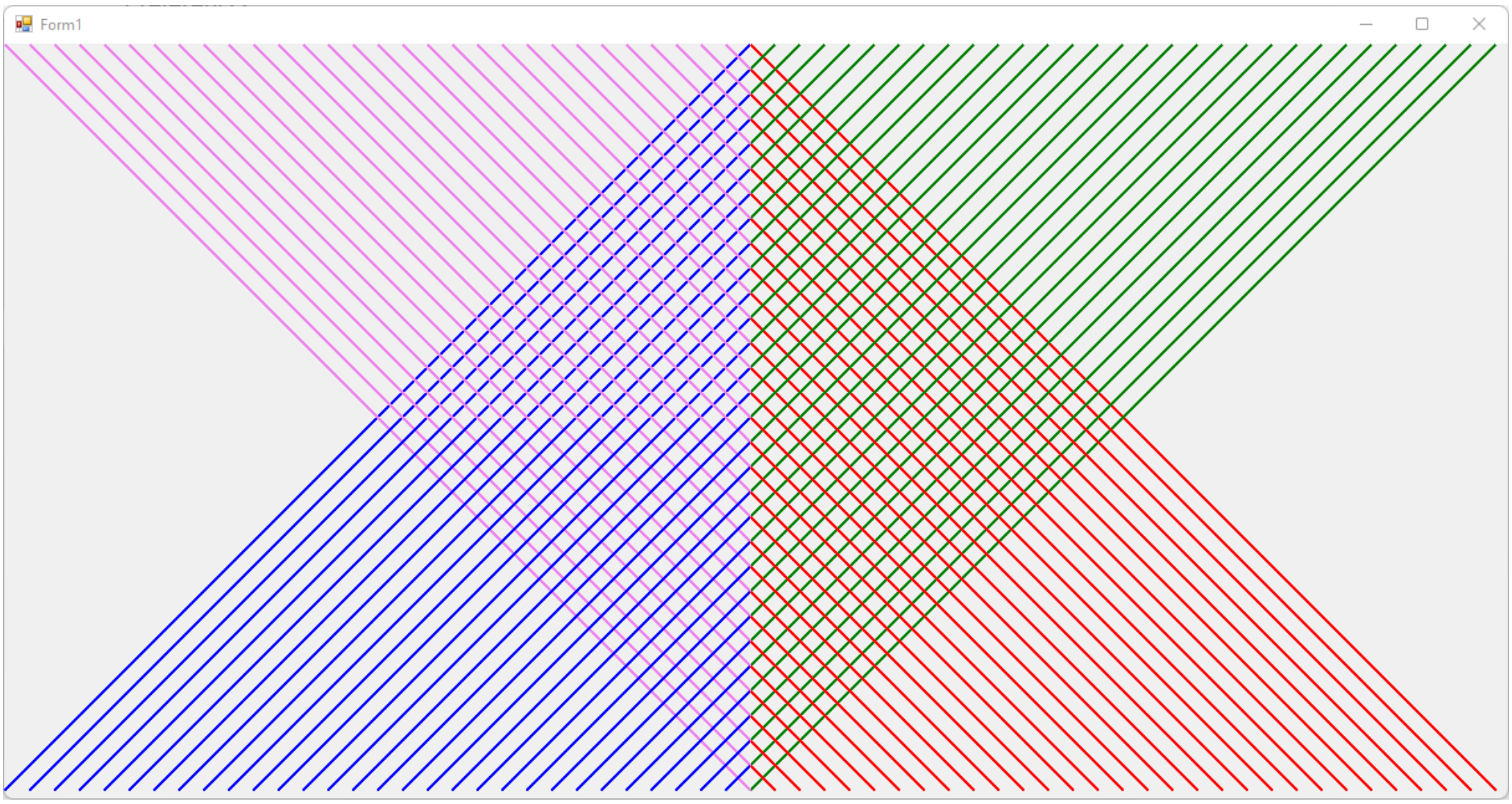


Ilustración 39: Cruce

Celdas

02. Ciclos/15. Celdas.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            // Necesario para los gráficos GDI+
            Graphics grafico = e.Graphics;

            // Lápiz para el color de la línea y con un ancho de 1
            Pen lapiz = new Pen(Color.Blue, 1);

            for (int Fila = 0; Fila <= 10; Fila++) {
                for (int Columna = 0; Columna <= 10; Columna++)
                    grafico.DrawRectangle(lapiz, Fila * 35 + 80, Columna * 35 + 40, 30, 30);
            }
        }
    }
}
```

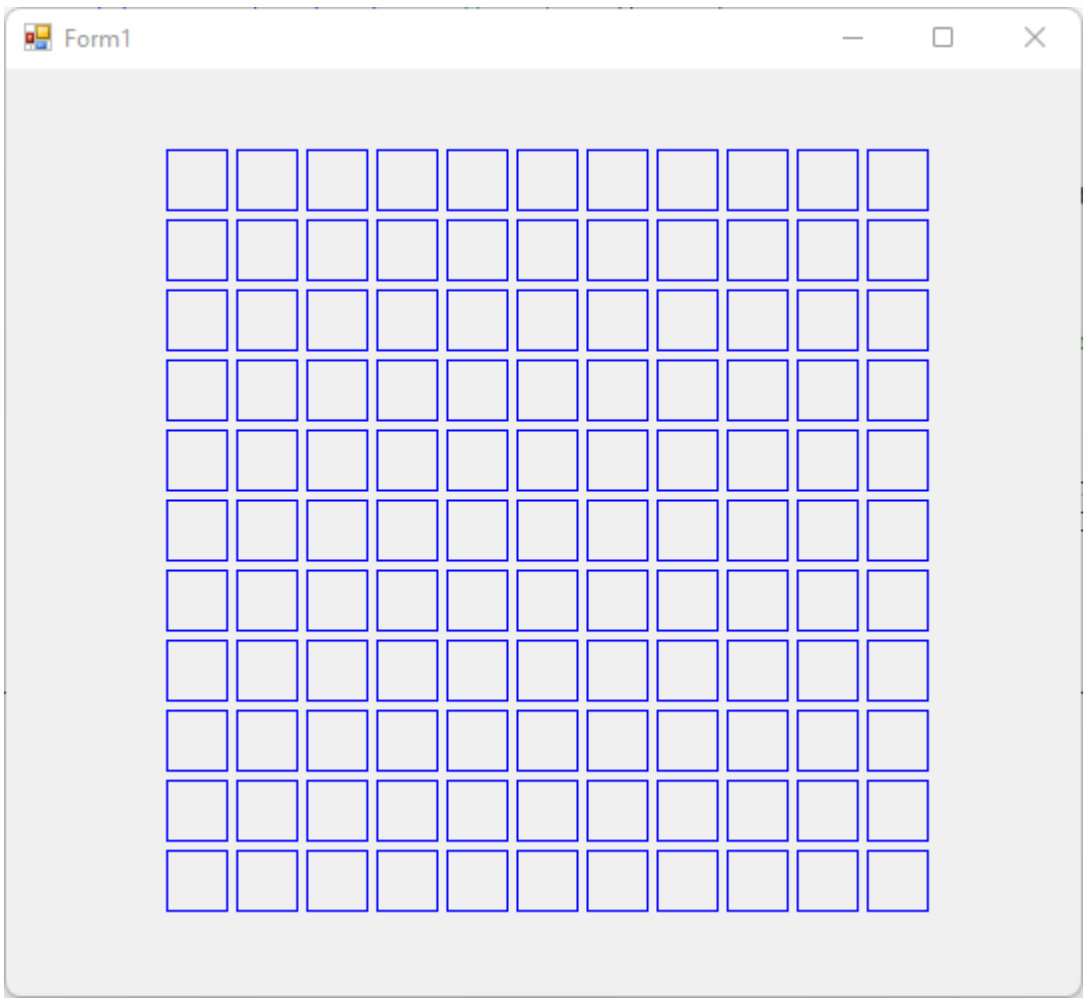


Ilustración 40: Celdas

Simulando curvas

02. Ciclos/16. Simulando Curvas.cs

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics grafico = e.Graphics;

            // Lápiz para el color de la línea y con un ancho de 1
            Pen lapiz = new Pen(Color.Blue, 1);

            // Parte superior
            for (int Posicion = 0; Posicion <= 200; Posicion += 20) {
                // Línea requiere:  Lapiz, X1, Y1, X2, Y2
                grafico.DrawLine(lapiz, 200, Posicion, Posicion + 200, 200);
                grafico.DrawLine(lapiz, 200, Posicion, 200 - Posicion, 200);
            }

            //Parte inferior
            for (int Posicion = 400; Posicion >= 200; Posicion += -20) {
                // Línea requiere:  Lapiz, X1, Y1, X2, Y2
                grafico.DrawLine(lapiz, 200, Posicion, 600 - Posicion, 200);
                grafico.DrawLine(lapiz, 200, Posicion, Posicion - 200, 200);
            }
        }
    }
}
```

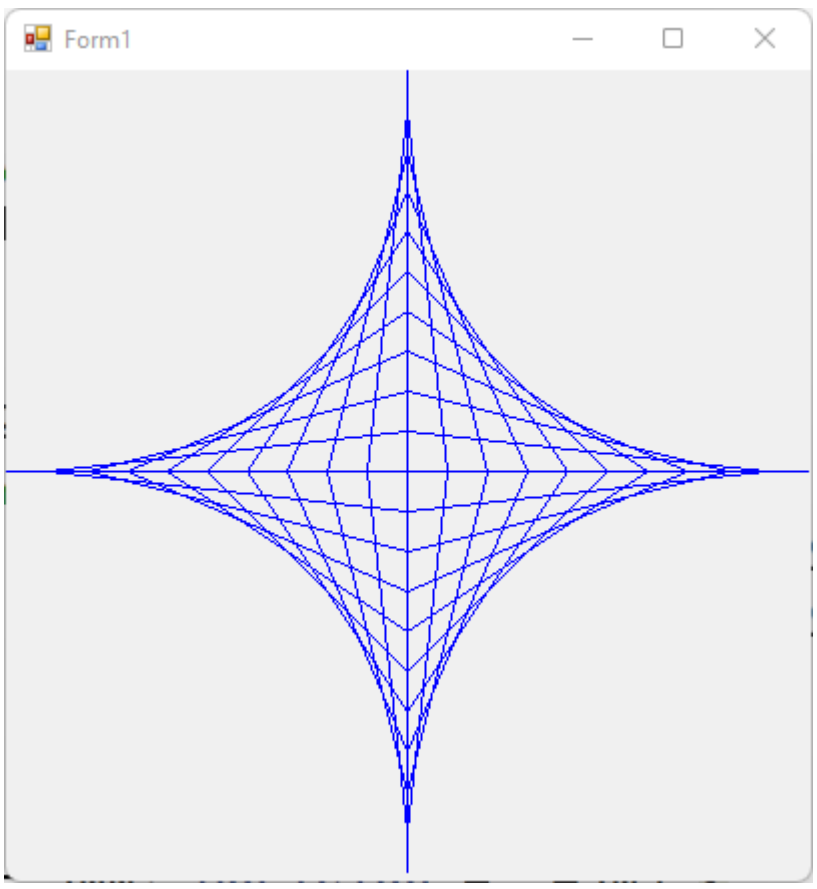


Ilustración 41: Simulando curvas

Animación

Haciendo uso del Paint()

Es necesario agregar el control “Timer” [7] [8] al formulario

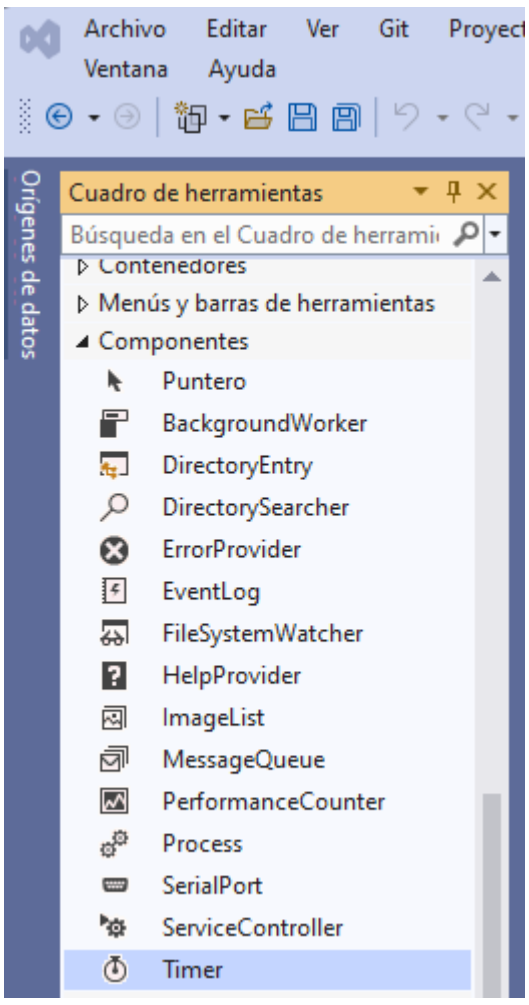


Ilustración 42: Control Timer

Se ajustan las propiedades de este control: (Name) y Enabled que debe estar en **True**

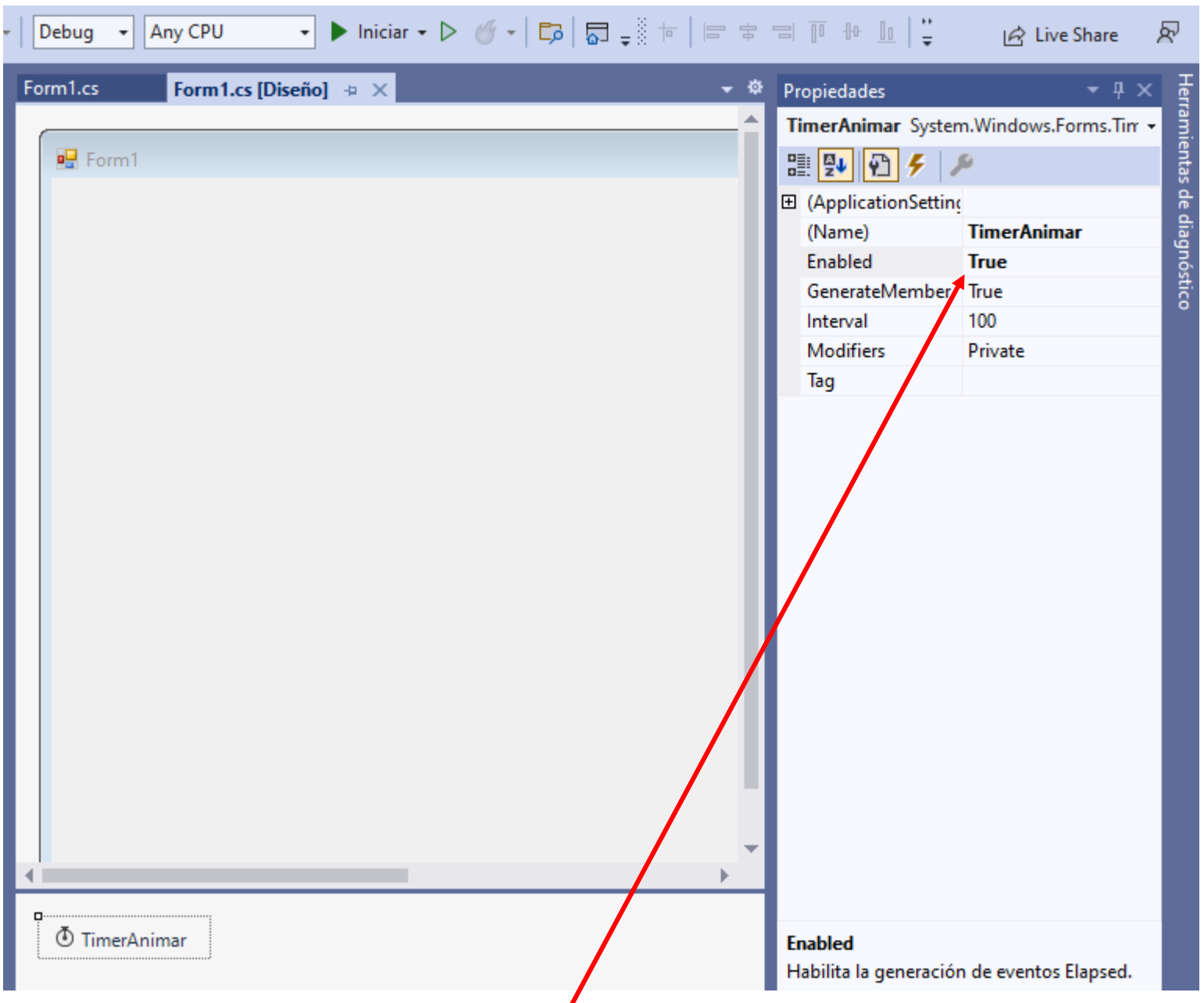


*Ilustración 43: Propiedades del control Timer. Enabled debe estar en **True***

¡OJO!: Hay que poner en **True** la propiedad DoubleBuffered [9] del formulario

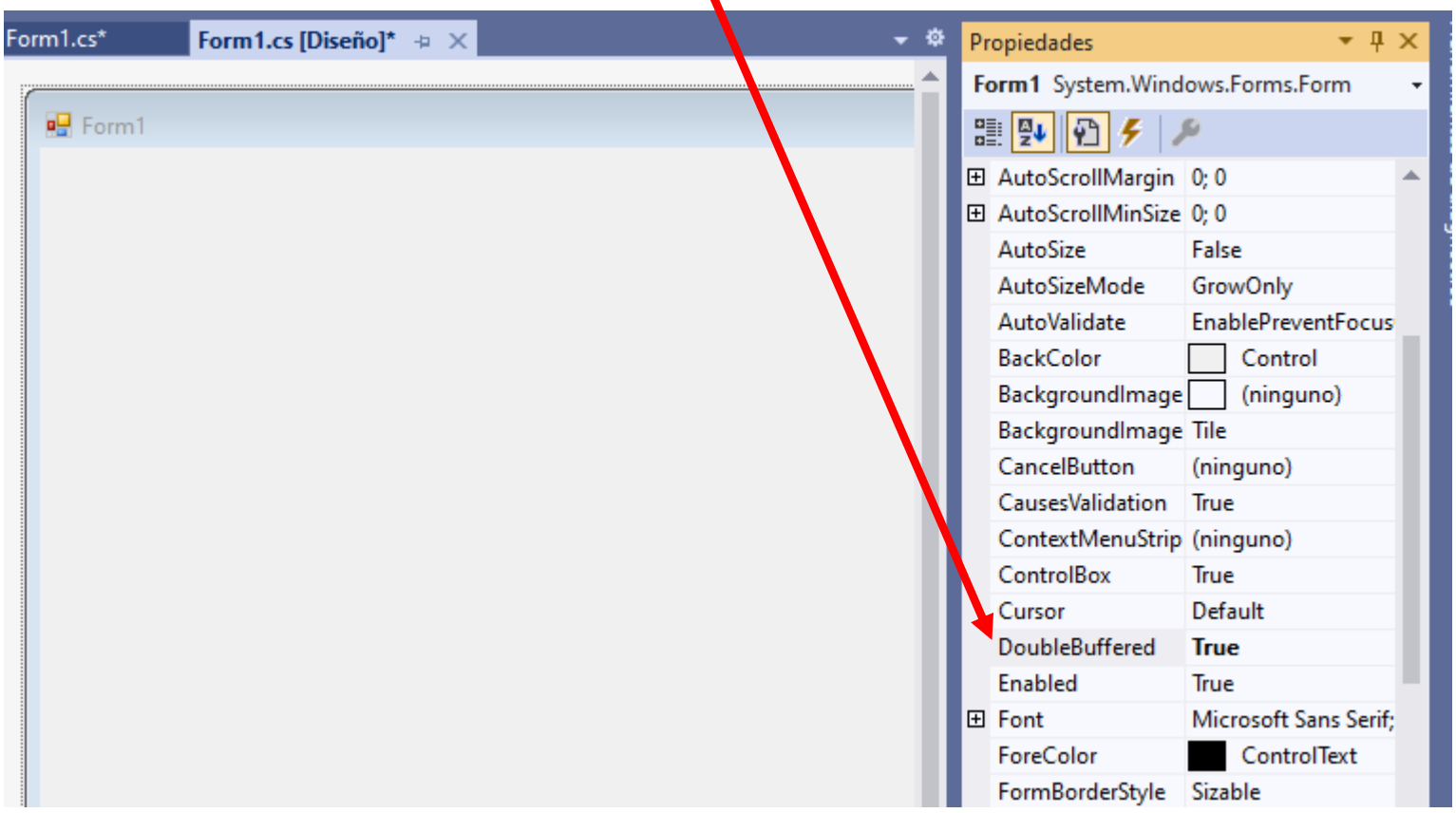


Ilustración 44: Propiedad DoubleBuffered del formulario en True

Ahora se habilita el evento Tick del Timer

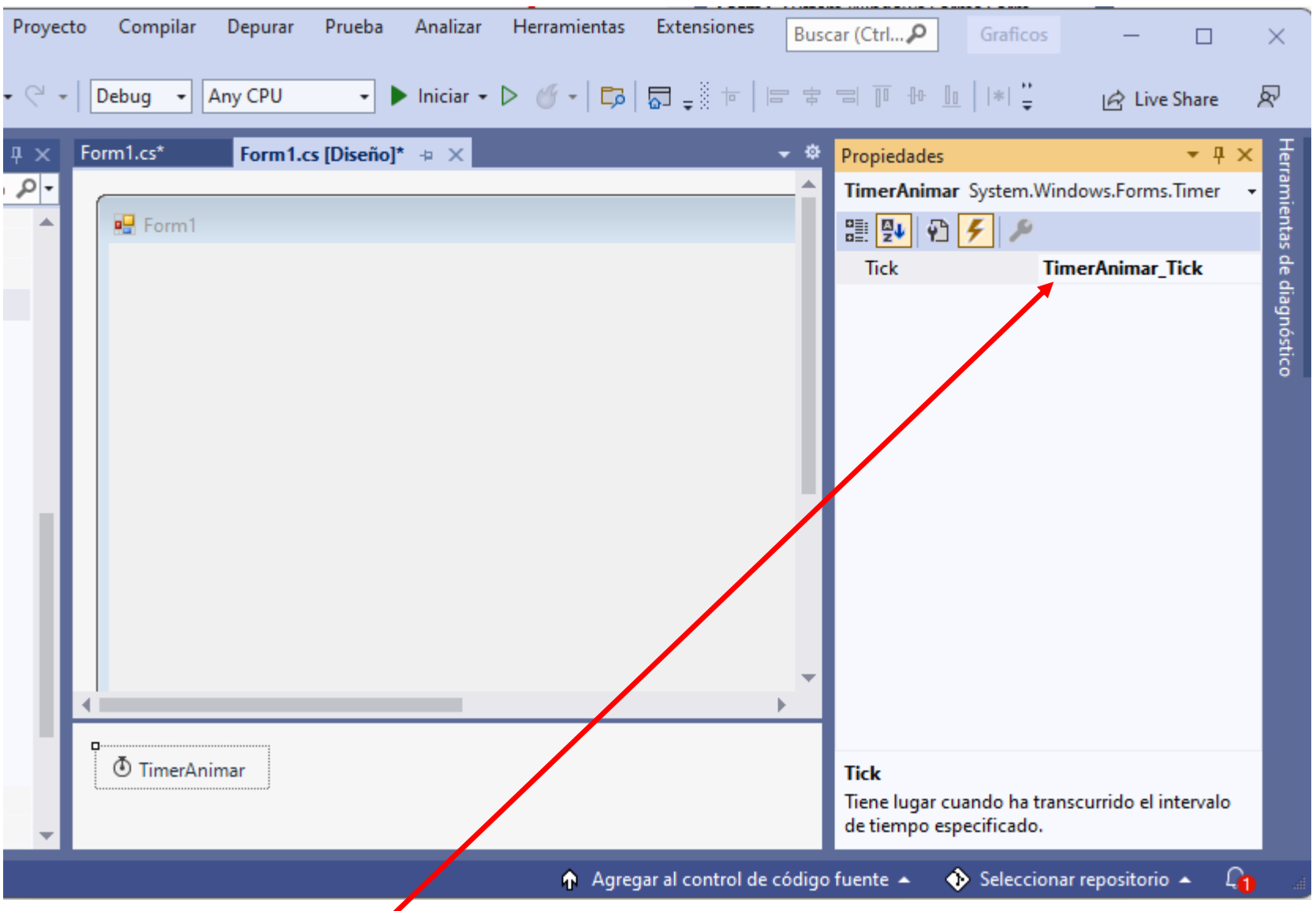


Ilustración 45: Habilitar el evento Tick del Timer

Y ahora el código

03. Animación/01. Inicio.zip

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        int PosX, PosY; //Coordenadas del cuadrado relleno

        public Form1() {
            InitializeComponent();

            //Inicializa las posiciones
            PosX = 10;
            PosY = 20;
        }

        private void TimerAnimar_Tick(object sender, System.EventArgs e) {
            //Por cada tick, incrementa en 10 el valor de la posición X del cuadrado relleno
            PosX += 10;
            Refresh(); //Refresca el formulario y llama a Paint()
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics lienzo = e.Graphics;
            SolidBrush Relleno = new SolidBrush(Color.Chocolate);

            //================================
            //Rectángulo: Xpos, Ypos, ancho, alto
            //================================
            lienzo.FillRectangle(Relleno, PosX, PosY, 40, 40);
        }
    }
}
```

El resultado es que el rectángulo se desplaza por la pantalla de izquierda a derecha

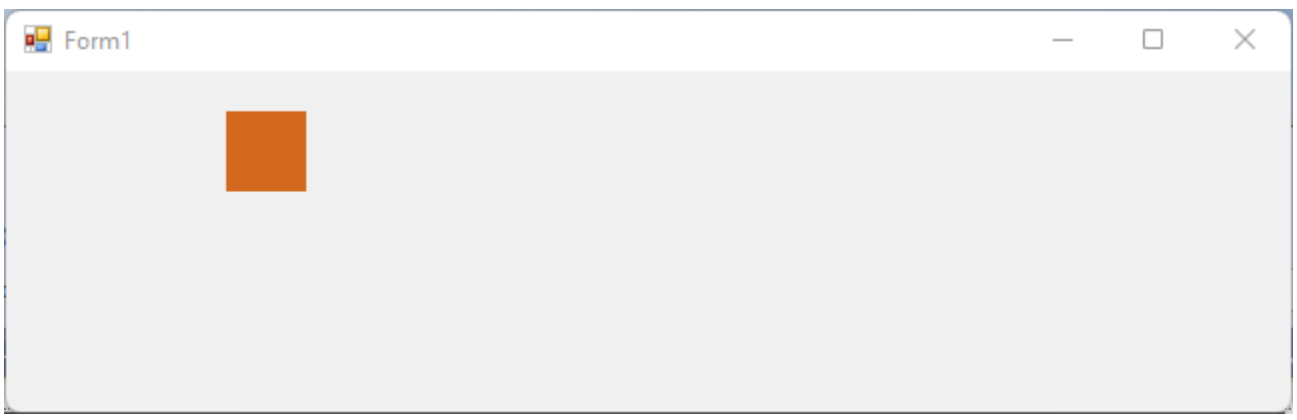


Ilustración 46: Desplazamiento del rectángulo

Ilustración 47: Desplazamiento del rectángulo

¡OJO! La animación no será suave, es una limitante que tiene Winforms [10]. En algunos foros recomiendan cambiar la instrucción Refresh(); por Invalidate(); . Ambas instrucciones redibujan, pero operan de distinta forma:

Invalidate() : Envía el mensaje al sistema operativo para que redibuje, pero no lo hace inmediatamente.

Refresh() : Fuerza a redibujar inmediatamente.

Rebotando en pantalla

El siguiente ejemplo, muestra un cuadrado relleno rebotando en las paredes de la ventana.

03. Animación/02. Rebote.zip

```
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        int PosX, PosY; //Coordenadas del cuadrado relleno
        int Tamano; //Tamaño del lado del cuadrado
        int IncrementoX, IncrementoY; //Desplazamiento del cuadrado relleno

        public Form1() {
            InitializeComponent();

            //Inicializa la posición y tamaño del cuadrado relleno
            PosX = 10;
            PosY = 20;
            Tamano = 40;

            //Velocidad con que se desplaza el cuadrado relleno
            IncrementoX = 5;
            IncrementoY = 5;
        }

        private void TimerAnimar_Tick(object sender, System.EventArgs e) {
            Logica(); //Lógica de la animación
            Refresh(); //Visual de la animación
        }

        void Logica() {
            //Si colisiona con alguna pared cambia el incremento
            if (PosX + Tamano > this.ClientSize.Width || PosX < 0) IncrementoX *= -1;
            if (PosY + Tamano > this.ClientSize.Height || PosY < 0) IncrementoY *= -1;

            //Cambia la posición de X y Y
            PosX += IncrementoX;
            PosY += IncrementoY;
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Fondo de la ventana
            e.Graphics.FillRectangle(Brushes.Black, new Rectangle(0, 0, this.Width, this.Height));

            //Gráfico a animar
            e.Graphics.FillRectangle(Brushes.Chocolate, PosX, PosY, Tamano, Tamano);
        }
    }
}
```

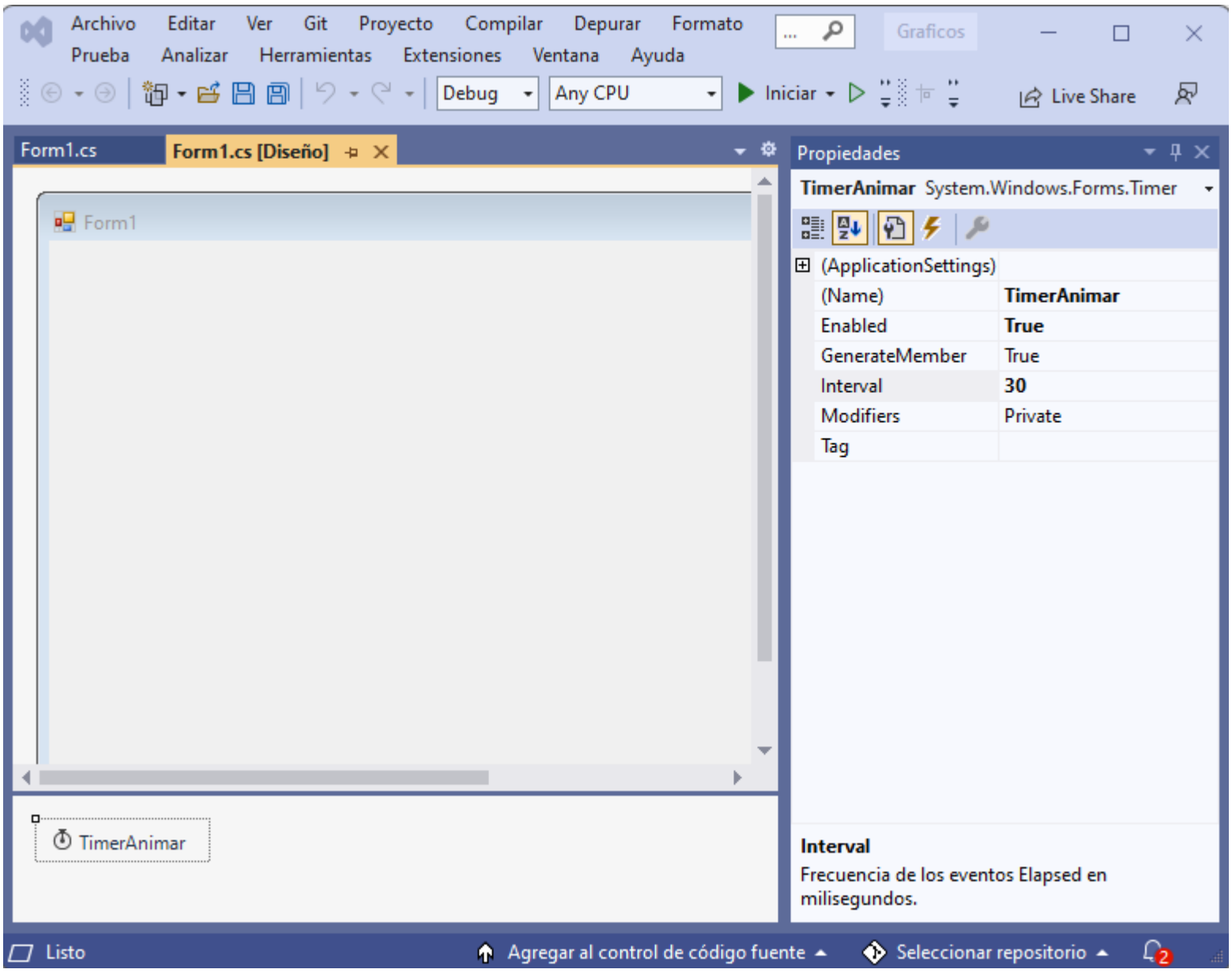


Ilustración 48: Control Timer

Ilustración 49: Ejemplo de su ejecución

Lógica y representación

Es muy recomendable que la parte lógica de la aplicación gráfica se concentre sobre un arreglo bidimensional y la parte visual es reflejar lo que sucede en ese arreglo bidimensional en pantalla.

Reescribimos el programa de rebote anterior, haciendo uso de un arreglo bidimensional.

03. Animación/03. Lógica.zip

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        int[,] Tablero; //Dónde ocurre realmente la acción
        int PosX, PosY; //Coordenadas del cuadrado relleno
        int IncrementoX, IncrementoY; //Desplazamiento del cuadrado relleno

        public Form1() {
            InitializeComponent();

            //Inicializa el tablero
            Tablero = new int[30, 30];

            //Inicializa la posición del cuadrado relleno
            Random azar = new Random();
            PosX = azar.Next(0, Tablero.GetLength(0));
            PosY = azar.Next(0, Tablero.GetLength(1));

            //Desplaza el cuadrado relleno
            IncrementoX = 1;
            IncrementoY = 1;
        }

        private void TimerAnimar_Tick(object sender, System.EventArgs e) {
            Logica(); //Lógica de la animación
            Refresh(); //Visual de la animación
        }

        void Logica() {
            //Borra la posición anterior
            Tablero[PosX, PosY] = 0;

            //Si colisiona con alguna pared cambia el incremento
            if (PosX + IncrementoX >= Tablero.GetLength(0) || PosX + IncrementoX < 0) IncrementoX *= -1;
            if (PosY + IncrementoY >= Tablero.GetLength(1) || PosY + IncrementoY < 0) IncrementoY *= -1;

            //Cambia la posición de X y Y
            PosX += IncrementoX;
            PosY += IncrementoY;

            //Nueva posición
            Tablero[PosX, PosY] = 1;
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Tamaño de cada celda
            int tamanoX = this.ClientSize.Width / Tablero.GetLength(0);
            int tamanoY = this.ClientSize.Height / Tablero.GetLength(1);

            //Fondo de la ventana
            e.Graphics.FillRectangle(Brushes.Black, new Rectangle(0, 0, this.Width, this.Height));

            //Dibuja la malla y la posición del rectángulo relleno
            for (int Fila = 0; Fila < Tablero.GetLength(0); Fila++) {
                for (int Columna = 0; Columna < Tablero.GetLength(1); Columna++)
                    if (Tablero[Fila, Columna] == 0)
                        e.Graphics.DrawRectangle(Pens.Blue, Fila * tamanoX, Columna * tamanoY, tamanoX,
tamanoY);
                    else
                        e.Graphics.FillRectangle(Brushes.Chocolate, Fila * tamanoX, Columna * tamanoY,
tamanoX, tamanoY);
                }
            }
        }
}
```

Así ejecuta

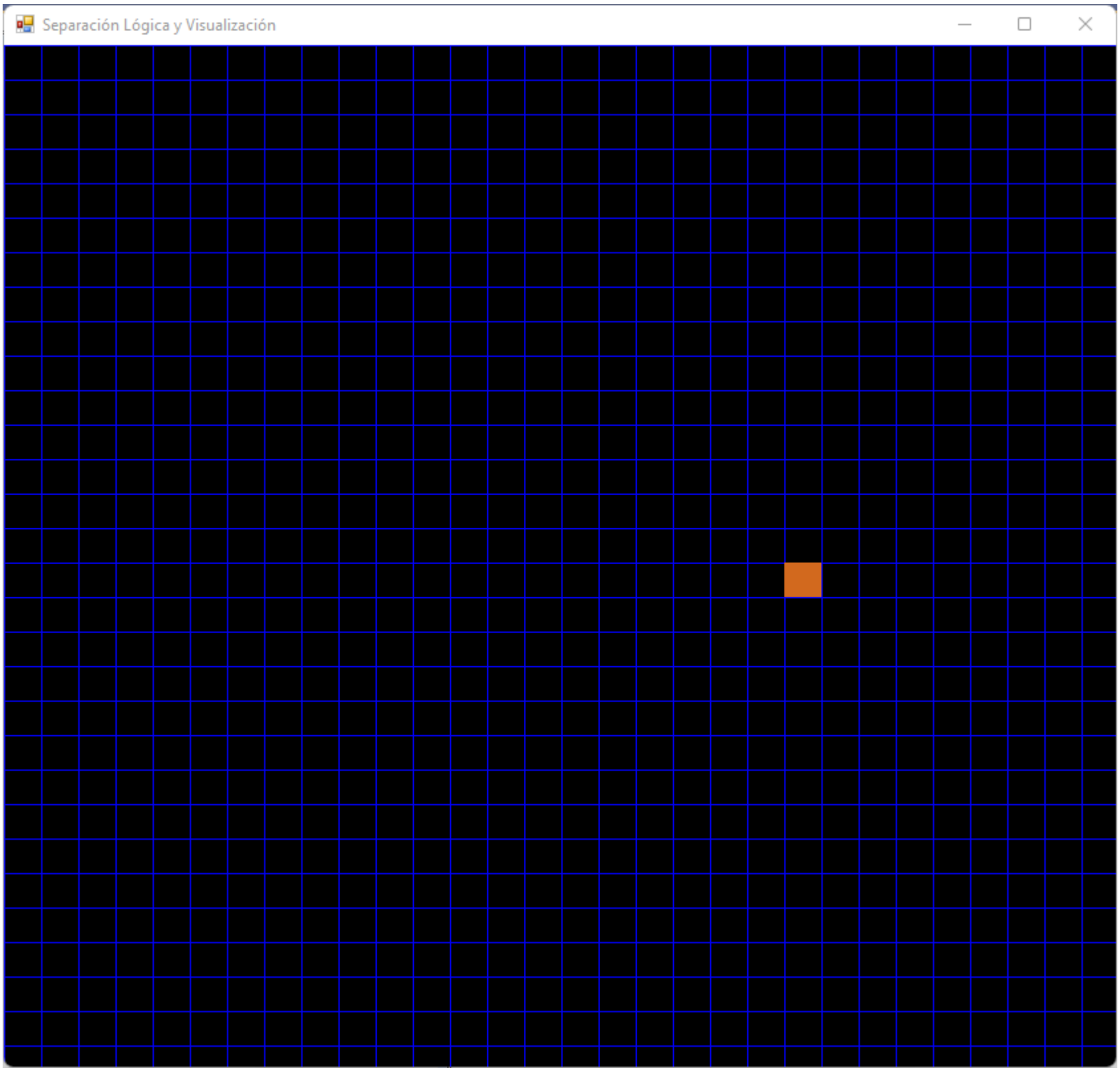


Ilustración 50: Ejemplo de ejecución

Es mucho mejor así, porque hay pleno control del tamaño del tablero, cómo se desplaza, dónde rebota. En el código de ejemplo, si se cambia el tamaño de la ventana, el programa reacciona al nuevo tamaño.

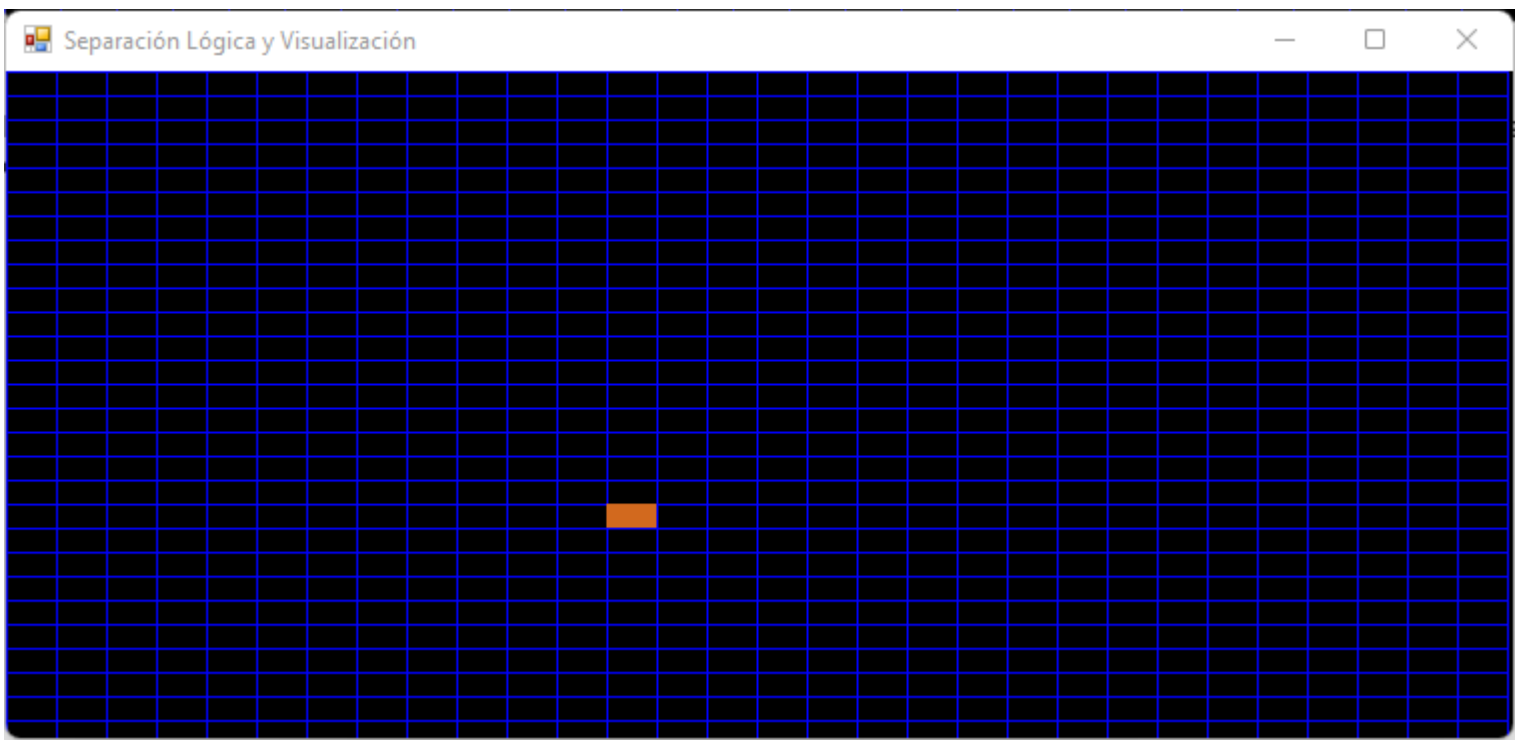


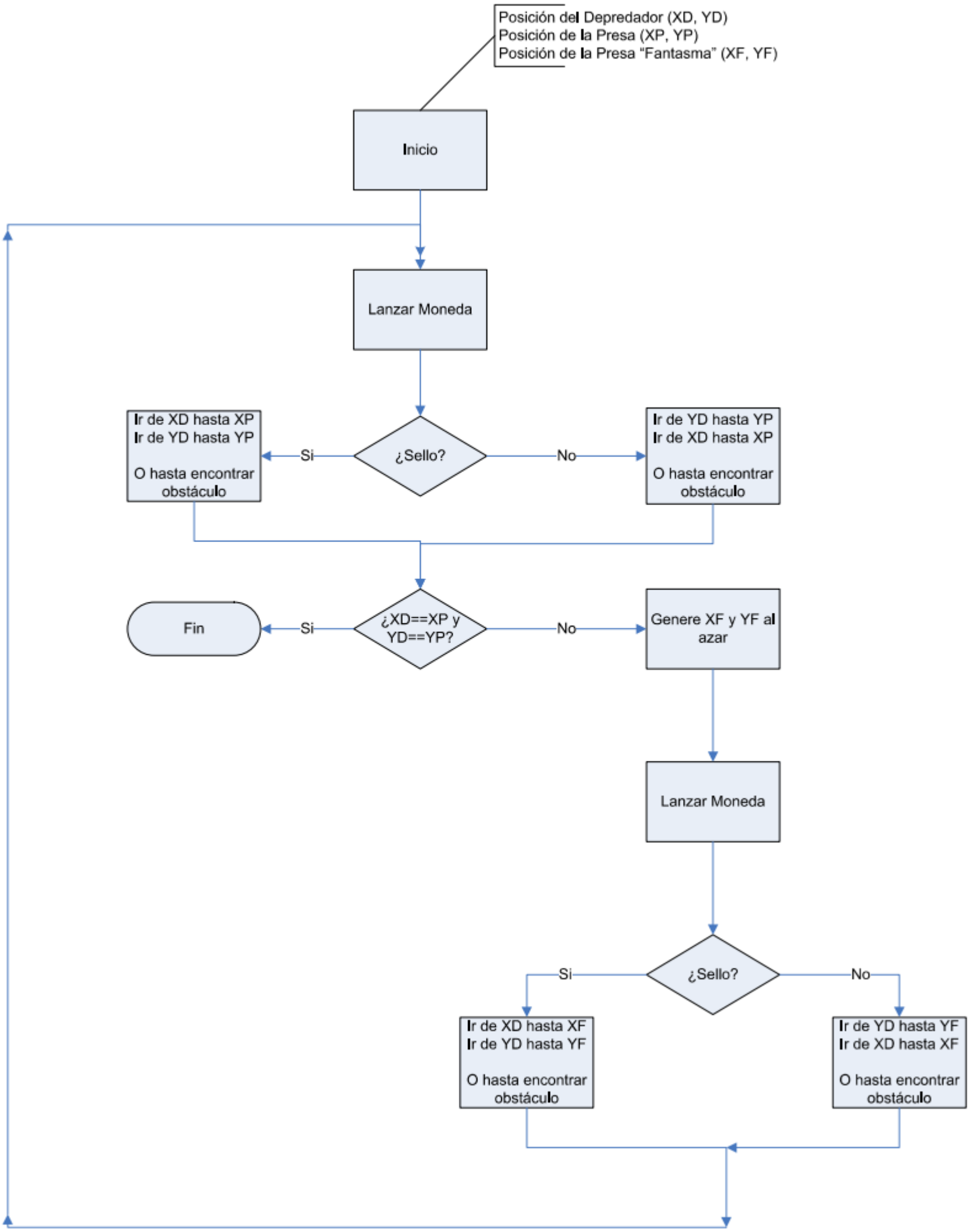
Ilustración 51: Reacción al cambio de tamaño de la ventana

Depredador y Presa

Dado un arreglo bidimensional de 30*30, se ubica en una casilla (seleccionada al azar) un depredador y en otra casilla (seleccionada al azar) una presa. El objetivo es hacer que el depredador salte de casilla en casilla, este salto puede ser horizontal o vertical o diagonal hasta alcanzar la casilla de la presa. El tamaño del salto del depredador es de una(1) casilla cada vez. La presa se mantiene inmóvil.

Se deben agregar obstáculos al azar a este tablero bidimensional (casillas en las cuales el depredador **NO** puede saltar).

Un algoritmo para que el depredador alcance la presa:



```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Constantes para calificar cada celda del tablero
        private const int ESCAMINO = 0;
        private const int ESOBSTACULO = 1;
        private const int ESDEPREDADOR = 2;
        private const int ESPRESA = 3;

        int[,] Tablero; //Dónde ocurre realmente la acción
        int PosDepredaX, PosDepredaY; //Coordenadas del depredador
        int DepredaX, DepredaY; //Desplazamiento del depredador
        int PresaX, PresaY; //Coordenadas de la presa
        int FantasmaX, FantasmaY; //Coordenadas de la presa fantasma para desatorar
        bool BuscaFantasma; //Alterna entre buscar la presa real o la presa fantasma
        Random azar; //Único generador de números aleatorios.

        public Form1() {
            InitializeComponent();
            azar = new Random();
            IniciarParametros();
        }

    }

    private void mnuReiniciar_Click(object sender, EventArgs e) {
        IniciarParametros();
    }

    public void IniciarParametros() {
        //Inicializa el tablero.
        Tablero = new int[30, 30];

        //Habrán obstáculos en el 20%
        int Obstaculos = Tablero.GetLength(0) * Tablero.GetLength(1) * 20 / 100;
        for (int cont = 1; cont <= Obstaculos; cont++) {
            int obstaculoX, obstaculoY;
            do {
                obstaculoX = azar.Next(0, Tablero.GetLength(0));
                obstaculoY = azar.Next(0, Tablero.GetLength(1));
            } while (Tablero[obstaculoX, obstaculoY] != 0);
            Tablero[obstaculoX, obstaculoY] = ESOBSTACULO;
        }

        //Inicializa la posición del depredador
        do {
            PosDepredaX = azar.Next(0, Tablero.GetLength(0));
            PosDepredaY = azar.Next(0, Tablero.GetLength(1));
        } while (Tablero[PosDepredaX, PosDepredaY] != 0);
        Tablero[PosDepredaX, PosDepredaY] = ESDEPREDADOR;

        //Inicializa la posición de la presa
        do {
            PresaX = azar.Next(0, Tablero.GetLength(0));
            PresaY = azar.Next(0, Tablero.GetLength(1));
        } while (Tablero[PresaX, PresaY] != 0);
        Tablero[PresaX, PresaY] = ESPRESA;

        //Desplazamiento del depredador
        DepredaX = 1;
        DepredaY = 1;

        TimerAnimar.Start();
    }


    private void TimerAnimar_Tick(object sender, System.EventArgs e) {
        Logica(); //Lógica de la animación
        Refresh(); //Visual de la animación
    }

    public void Logica() {
        Tablero[PosDepredaX, PosDepredaY] = ESCAMINO;

        if (BuscaFantasma == false) {
            //Esta buscando la presa
```

```
                if (PosDepredaX > PresaX) DepredaX = -1;
                else if (PosDepredaX < PresaX) DepredaX = 1;
                else DepredaX = 0;

                if (PosDepredaY > PresaY) DepredaY = -1;
                else if (PosDepredaY < PresaY) DepredaY = 1;
                else DepredaY = 0;

                if (PosDepredaX == PresaX && PosDepredaY == PresaY) {  //Verifica si ya llegó a la presa
                    TimerAnimar.Stop();
                    MessageBox.Show("El depredador alcanzó a la presa", "Depredador - Presa",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                    return;
                }
                //Si no, verifica si puede desplazarse a la nueva ubicación
                else if (Tablero[PosDepredaX + DepredaX, PosDepredaY + DepredaY] == ESCAMINO ||
Tablero[PosDepredaX + DepredaX, PosDepredaY + DepredaY] == ESPRESA) {
                    PosDepredaX += DepredaX;
                    PosDepredaY += DepredaY;
                }
                else { //Si no, entonces está atorado con los obstáculos. Luego genera coordenadas de
presa fantasma
                    do {
                        FantasmaX = azar.Next(0, Tablero.GetLength(0));
                        FantasmaY = azar.Next(0, Tablero.GetLength(1));
                    } while (Tablero[FantasmaX, FantasmaY] != ESCAMINO);
                    BuscaFantasma = true;
                }
            }
            else { //Está buscando la presa fantasma
                if (PosDepredaX > FantasmaX) DepredaX = -1;
                else if (PosDepredaX < FantasmaX) DepredaX = 1;
                else DepredaX = 0;

                if (PosDepredaY > FantasmaY) DepredaY = -1;
                else if (PosDepredaY < FantasmaY) DepredaY = 1;
                else DepredaY = 0;

                //Si ha dado con la presa fantasma o se queda atorado, deja de perseguir la presa fantasma
                if (PosDepredaX == FantasmaX && PosDepredaY == FantasmaY || Tablero[PosDepredaX +
DepredaX, PosDepredaY + DepredaY] == ESOBSTACULO)
                    BuscaFantasma = false;
                else {
                        PosDepredaX += DepredaX;
                        PosDepredaY += DepredaY;
                }
            }

            Tablero[PosDepredaX, PosDepredaY] = ESDEPREDADOR;
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Tamaño de cada celda
            int tamanoX = 500 / Tablero.GetLength(0);
            int tamanoY = 500 / Tablero.GetLength(1);
            int desplaza = 50;

            //Dibuja el arreglo bidimensional
            for (int Fila = 0; Fila < Tablero.GetLength(0); Fila++) {
                for (int Columna = 0; Columna < Tablero.GetLength(1); Columna++) {
                    int UbicaX = Fila * tamanoX + desplaza;
                    int UbicaY = Columna * tamanoY + desplaza;
                    switch (Tablero[Fila, Columna]) {
                        case ESCAMINO: e.Graphics.DrawRectangle(Pens.Blue, UbicaX, UbicaY, tamanoX,
tamanoY); break;
                        case ESOBSTACULO: e.Graphics.FillRectangle(Brushes.Black, UbicaX, UbicaY, tamanoX,
tamanoY); break;
                        case ESDEPREDADOR: e.Graphics.FillRectangle(Brushes.Red, UbicaX, UbicaY, tamanoX,
tamanoY); break;
                        case ESPRESA: e.Graphics.FillRectangle(Brushes.Blue, UbicaX, UbicaY, tamanoX,
tamanoY); break;
                    }
                }
            }
        }
    }
}
```

Y así ejecuta:

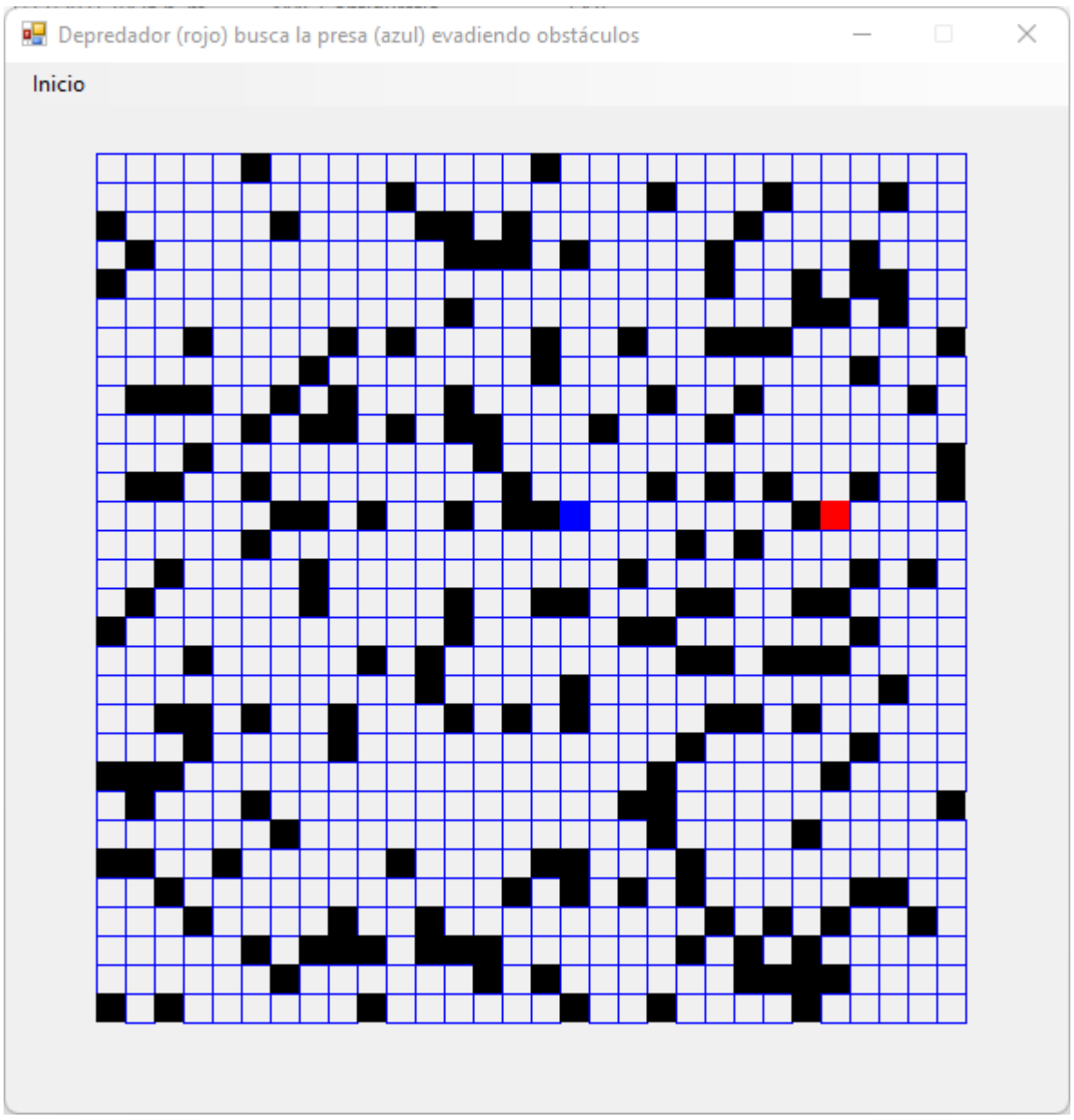


Ilustración 52: Depredador en busca de la presa

¡OJO! El algoritmo de búsqueda de la presa es bastante ineficiente en su cometido: El depredador puede tardar un tiempo considerable en encontrar la presa, además este algoritmo no valida que exista un camino viable por lo que se puede quedar operando en forma infinita buscando como llegar a la presa.

Cuando el depredador alcanza la presa sale este mensaje:

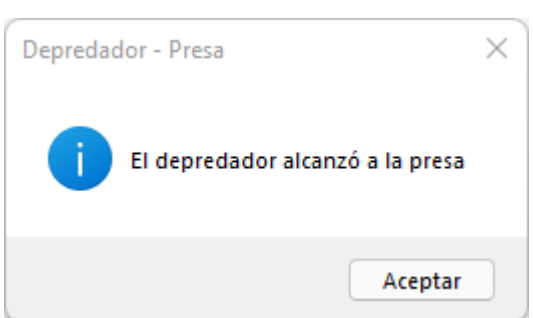


Ilustración 53: Finaliza con el depredador alcanzando a la presa

Luego se puede reiniciar el juego en el menú.

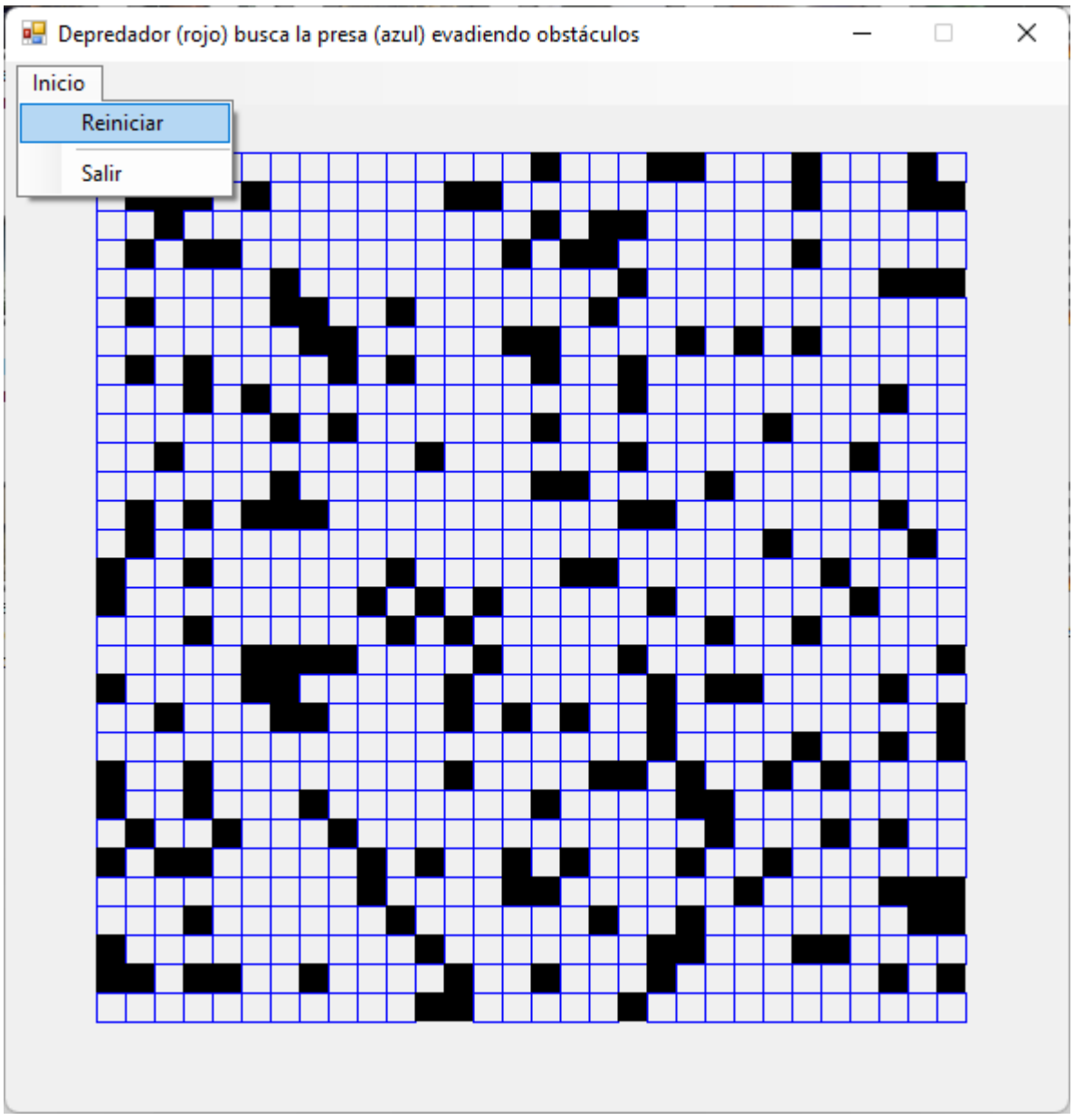


Ilustración 54: Reiniciar la simulación

El juego de la vida

Este juego https://es.wikipedia.org/wiki/Juego_de_la_vida inventado por John Horton Conway, ha llamado mucho la atención por como dada una serie de unas reglas muy sencillas, genera patrones complejos.

03. Animación/05. JuevoVida.zip

```
//El juego de la vida
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Constantes para calificar cada celda del tablero
        private const int INACTIVA = 0;
        private const int ACTIVA = 1;

        int[,] Tablero; //Dónde ocurre realmente la acción
        Random azar; //Único generador de números aleatorios.

        public Form1() {
            InitializeComponent();
            azar = new Random();
            IniciarParametros();
        }

    private void mnuReiniciar_Click(object sender, EventArgs e) {
        IniciarParametros();
    }

    public void IniciarParametros() {
        //Inicializa el tablero.
        Tablero = new int[30, 30];

        //Habrán celdas activas en el 20%
        int Activos = Tablero.GetLength(0) * Tablero.GetLength(1) * 20 / 100;
        for (int cont = 1; cont <= Activos; cont++) {
            int posX, posY;
            do {
                posX = azar.Next(0, Tablero.GetLength(0));
                posY = azar.Next(0, Tablero.GetLength(1));
            } while (Tablero[posX, posY] != 0);
            Tablero[posX, posY] = ACTIVA;
        }

        TimerAnimar.Start();
    }


    private void TimerAnimar_Tick(object sender, System.EventArgs e) {
        Logica(); //Lógica de la animación
        Refresh(); //Visual de la animación
    }

    public void Logica() {
        //Copia el tablero a uno temporal
        int[,] TableroTmp = Tablero.Clone() as int[,];

        //Va de celda en celda
        for (int posX = 0; posX < Tablero.GetLength(0); posX++) {
            for (int posY = 0; posY < Tablero.GetLength(1); posY++) {

                //Determina el número de vecinos activos
                int BordeXizq = posX - 1 < 0 ? Tablero.GetLength(0) - 1 : posX - 1;
                int BordeYarr = posY - 1 < 0 ? Tablero.GetLength(1) - 1 : posY - 1;

                int BordeXder = posX + 1 >= Tablero.GetLength(0) ? 0 : posX + 1;
                int BordeYaba = posY + 1 >= Tablero.GetLength(1) ? 0 : posY + 1;

                int vecinosActivos = Tablero[BordeXizq, BordeYarr];
                vecinosActivos += Tablero[posX, BordeYarr];
                vecinosActivos += Tablero[BordeXder, BordeYarr];

                vecinosActivos += Tablero[BordeXizq, posY];
                vecinosActivos += Tablero[BordeXder, posY];

                vecinosActivos += Tablero[BordeXizq, BordeYaba];
                vecinosActivos += Tablero[posX, BordeYaba];
```

```
                    vecinosActivos += Tablero[BordeXder, BordeYaba];

                    //Los cambios se registran en el tablero temporal

                    //Si la celda está inactiva y tiene 3 celdas activas alrededor, entonces la celda se
activa
                    if (Tablero[posX, posY]==INACTIVA && vecinosActivos == 3) TableroTmp[posX, posY] =
ACTIVA;

                    //Si la celda está activa y tiene menos de 2 celdas o más de 3 celdas, entonces la
celda se inactiva
                    if (Tablero[posX, posY] == ACTIVA && (vecinosActivos < 2 || vecinosActivos > 3))
TableroTmp[posX, posY] = INACTIVA;
                }
            }

            //El cambio en el tablero temporal se copia sobre el tablero
            Tablero = TableroTmp.Clone() as int[,];
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Tamaño de cada celda
            int tamanoX = 500 / Tablero.GetLength(0);
            int tamanoY = 500 / Tablero.GetLength(1);
            int desplaza = 50;

            //Dibuja el arreglo bidimensional
            for (int Fila = 0; Fila < Tablero.GetLength(0); Fila++) {
                for (int Columna = 0; Columna < Tablero.GetLength(1); Columna++) {
                    int UbicaX = Fila * tamanoX + desplaza;
                    int UbicaY = Columna * tamanoY + desplaza;
                    switch (Tablero[Fila, Columna]) {
                        case INACTIVA: e.Graphics.DrawRectangle(Pens.Blue, UbicaX, UbicaY, tamanoX,
tamanoY); break;
                        case ACTIVA: e.Graphics.FillRectangle(Brushes.Black, UbicaX, UbicaY, tamanoX,
tamanoY); break;
                    }
                }
            }
        }
    }
}
```

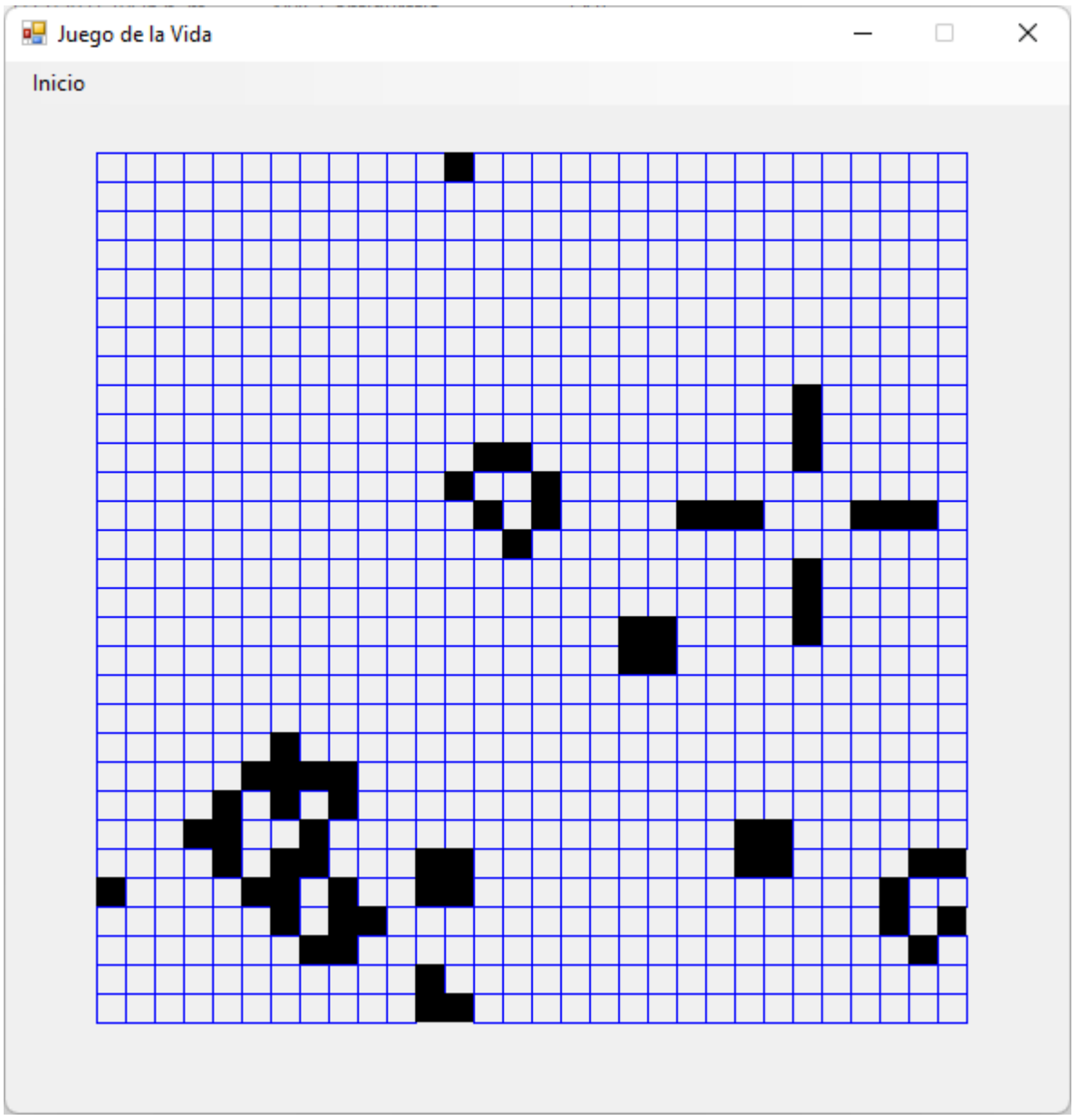


Ilustración 55: Ejecución del programa de Juego de la Vida

Detectar el clic del ratón

¿Cómo saber en qué celda hizo clic el usuario y cambiar el estado de esa celda?

03. Animación/06. Clic del ratón.zip

```
//Activar o desactivar celdas
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Constantes para calificar cada celda del tablero
        private const int INACTIVA = 0;
        private const int ACTIVA = 1;

        int[,] Tablero; //Dónde ocurre realmente la acción

        //Tamaño de cada celda
        int tamanoX, tamanoY, desplaza;

        public Form1() {
            InitializeComponent();
            IniciarParametros();
        }

        private void mnuReiniciar_Click(object sender, EventArgs e) {
            IniciarParametros();
        }

        public void IniciarParametros() {
            Tablero = new int[30, 30]; //Inicializa el tablero.

            //Tamaño de cada celda
            tamanoX = 500 / Tablero.GetLength(0);
            tamanoY = 500 / Tablero.GetLength(1);
            desplaza = 50;

            TimerAnimar.Start();
        }


        private void TimerAnimar_Tick(object sender, System.EventArgs e) {
            Refresh(); //Visual de la animación
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Dibuja el arreglo bidimensional
            for (int Fila = 0; Fila < Tablero.GetLength(0); Fila++) {
                for (int Columna = 0; Columna < Tablero.GetLength(1); Columna++) {
                    int UbicaX = Fila * tamanoX + desplaza;
                    int UbicaY = Columna * tamanoY + desplaza;
                    switch (Tablero[Fila, Columna]) {
                        case INACTIVA: e.Graphics.DrawRectangle(Pens.Blue, UbicaX, UbicaY, tamanoX,
tamanoY); break;
                        case ACTIVA: e.Graphics.FillRectangle(Brushes.Black, UbicaX, UbicaY, tamanoX,
tamanoY); break;
                    }
                }
            }
        }

        //Cuando da clic sobre una celda, la activa o la desactiva
        private void Form1_MouseClick(object sender, MouseEventArgs e) {
            int Fila = (e.X - desplaza) / tamanoX;
            int Columna = (e.Y - desplaza) / tamanoY;

            if (Fila >= 0 && Columna >= 0 && Fila < Tablero.GetLength(0) && Columna <
Tablero.GetLength(1)) {
                switch (Tablero[Fila, Columna]) {
                    case INACTIVA: Tablero[Fila, Columna] = ACTIVA; break;
                    case ACTIVA: Tablero[Fila, Columna] = INACTIVA; break;
                }
            }
        }
    }
}
```

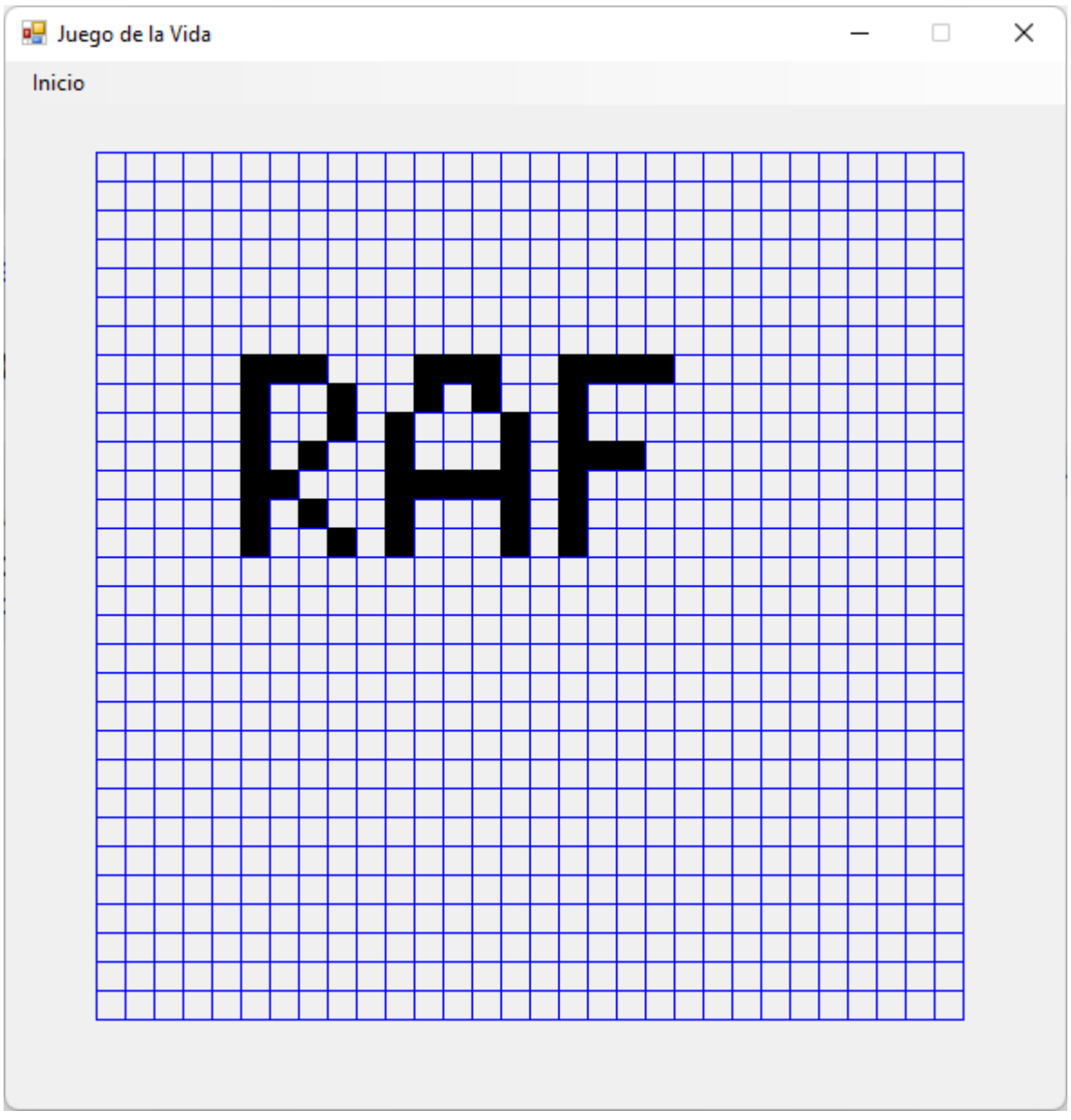


Ilustración 56: Ejemplo de activar y desactivar celdas con el clic del ratón

Mejorando el juego de la vida

En este ejemplo, el jugador puede darle pausa al juego, hacer cambios en las celdas con el ratón y luego volver a reanudar el juego tomando los cambios hechos.

03. Animación/07. JuegoVidaMejorado.zip

```
//Poner en pausa, poder cambiar las celdas y reanudar el juego
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Constantes para calificar cada celda del tablero
        private const int INACTIVA = 0;
        private const int ACTIVA = 1;

        int[,] Tablero; //Dónde ocurre realmente la acción

        //Tamaño de cada celda
        int tamanoX, tamanoY, desplaza;

        //Pausa o continúa el juego
        bool JuegoPausado;

        Random azar; //Único generador de números aleatorios.

        public Form1() {
            InitializeComponent();
            azar = new Random();
            IniciarParametros();
        }

        private void mnuReiniciar_Click(object sender, EventArgs e) {
            IniciarParametros();
        }

        public void IniciarParametros() {
            Tablero = new int[30, 30]; //Inicializa el tablero.

            //Tamaño de cada celda
            tamanoX = 500 / Tablero.GetLength(0);
            tamanoY = 500 / Tablero.GetLength(1);
            desplaza = 50;

            //¿Pausa o no el juego?
            JuegoPausado = false;

            //Habrán celdas activas en el 20%
            int Activos = Tablero.GetLength(0) * Tablero.GetLength(1) * 20 / 100;
            for (int cont = 1; cont <= Activos; cont++) {
                int posX, posY;
                do {
                    posX = azar.Next(0, Tablero.GetLength(0));
                    posY = azar.Next(0, Tablero.GetLength(1));
                } while (Tablero[posX, posY] != 0);
                Tablero[posX, posY] = ACTIVA;
            }

            TimerAnimar.Start();
        }


        private void TimerAnimar_Tick(object sender, System.EventArgs e) {
            if (JuegoPausado) return;
            Logica(); //Si el juego no está pausado entonces se ejecuta la lógica
            Refresh(); //Visual de la animación
        }

        public void Logica() {
            //Copia el tablero a uno temporal
            int[,] TableroTmp = Tablero.Clone() as int[,];

            //Va de celda en celda
            for (int posX = 0; posX < Tablero.GetLength(0); posX++) {
                for (int posY = 0; posY < Tablero.GetLength(1); posY++) {

                    //Determina el número de vecinos activos
```

```
                    int BordeXizq = posX - 1 < 0 ? Tablero.GetLength(0) - 1 : posX - 1;
                    int BordeYarr = posY - 1 < 0 ? Tablero.GetLength(1) - 1 : posY - 1;

                    int BordeXder = posX + 1 >= Tablero.GetLength(0) ? 0 : posX + 1;
                    int BordeYaba = posY + 1 >= Tablero.GetLength(1) ? 0 : posY + 1;

                    int vecinosActivos = Tablero[BordeXizq, BordeYarr];
                    vecinosActivos += Tablero[posX, BordeYarr];
                    vecinosActivos += Tablero[BordeXder, BordeYarr];

                    vecinosActivos += Tablero[BordeXizq, posY];
                    vecinosActivos += Tablero[BordeXder, posY];

                    vecinosActivos += Tablero[BordeXizq, BordeYaba];
                    vecinosActivos += Tablero[posX, BordeYaba];
                    vecinosActivos += Tablero[BordeXder, BordeYaba];

                    //Los cambios se registran en el tablero temporal

                    //Si la celda está inactiva y tiene 3 celdas activas alrededor, entonces la celda se
activa
                    if (Tablero[posX, posY] == INACTIVA && vecinosActivos == 3) TableroTmp[posX, posY] =
ACTIVA;

                    //Si la celda está activa y tiene menos de 2 celdas o más de 3 celdas, entonces la
celda se inactiva
                    if (Tablero[posX, posY] == ACTIVA && (vecinosActivos < 2 || vecinosActivos > 3))
TableroTmp[posX, posY] = INACTIVA;
                }
            }

            //El cambio en el tablero temporal se copia sobre el tablero
            Tablero = TableroTmp.Clone() as int[,];
        }

        private void mnuPausar_Click(object sender, EventArgs e) {
            if (!JuegoPausado) {
                JuegoPausado = true;
                mnuPausar.Text = "Continuar";
                this.Text = "Juego de la vida [PAUSADO]";
            }
            else {
                JuegoPausado = false;
                mnuPausar.Text = "Pausar";
                this.Text = "Juego de la vida [EN EJECUCIÓN]";
            }
        }

        private void mnuSalir_Click(object sender, EventArgs e) {
            Application.Exit();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Dibuja el arreglo bidimensional
            for (int Fila = 0; Fila < Tablero.GetLength(0); Fila++) {
                for (int Columna = 0; Columna < Tablero.GetLength(1); Columna++) {
                    int UbicaX = Fila * tamanoX + desplaza;
                    int UbicaY = Columna * tamanoY + desplaza;
                    switch (Tablero[Fila, Columna]) {
                        case INACTIVA: e.Graphics.DrawRectangle(Pens.Blue, UbicaX, UbicaY, tamanoX,
tamanoY); break;
                        case ACTIVA: e.Graphics.FillRectangle(Brushes.Black, UbicaX, UbicaY, tamanoX,
tamanoY); break;
                    }
                }
            }
        }

        //Cuando da clic sobre una celda, la activa o la desactiva siempre y cuando esté pausado el juego
        private void Form1_MouseClick(object sender, MouseEventArgs e) {
            if (!JuegoPausado) return;

            int Fila = (e.X - desplaza) / tamanoX;
            int Columna = (e.Y - desplaza) / tamanoY;

            if (Fila >= 0 && Columna >= 0 && Fila < Tablero.GetLength(0) && Columna <
Tablero.GetLength(1)) {
                switch (Tablero[Fila, Columna]) {
                    case INACTIVA: Tablero[Fila, Columna] = ACTIVA; break;
```

```
                        case ACTIVA: Tablero[Fila, Columna] = INACTIVA; break;
                    }
                }
                Refresh();
            }
        }
}
```

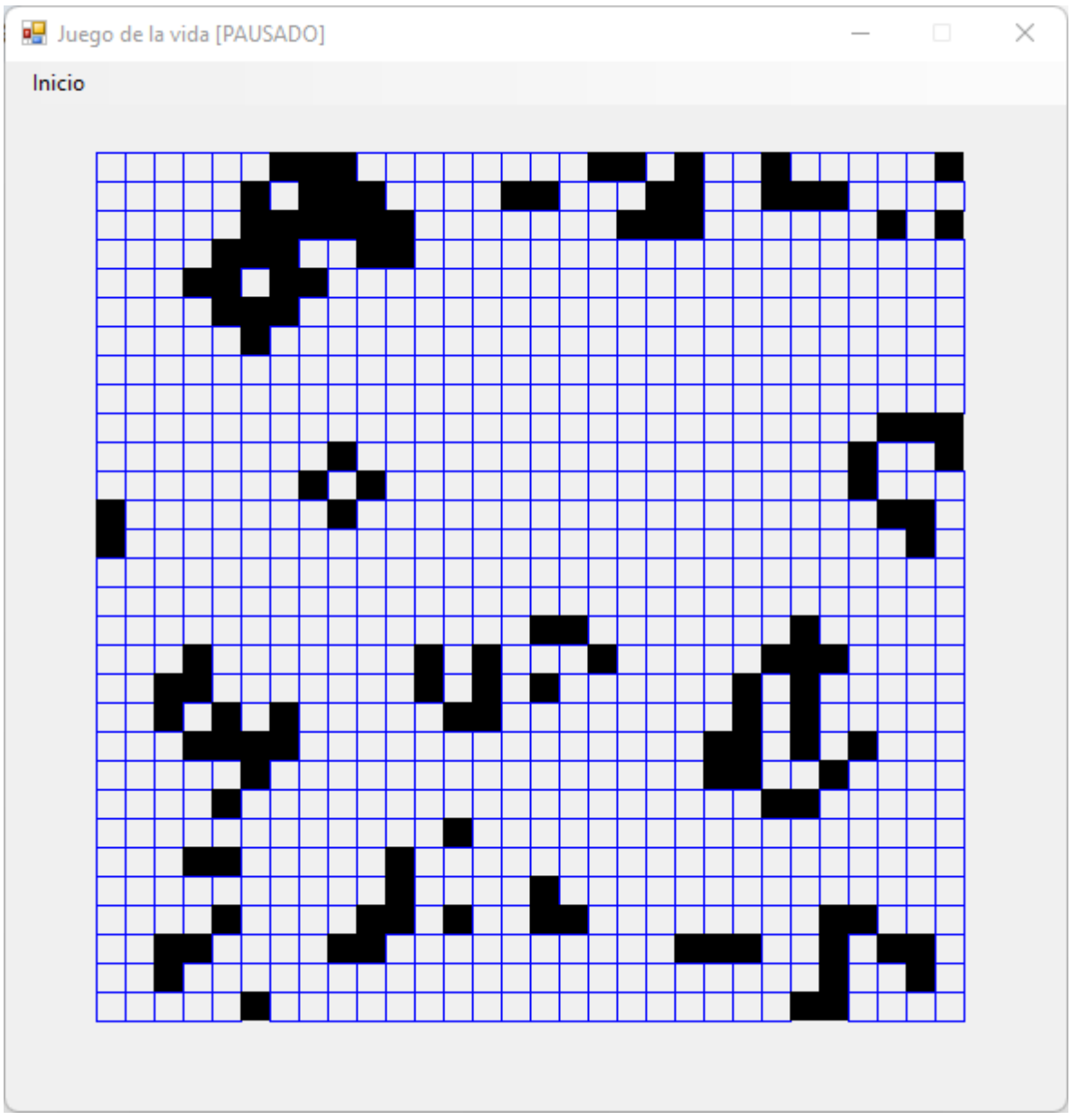


Ilustración 57: Se pone en pausa el juego

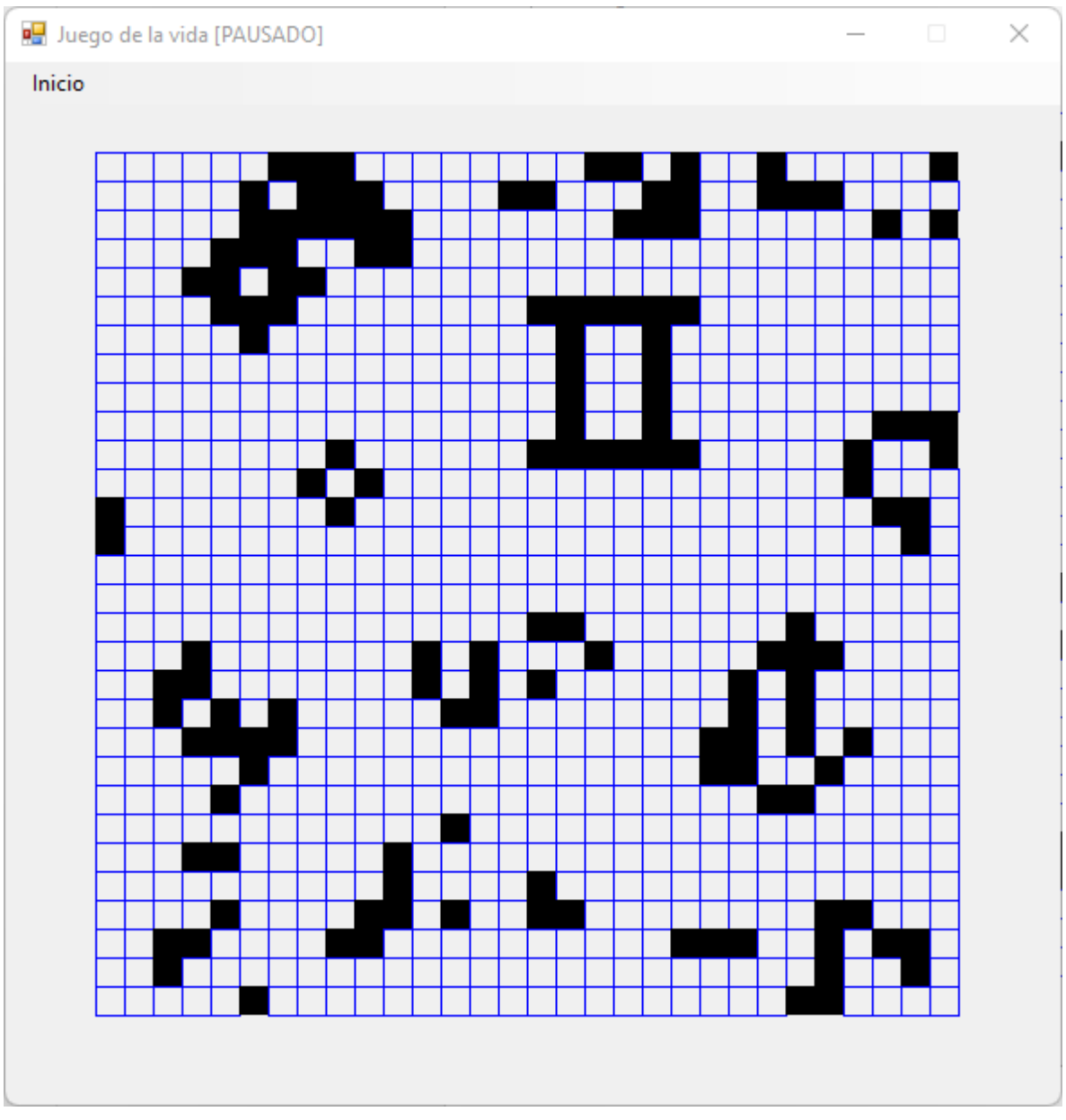


Ilustración 58: Se puede cambiar algo del tablero y volverlo a reanudar

Geometría

Se documentan las bases geométricas para trabajar con figuras 2D y 3D. Este conocimiento permite que los gráficos sean independientes del lenguaje de programación.

Líneas rectas con el algoritmo de Bresenham

Un método rápido para dibujar líneas píxel a píxel. Más información en: https://es.wikipedia.org/wiki/Algoritmo_de_Bresenham [11]

04. Geometría/01. Bresenham.cs

```
//Algoritmo de Bresenham, para dibujar líneas rectas rápidamente
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {

        public Form1() {
            InitializeComponent();
        }

        public void DibujarLineaRecta(Graphics lienzo, int iniX, int iniY, int finX, int finY) {
            int Contador, Distancia;
            int Xerror=0, Yerror=0, CambioX, CambioY, incrementoX, incrementoY;

            CambioX = finX - iniX;
            CambioY = finY - iniY;

            if (CambioX > 0) incrementoX = 1;
            else if (CambioX == 0) incrementoX = 0;
            else incrementoX = -1;

            if (CambioY > 0) incrementoY = 1;
            else if (CambioY == 0) incrementoY = 0;
            else incrementoY = -1;

            if (CambioX < 0) CambioX *= -1;
            if (CambioY < 0) CambioY *= -1;

            if (CambioX > CambioY)
                Distancia = CambioX;
            else
                Distancia = CambioY;

            for (Contador = 0; Contador <= Distancia + 1; Contador++) {
                lienzo.FillRectangle(Brushes.Black, iniX, iniY, 1, 1);
                Xerror += CambioX;
                Yerror += CambioY;
                if (Xerror > Distancia) {
                    Xerror -= Distancia;
                    iniX += incrementoX;
                }

                if (Yerror > Distancia) {
                    Yerror -= Distancia;
                    iniY += incrementoY;
                }
            }
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Ejemplo
            DibujarLineaRecta(e.Graphics, 16, 83, 197, 206);
        }
    }
}
```

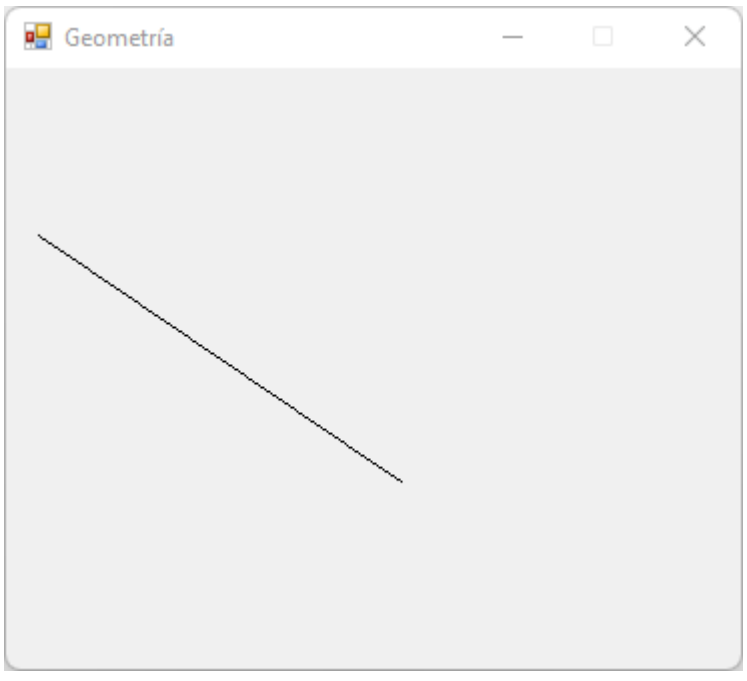


Ilustración 59: Línea recta

¡OJO! Por supuesto, uno usaría la instrucción DrawLine propia de C#. El programa anterior es para mostrar cómo está implementado el algoritmo haciendo uso de matemáticas de enteros.

Traslado de figuras planas

Mover la figura en el eje X y en el eje Y [12], es simplemente sumar o restar a los valores de las coordenadas:

NuevaPosicionX = PosicionOriginalX + MueveX

NuevaPosicionY = PosicionOriginalY + MueveY

04. Geometría/02. Traslado.cs

```
//Traslado de figuras en un plano
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {

        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Traslado horizontal
            int mueveX = 0;

            //Traslado vertical
            int mueveY = 0;

            //Dibuja un rectángulo con cuatro líneas
            e.Graphics.DrawLine(Pens.Black, 10 + mueveX, 10 + mueveY, 400 + mueveX, 10 + mueveY);
            e.Graphics.DrawLine(Pens.Black, 10 + mueveX, 250 + mueveY, 400 + mueveX, 250 + mueveY);
            e.Graphics.DrawLine(Pens.Black, 400 + mueveX, 10 + mueveY, 400 + mueveX, 250 + mueveY);
            e.Graphics.DrawLine(Pens.Black, 10 + mueveX, 10 + mueveY, 10 + mueveX, 250 + mueveY);
        }
    }
}
```

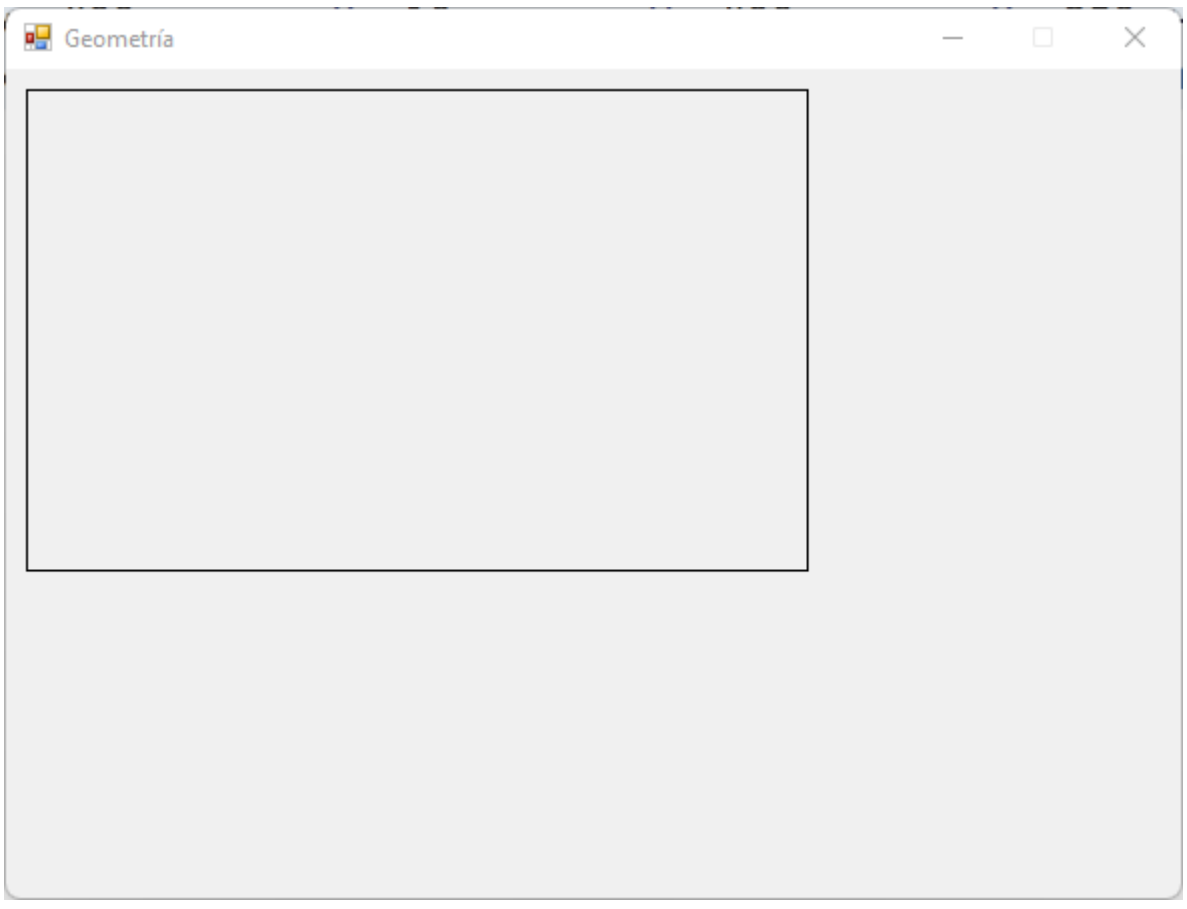


Ilustración 60: Posición original

Se mueve a la derecha con mueveX=100

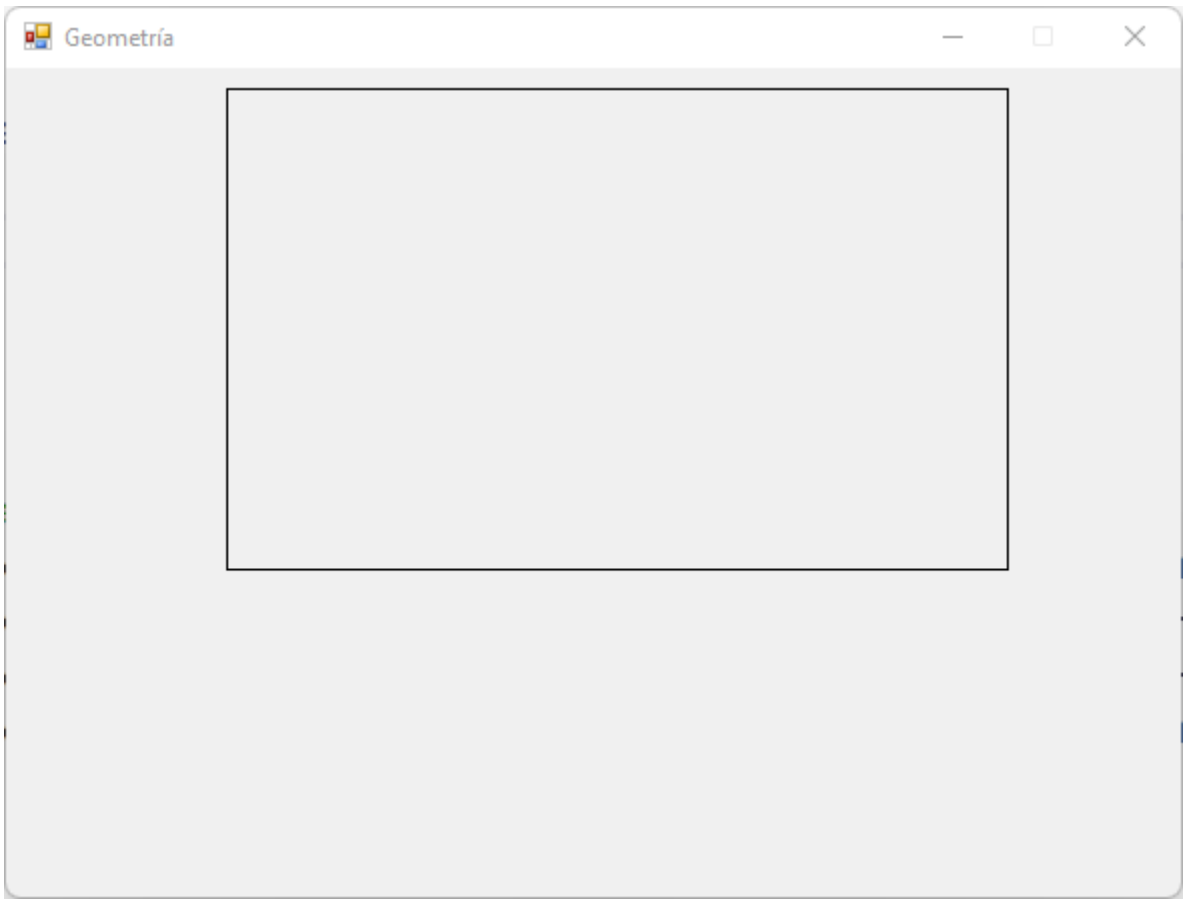


Ilustración 61: Desplazado a la derecha con mueveX = 100

Se desplaza hacia abajo con mueveY = 100

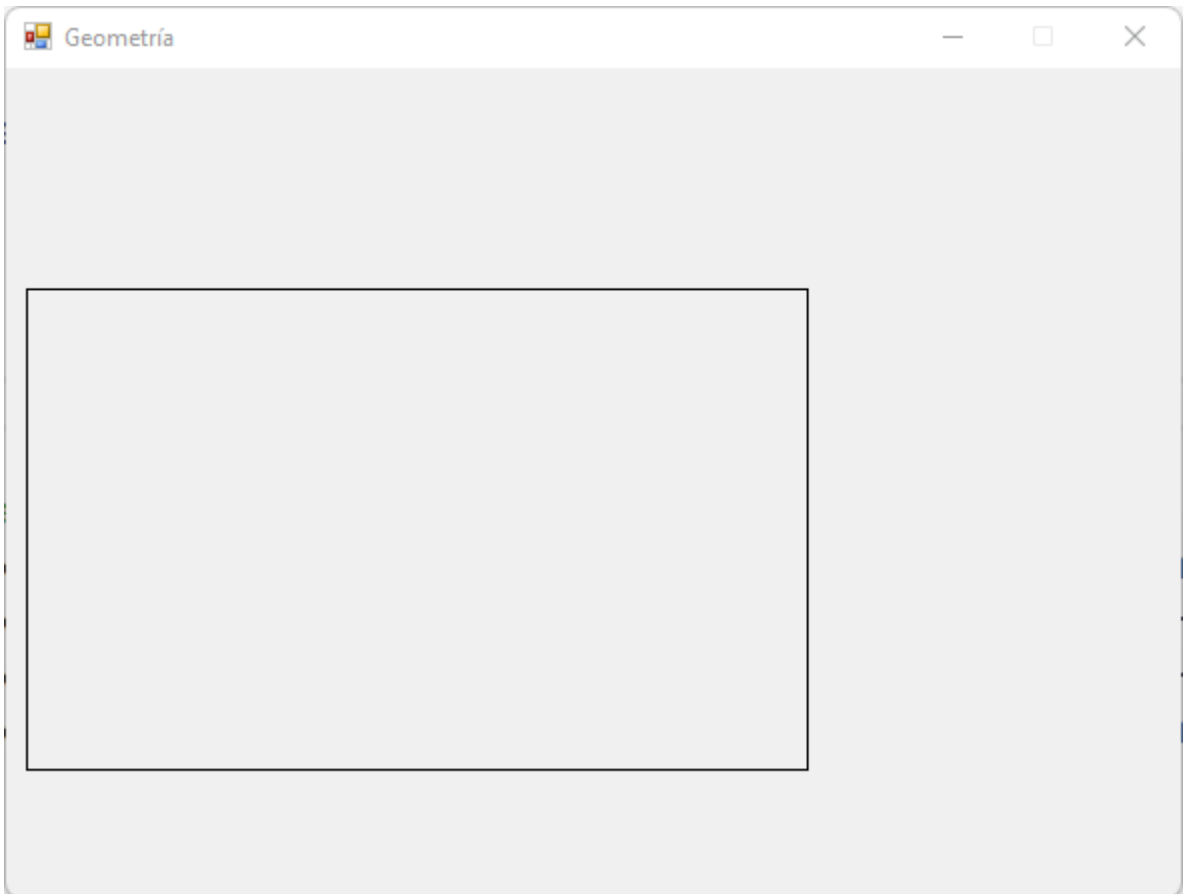


Ilustración 62: Desplazado hacia abajo con mueveY=100

Se mueve a la izquierda y abajo con mueveX=100 y mueveY=100 respectivamente

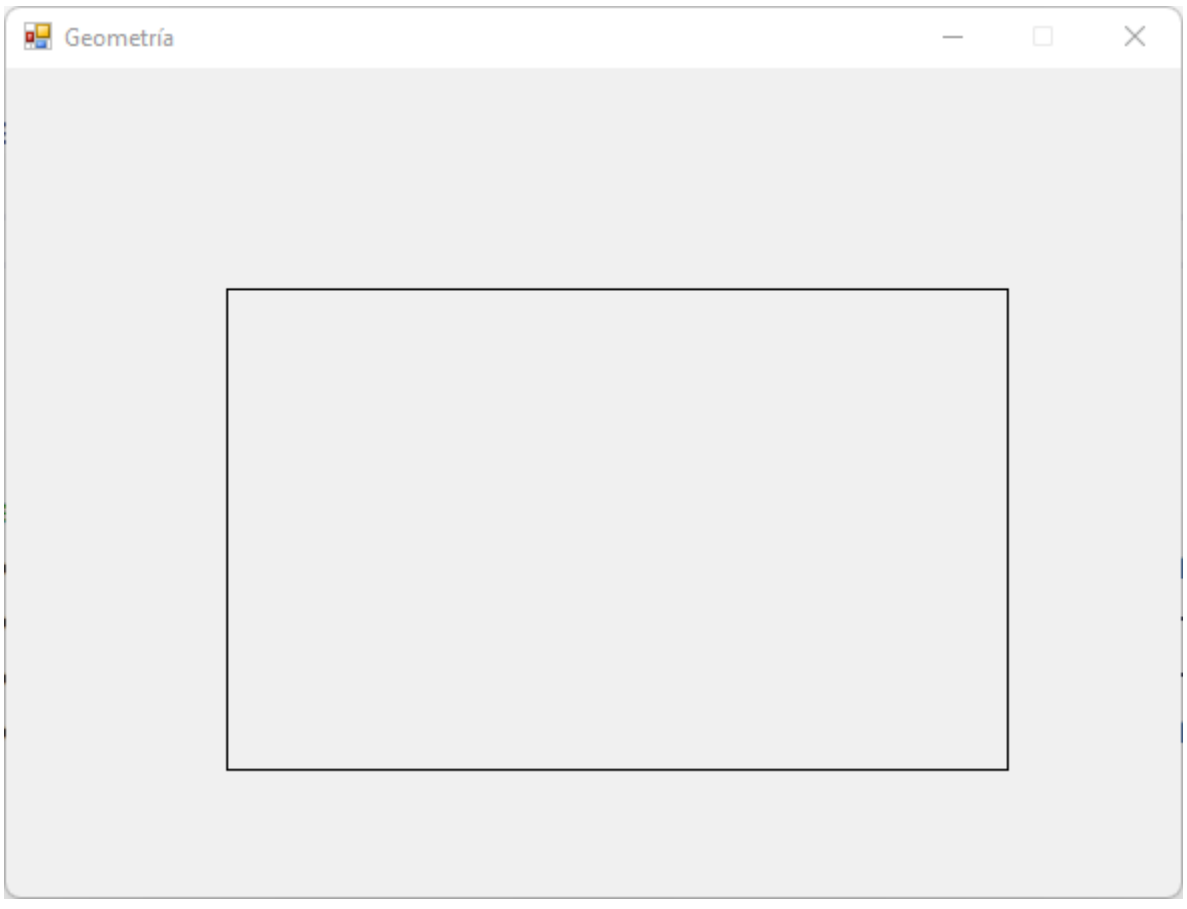


Ilustración 63: Se mueve a la derecha y abajo

Girar una figura en 2D

Para el giro [13] en determinado ángulo, se usa la siguiente fórmula para una coordenada plana:

PosicionNuevaX = PosicionOriginalX * coseno(Angulo) - PosicionOriginalY * seno(Angulo)

PosicionNuevaY = PosicionOriginalX * seno(Angulo) + PosicionOriginalY * coseno(Angulo)

04. Geometría/03. Giro.cs

```
//Giro de figuras en un plano
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {

        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Coordenadas de la figura
            int posXa, posYa, posXb, posYb, posXc, posYc;

            posXa = 80;
            posYa = 80;
            posXb = 400;
            posYb = 300;
            posXc = 500;
            posYc = 200;

            //Dibuja un triángulo con tres líneas
            e.Graphics.DrawLine(Pens.Black, posXa, posYa, posXb, posYb);
            e.Graphics.DrawLine(Pens.Black, posXb, posYb, posXc, posYc);
            e.Graphics.DrawLine(Pens.Black, posXc, posYc, posXa, posYa);

            //Ángulo de giro
            int AnguloGiro = 5;
            double AnguloRadianes = AnguloGiro * Math.PI / 180;

            //Cálcula el giro
            int posXga = Convert.ToInt32(posXa * Math.Cos(AnguloRadianes) - posYa *
Math.Sin(AnguloRadianes));
            int posYga = Convert.ToInt32(posXa * Math.Sin(AnguloRadianes) + posYa *
Math.Cos(AnguloRadianes));

            int posXgb = Convert.ToInt32(posXb * Math.Cos(AnguloRadianes) - posYb *
Math.Sin(AnguloRadianes));
            int posYgb = Convert.ToInt32(posXb * Math.Sin(AnguloRadianes) + posYb *
Math.Cos(AnguloRadianes));

            int posXgc = Convert.ToInt32(posXc * Math.Cos(AnguloRadianes) - posYc *
Math.Sin(AnguloRadianes));
            int posYgc = Convert.ToInt32(posXc * Math.Sin(AnguloRadianes) + posYc *
Math.Cos(AnguloRadianes));

            //Dibuja el triángulo con el giro
            e.Graphics.DrawLine(Pens.Red, posXga, posYga, posXgb, posYgb);
            e.Graphics.DrawLine(Pens.Red, posXgb, posYgb, posXgc, posYgc);
            e.Graphics.DrawLine(Pens.Red, posXgc, posYgc, posXga, posYga);
        }
    }
}
```

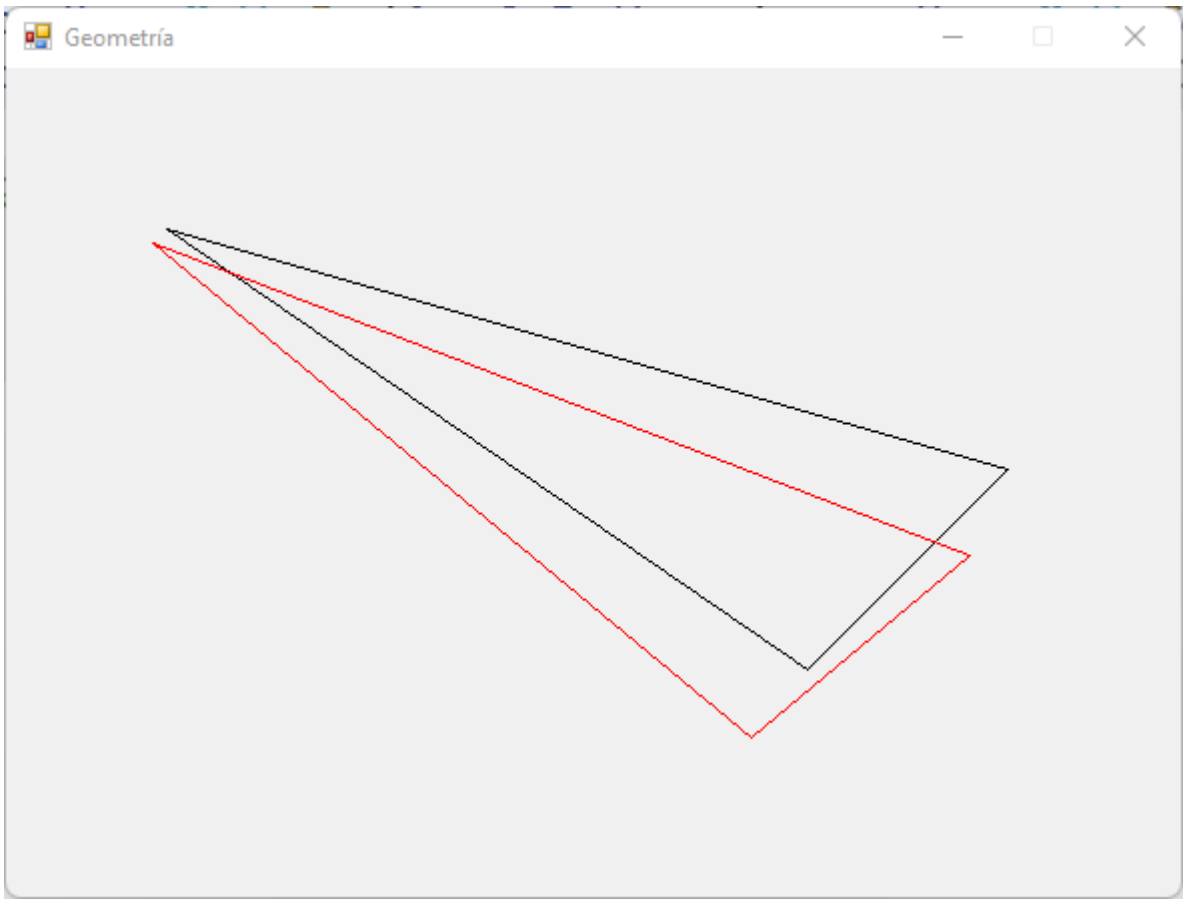


Ilustración 64: Giro del triángulo. En negro el original, en rojo al girarlo

En el ejemplo, se usó un giro de 5 grados (que se deben convertir en radianes antes de hacer uso de las funciones de seno y coseno). El ángulo se aplica desde la posición 0,0 de la ventana.

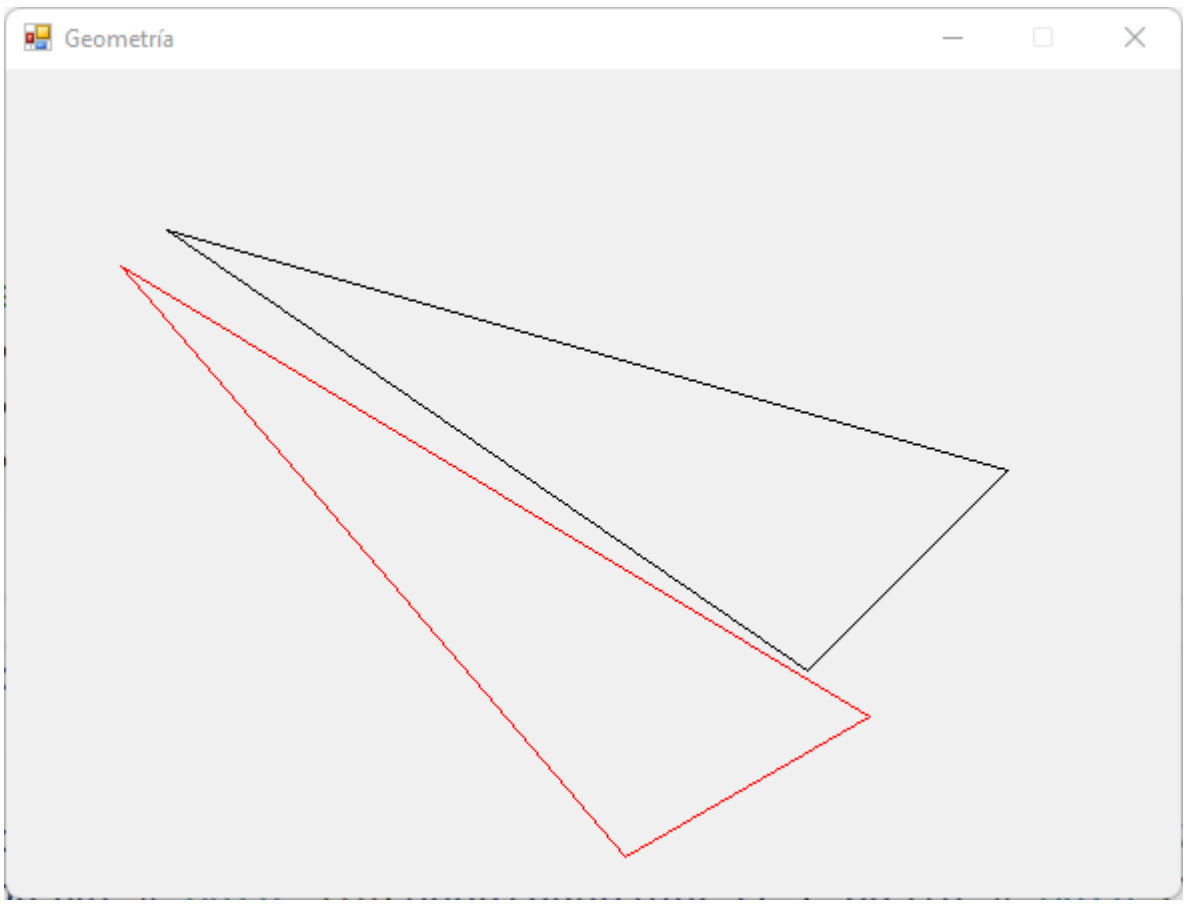


Ilustración 65: Giro de 15 grados

Giro sobre si misma

Necesitamos calcular el centroide de la figura (fácil si es un rectángulo), se aplica el giro y se compara cuánto se desplazó en X y Y con respecto a la posición original. Una vez tenemos esos datos, se restaura para toda la figura y tenemos el giro sobre sí misma.

04. Geometría/ 04. GiroEnSi.cs

```
//Giro de figuras en un plano
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {

        public Form1() {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Datos del rectángulo
            int posXa, posYa, Largo, Alto;
            posXa = 80;
            posYa = 80;
            Largo = 400;
            Alto = 180;

            //Deduce las otras tres coordenadas del rectángulo
            int posXb, posYb, posXc, posYc, posXd, posYd;
            posXb = posXa + Largo;
            posYb = posYa;
            posXc = posXa + Largo;
            posYc = posYa + Alto;
            posXd = posXa;
            posYd = posYa + Alto;

            //Dibuja un rectángulo con cuatro líneas
            e.Graphics.DrawLine(Pens.Black, posXa, posYa, posXb, posYb);
            e.Graphics.DrawLine(Pens.Black, posXb, posYb, posXc, posYc);
            e.Graphics.DrawLine(Pens.Black, posXc, posYc, posXd, posYd);
            e.Graphics.DrawLine(Pens.Black, posXd, posYd, posXa, posYa);

            //Ángulo de giro
            int AnguloGiro = 15;
            double AnguloRadianes = AnguloGiro * Math.PI / 180;

            //Centroide
            int posXcentro = posXa + Largo / 2;
            int posYcentro = posYa + Alto / 2;

            //Cálcula el giro
            int posXga = Convert.ToInt32(posXa * Math.Cos(AnguloRadianes) - posYa *
Math.Sin(AnguloRadianes));
            int posYga = Convert.ToInt32(posXa * Math.Sin(AnguloRadianes) + posYa *
Math.Cos(AnguloRadianes));

            int posXgb = Convert.ToInt32(posXb * Math.Cos(AnguloRadianes) - posYb *
Math.Sin(AnguloRadianes));
            int posYgb = Convert.ToInt32(posXb * Math.Sin(AnguloRadianes) + posYb *
Math.Cos(AnguloRadianes));

            int posXgc = Convert.ToInt32(posXc * Math.Cos(AnguloRadianes) - posYc *
Math.Sin(AnguloRadianes));
            int posYgc = Convert.ToInt32(posXc * Math.Sin(AnguloRadianes) + posYc *
Math.Cos(AnguloRadianes));

            int posXgd = Convert.ToInt32(posXd * Math.Cos(AnguloRadianes) - posYd *
Math.Sin(AnguloRadianes));
            int posYgd = Convert.ToInt32(posXd * Math.Sin(AnguloRadianes) + posYd *
Math.Cos(AnguloRadianes));

            //Giro del centroide
            int posXgcentro = Convert.ToInt32(posXcentro * Math.Cos(AnguloRadianes) - posYcentro *
Math.Sin(AnguloRadianes));
            int posYgcentro = Convert.ToInt32(posXcentro * Math.Sin(AnguloRadianes) + posYcentro *
Math.Cos(AnguloRadianes));

            //¿Cuánto se desplazo el centroide?
            int desplazaX = posXgcentro - posXcentro;
```

```
            int desplazaY = posYgcentro - posYcentro;

            //Dibuja el triángulo con el giro
            e.Graphics.DrawLine(Pens.Red, posXga - desplazaX, posYga - desplazaY, posXgb - desplazaX,
posYgb - desplazaY);
            e.Graphics.DrawLine(Pens.Red, posXgb - desplazaX, posYgb - desplazaY, posXgc - desplazaX,
posYgc - desplazaY);
            e.Graphics.DrawLine(Pens.Red, posXgc - desplazaX, posYgc - desplazaY, posXgd - desplazaX,
posYgd - desplazaY);
            e.Graphics.DrawLine(Pens.Red, posXga - desplazaX, posYga - desplazaY, posXgd - desplazaX,
posYgd - desplazaY);
        }
    }
}
```

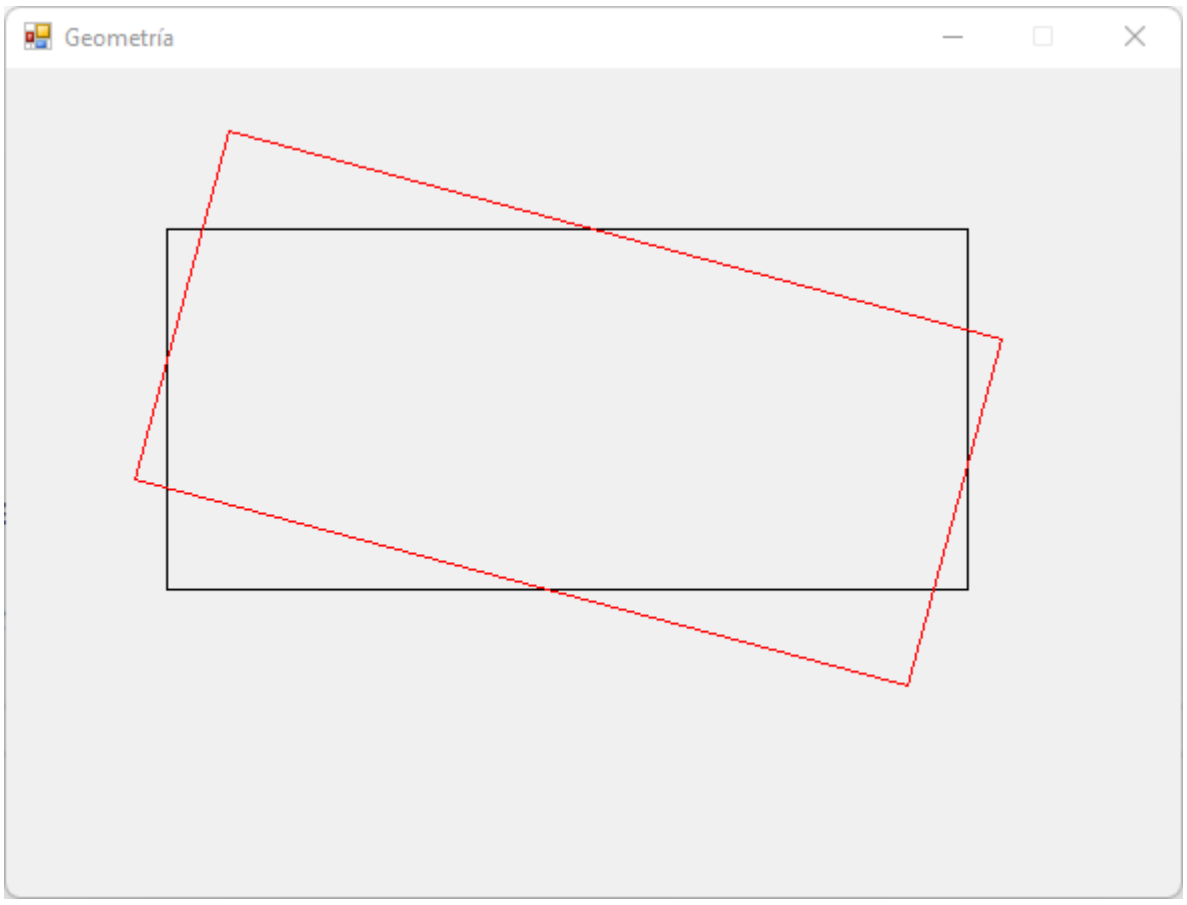


Ilustración 66: Giro sobre sí misma

Gráficos matemáticos en 2D

Dada una ecuación del tipo Y=F(X), por ejemplo:

$$Y = 0.1X^6 + 0.6X^5 - 0.7X^4 - 5.7X^3 + 2X^2 + 2X + 1$$

Para graficarla, se deben hacer estos pasos:

1. Saber el valor de inicio de X y el valor final de X. Se llamarán Xini y Xfin respectivamente.
2. Saber cuántos puntos se van a calcular. Se llamará numPuntos.
3. Con esos datos, se hace uso de un ciclo que calcule los valores de Y. Se almacena el par X, Y
4. Debido a que el eje Y aumenta hacia abajo en una pantalla o ventana, habrá que darles la vuelta a los valores de Y calculados, es decir, multiplicarlos por -1. Luego realmente se almacena X, -Y
5. Se requieren cuatro datos:
    a. El mínimo valor de X. Es el mismo Xini.
    b. El máximo valor de X. Se llamará maximoXreal que muy probablemente difiera de Xfin.
    c. El mínimo valor de Y obtenido. Se llamará Ymin.
    d. El máximo valor de Y obtenido. Se llamará Ymax.
6. También se requieren estos dos datos para poder ajustar el gráfico matemático a un área rectangular definida en la ventana.
    a. Coordenada superior izquierda en pantalla. Serán las coordenadas enteras positivas XpantallaIni, YpantallaIni
    b. Coordenada inferior derecha en pantalla. Serán las coordenadas enteras positivas XpantallaFin, YpantallaFin
7. Se calculan unas constantes de conversión con estas fórmulas:
    a. convierteX = (XpantallaFin – XpantallaIni) / (maximoXreal – Xini)
    b. convierteY = (YpantallaFin – YpantallaIni) / (Ymax – Ymin)
8. Tomar cada coordenada calculada de la ecuación (valor, valorY) y hacerle la conversión a pantalla plana con la siguiente fórmula:
    a. pantallaX = convierteX * (valorX - Xini) + XpantallaIni
    b. pantallaY = convierteY * (valorY - Ymin) + YpantallaIni
9. Se grafican esos puntos y se unen con líneas.

A continuación, el código que se encuentra en un archivo .zip porque es un proyecto escrito en Microsoft Visual Studio 2022. La clase Puntos es para almacenar los valores reales de la ecuación (valor, valorY) y los valores convertidos para que cuadren en pantalla (pantallaX, pantallaY)

05. Grafico2D/Grafico2D.zip/Puntos.cs

```
namespace Graficos {
    internal class Puntos {
        //Valor X, Y reales de la ecuación
        public double valorX, valorY;

        //Puntos convertidos a coordenadas enteras de pantalla
        public int pantallaX, pantallaY;

        public Puntos(double valorX, double valorY) {
            this.valorX = valorX;
            this.valorY = valorY;
        }
    }
}
```

Se crea un List que almacena todos los puntos. El procedimiento Lógica() se encarga de calcular los puntos reales y como cuadrarlos en pantalla. Cuando se llama a Paint() , sólo se lee el List que hace el gráfico con esos datos.

05. Grafico2D/Grafico2D.zip/Form1.cs

```
//Gráfico matemático en 2D
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        List<Puntos> puntos;
        int XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin;

        public Form1() {
            InitializeComponent();
            puntos = new List<Puntos>();
```

```
        //Área dónde se dibujará el gráfico matemático
        XpantallaIni = 20;
        YpantallaIni = 20;
        XpantallaFin = 600;
        YpantallaFin = 400;

        //Algoritmo que calcula los puntos del gráfico
        double minX = -5;
        double maxX = 3;
        int numPuntos = 80;
        Logica(minX, maxX, numPuntos);
    }

    public void Logica(double Xini, double Xfin, int numPuntos) {
        //Calcula los puntos de la ecuación a graficar
        double pasoX = (Xfin - Xini) / numPuntos;
        double Ymin = double.MaxValue; //El mínimo valor de Y obtenido
        double Ymax = double.MinValue; //El máximo valor de Y obtenido
        double maximoXreal = double.MinValue; //El máximo valor de X (difiere de Xfin)

        puntos.Clear();
        for (double X = Xini; X <= Xfin; X += pasoX) {
            double valY = -1*Ecuacion(X); //Se invierte el valor porque el eje Y aumenta hacia abajo
            if (valY > Ymax) Ymax = valY;
            if (valY < Ymin) Ymin = valY;
            if (X > maximoXreal) maximoXreal = X;
            puntos.Add(new Puntos(X, valY));
        }
        //¡OJO! X puede que no llegue a ser Xfin, por lo que la variable maximoXreal almacena el valor
máximo de X

        //Calcula los puntos a poner en la pantalla
        double convierteX = (XpantallaFin - XpantallaIni) / (maximoXreal - Xini);
        double convierteY = (YpantallaFin - YpantallaIni) / (Ymax - Ymin);

        for (int cont = 0; cont < puntos.Count; cont++) {
            puntos[cont].pantallaX = Convert.ToInt32(convierteX * (puntos[cont].valorX - Xini) +
XpantallaIni);
            puntos[cont].pantallaY = Convert.ToInt32(convierteY * (puntos[cont].valorY - Ymin) +
YpantallaIni);
        }
    }

    //Aquí está la ecuación que se desee graficar con variable independiente X
    public double Ecuacion(double X) {
        return 0.1*Math.Pow(X,6)+0.6*Math.Pow(X,5)-0.7*Math.Pow(X,4)-5.7*Math.Pow(X,3)+2*X*X+2*X+1;
    }

    //Pinta la ecuación
    private void Form1_Paint(object sender, PaintEventArgs e) {
        Graphics lienzo = e.Graphics;
        Pen lapiz = new Pen(Color.Blue, 3);

        //Un recuadro para ver el área del gráfico
        lienzo.FillRectangle(Brushes.Black, XpantallaIni, YpantallaIni, XpantallaFin-XpantallaIni,
YpantallaFin-YpantallaIni);

        //Dibuja el gráfico matemático
        for (int cont = 0; cont < puntos.Count - 1; cont++) {
            lienzo.DrawLine(lapiz, puntos[cont].pantallaX, puntos[cont].pantallaY, puntos[cont +
1].pantallaX, puntos[cont + 1].pantallaY);
        }
    }
}
```

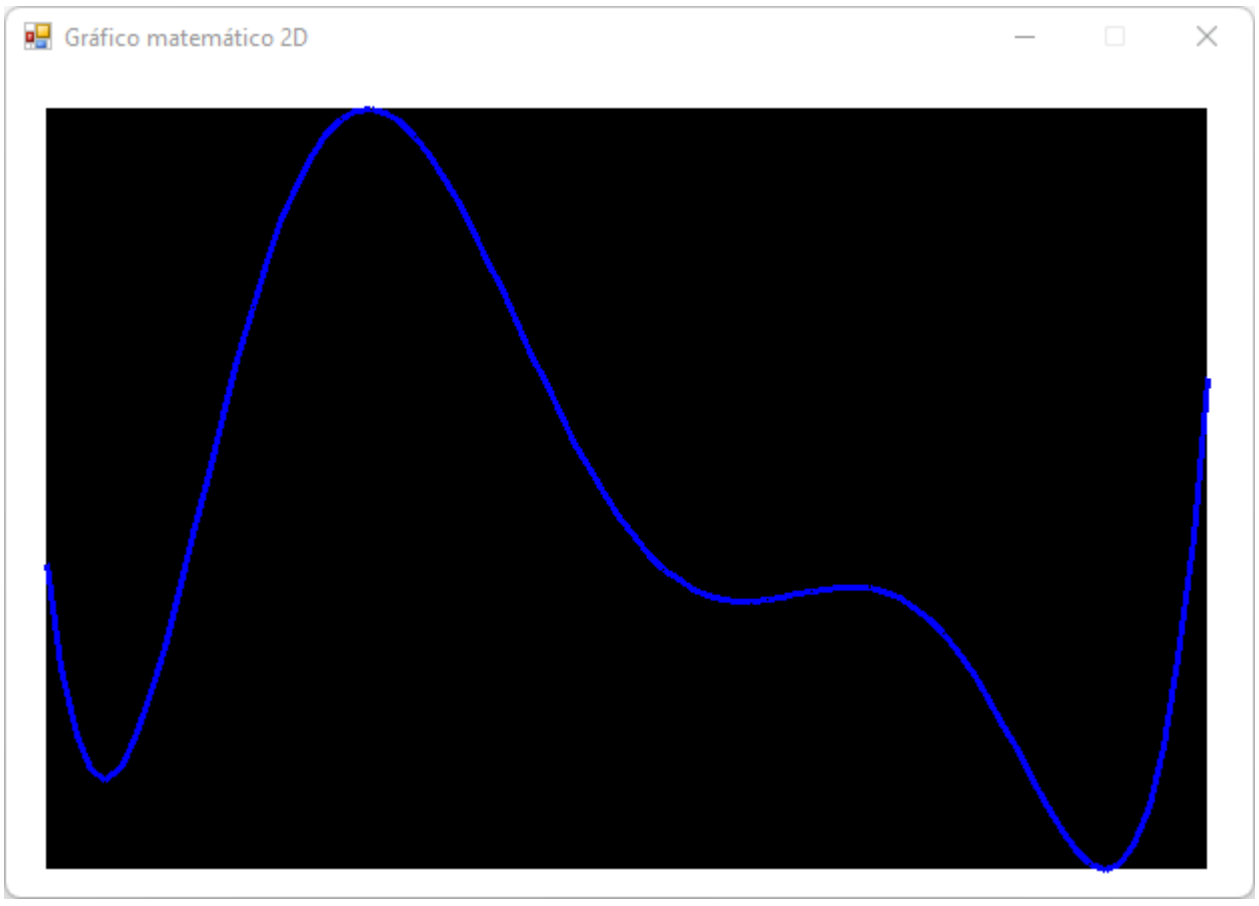


Ilustración 67: Gráfico de un polinomio

Gráfico matemático 2D animado

Dada una ecuación del tipo Y=F(X, T), donde X es la variables independiente y T es el tiempo, por ejemplo:

$$Y = X * (T + 1) * seno(X^2 * T^2)$$

Se procede a graficarlo de esta forma:

1. Saber el valor de inicio de T y el valor final de T. Se llamarán Tminimo y Tmaximo respectivamente.
2. Saber la tasa de incremento del tiempo, para ir desde Tminimo hasta Tmaximo, una vez se alcance Tmaximo se devuelve decrementando a la misma tasa hasta Tminimo y vuelve a empezar.
3. Saber el valor de inicio de X y el valor final de X. Se llamarán Xini y Xfin respectivamente.
4. Saber cuántos puntos se van a calcular. Se llamará numPuntos.
5. Con un control Timer, en cada tick se da un valor a T y se hacen los siguientes cálculos:
    a. Con X desde Xini hasta Xfin, se calculan los valores de Y. Se almacena el par X, Y
    b. Debido a que el eje Y aumenta hacia abajo en una pantalla o ventana, habrá que darles la vuelta a los valores de Y calculados, es decir, multiplicarlos por -1. Luego realmente se almacena X, -Y
    c. Se requieren cuatro datos:
        i. El mínimo valor de X. Es el mismo Xini.
        ii. El máximo valor de X. Se llamará maximoXreal que muy probablemente difiera de Xfin.
        iii. El mínimo valor de Y obtenido. Se llamará Ymin.
        iv. El máximo valor de Y obtenido. Se llamará Ymax.
    d. También se requieren estos dos datos para poder ajustar el gráfico matemático a un área rectangular definida en la ventana.
        i. Coordenada superior izquierda en pantalla. Serán las coordenadas enteras positivas XpantallaIni, YpantallaIni
        ii. Coordenada inferior derecha en pantalla. Serán las coordenadas enteras positivas XpantallaFin, YpantallaFin
    e. Se calculan unas constantes de conversión con estas fórmulas:
        i. convierteX = (XpantallaFin – XpantallaIni) / (maximoXreal – Xini)
        ii. convierteY = (YpantallaFin – YpantallaIni) / (Ymax – Ymin)
    f. Tomar cada coordenada calculada de la ecuación (valor, valorY) y hacerle la conversión a pantalla plana con la siguiente fórmula:
        i. pantallaX = convierteX * (valorX - Xini) + XpantallaIni
        ii. pantallaY = convierteY * (valorY - Ymin) + YpantallaIni
    g. Se grafican esos puntos y se unen con líneas.
    h. Se borra la pantalla y se da otro valor de T, eso da la sensación de animación.

A continuación, el código que se encuentra en un archivo .zip porque es un proyecto escrito en Microsoft Visual Studio 2022. La clase Puntos es para almacenar los valores reales de la ecuación (valor, valorY) y los valores convertidos para que cuadren en pantalla (pantallaX, pantallaY)

06. Grafico2Danimado/Grafico2Danimado.zip/Puntos.cs

```
namespace Graficos {
    internal class Puntos {
        //Valor X, Y reales de la ecuación
        public double valorX, valorY;

        //Puntos convertidos a coordenadas enteras de pantalla
        public int pantallaX, pantallaY;

        public Puntos(double valorX, double valorY) {
            this.valorX = valorX;
            this.valorY = valorY;
        }
    }
}
```

06. Grafico2Danimado/Grafico2Danimado.zip/Form1.cs

```
//Gráfico matemático en 2D animado
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Para la variable temporal
        private double TiempoValor, Tminimo, Tmaximo, Tincrementa;

        //Datos de la ecuación (desde donde inicia y termina X)
```

```
        double minX;
        double maxX;
        int numPuntos;

        //Donde almacena los puntos
        List<Puntos> puntos;

        //Datos de la pantalla
        int XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin;

        private void timerAnimar_Tick(object sender, EventArgs e) {
            TiempoValor += Tincrementa;
            if (TiempoValor <= Tminimo || TiempoValor >= Tmaximo) Tincrementa = -Tincrementa;
            Logica(minX, maxX, numPuntos, TiempoValor);
            Refresh();
        }

        public Form1() {
            InitializeComponent();
            puntos = new List<Puntos>();

            //Área dónde se dibujará el gráfico matemático
            XpantallaIni = 20;
            YpantallaIni = 20;
            XpantallaFin = 700;
            YpantallaFin = 500;

            //Inicia el tiempo
            Tminimo = 0;
            Tmaximo = 2;
            Tincrementa = 0.05;
            TiempoValor = Tminimo;

            //Datos de la ecuación
            minX = -5;
            maxX = 3;
            numPuntos = 200;
        }

        public void Logica(double Xini, double Xfin, int numPuntos, double Tiempo) {
            //Calcula los puntos de la ecuación a graficar
            double pasoX = (Xfin - Xini) / numPuntos;
            double Ymin = double.MaxValue; //El mínimo valor de Y obtenido
            double Ymax = double.MinValue; //El máximo valor de Y obtenido
            double maximoXreal = double.MinValue; //El máximo valor de X (difiere de Xfin)

            puntos.Clear();
            for (double X = Xini; X <= Xfin; X += pasoX) {
                double valY = -1*Ecuacion(X, Tiempo); //Se invierte el valor porque el eje Y aumenta hacia
abajo
                if (valY > Ymax) Ymax = valY;
                if (valY < Ymin) Ymin = valY;
                if (X > maximoXreal) maximoXreal = X;
                puntos.Add(new Puntos(X, valY));
            }
            //¡OJO! X puede que no llegue a ser Xfin, por lo que la variable maximoXreal almacena el valor
máximo de X

            //Calcula los puntos a poner en la pantalla
            double convierteX = (XpantallaFin - XpantallaIni) / (maximoXreal - Xini);
            double convierteY = (YpantallaFin - YpantallaIni) / (Ymax - Ymin);

            for (int cont = 0; cont < puntos.Count; cont++) {
                puntos[cont].pantallaX = Convert.ToInt32(convierteX * (puntos[cont].valorX - Xini) +
XpantallaIni);
                puntos[cont].pantallaY = Convert.ToInt32(convierteY * (puntos[cont].valorY - Ymin) +
YpantallaIni);
            }
        }

        //Aquí está la ecuación que se desee graficar con variable independiente X y T
        public double Ecuacion(double X, double T) {
            return X * (T+1) * Math.Sin(X * X * T * T);
        }

        //Pinta la ecuación
        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Un recuadro para ver el área del gráfico
```

```
            e.Graphics.DrawRectangle(Pens.Blue, XpantallaIni, YpantallaIni, XpantallaFin-XpantallaIni,
YpantallaFin-YpantallaIni);

            //Dibuja el gráfico matemático
            for (int cont = 0; cont < puntos.Count - 1; cont++) {
                e.Graphics.DrawLine(Pens.Black, puntos[cont].pantallaX, puntos[cont].pantallaY,
puntos[cont + 1].pantallaX, puntos[cont + 1].pantallaY);
            }
        }
    }
}
```

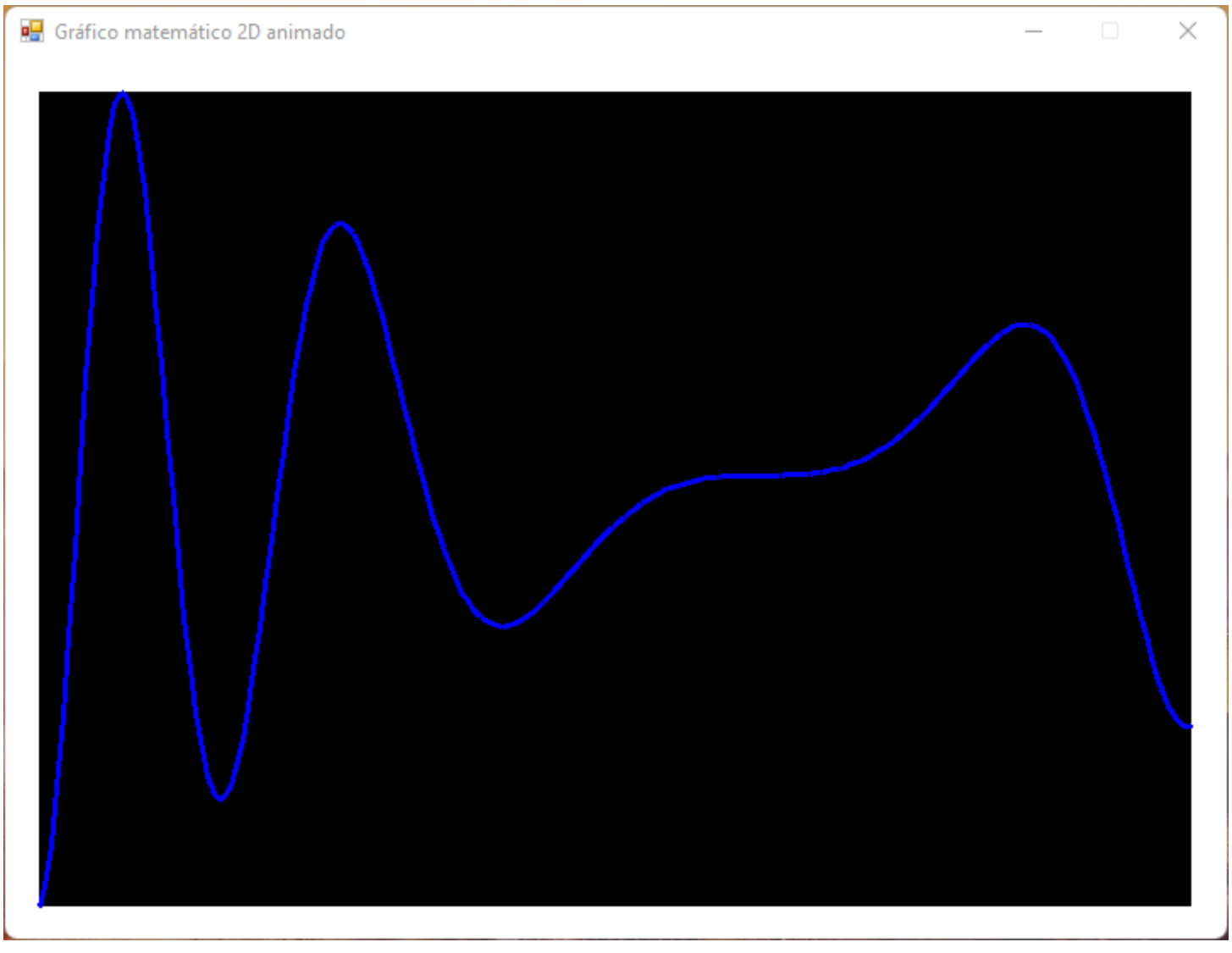


Ilustración 68: Un "frame" del gráfico 2D animado

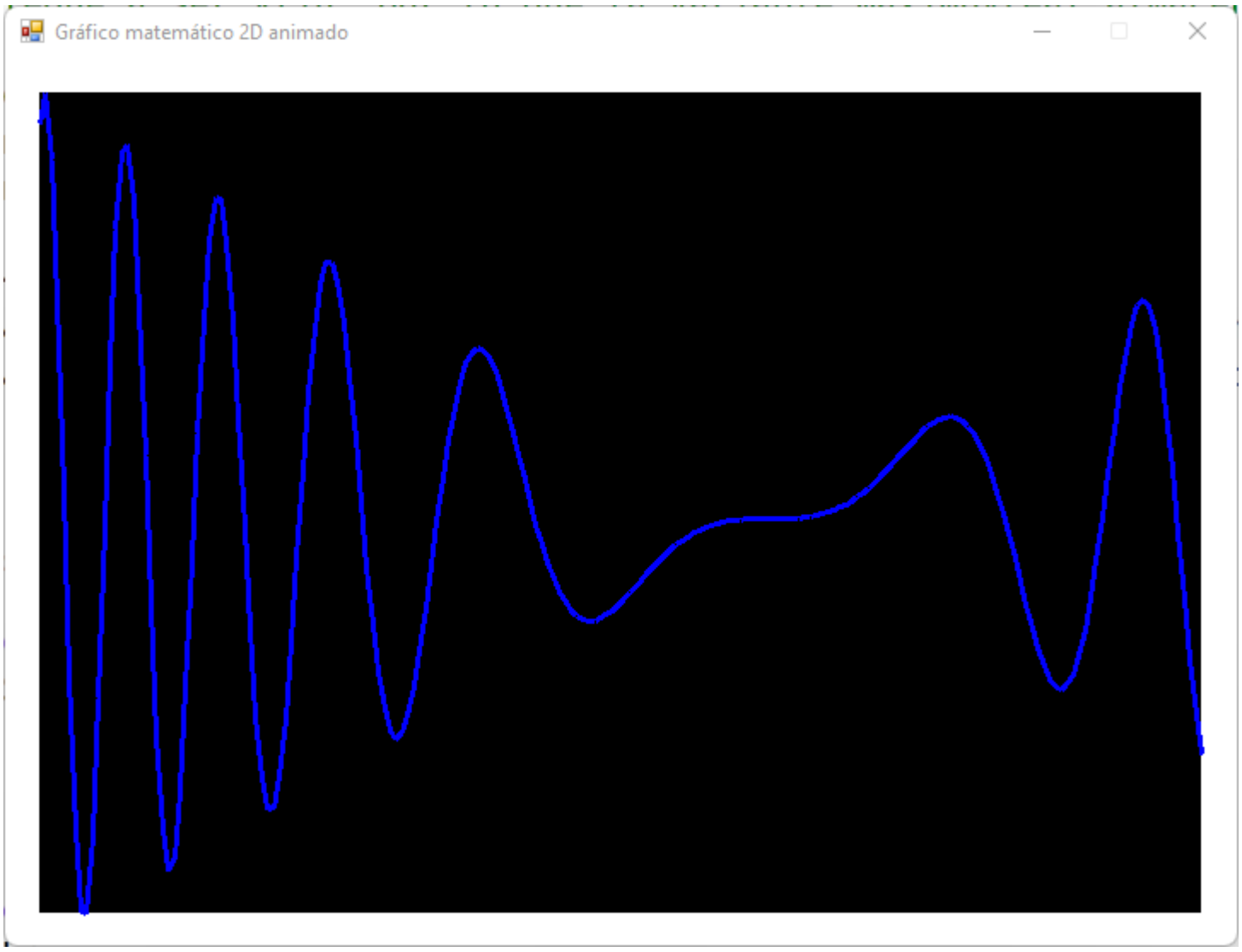


Ilustración 69: Otro "frame" del gráfico 2D animado

Gráficos polares

Dada una ecuación del tipo r=F(Θ), donde Θ es el ángulo y r el radio, por ejemplo:

$$r = 2 * seno(4 * \theta * \pi/180)$$

Se procede a graficarlo de esta forma:

1. Saber el valor de inicio de Θ y el valor final de Θ. Se llamarán minTheta y maxTheta respectivamente.
2. Saber cuántos puntos se van a calcular. Se llamará numPuntos.
3. Con esos datos, se hace uso de un ciclo que calcule los valores de r. Luego se hace esta conversión:

```
r = Ecuacion(theta);

X = r * Math.Cos(theta * Math.PI / 180);

Y = r * Math.Sin(theta * Math.PI / 180);
```

4. Debido a que el eje Y aumenta hacia abajo en una pantalla o ventana, habrá que darles la vuelta a los valores de Y calculados, es decir, multiplicarlos por -1. Luego realmente se almacena X, -Y
5. Se requieren cuatro datos:
   a. El mínimo valor de X. Es el mismo Xini.
   b. El máximo valor de X. Se llamará maximoXreal que muy probablemente difiera de Xfin.
   c. El mínimo valor de Y obtenido. Se llamará Ymin.
   d. El máximo valor de Y obtenido. Se llamará Ymax.
6. También se requieren estos dos datos para poder ajustar el gráfico matemático a un área rectangular definida en la ventana.
   a. Coordenada superior izquierda en pantalla. Serán las coordenadas enteras positivas XpantallaIni, YpantallaIni
   b. Coordenada inferior derecha en pantalla. Serán las coordenadas enteras positivas XpantallaFin, YpantallaFin
7. Se calculan unas constantes de conversión con estas fórmulas:
   a. convierteX = (XpantallaFin – XpantallaIni) / (maximoXreal – Xini)
   b. convierteY = (YpantallaFin – YpantallaIni) / (Ymax – Ymin)
8. Tomar cada coordenada calculada de la ecuación (valor, valorY) y hacerle la conversión a pantalla plana con la siguiente fórmula:
   a. pantallaX = convierteX * (valorX - Xini) + XpantallaIni
   b. pantallaY = convierteY * (valorY - Ymin) + YpantallaIni
9. Se grafican esos puntos y se unen con líneas.

A continuación, el código que se encuentra en un archivo .zip porque es un proyecto escrito en Microsoft Visual Studio 2022. La clase Puntos es para almacenar los valores reales de la ecuación (valor, valorY) y los valores convertidos para que cuadren en pantalla (pantallaX, pantallaY)

07. Polar/Polar.zip/Puntos.cs

```
namespace Graficos {
    internal class Puntos {
        //Valor X, Y reales de la ecuación
        public double valorX, valorY;

        //Puntos convertidos a coordenadas enteras de pantalla
        public int pantallaX, pantallaY;

        public Puntos(double valorX, double valorY) {
            this.valorX = valorX;
            this.valorY = valorY;
        }
    }
}
```

07. Polar/Polar.zip/Form1.cs

```
//Gráfico polar
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        List<Puntos> puntos;
        int XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin;

        public Form1() {
```

```
        InitializeComponent();
        puntos = new List<Puntos>();

        //Área dónde se dibujará el gráfico matemático
        XpantallaIni = 20;
        YpantallaIni = 20;
        XpantallaFin = 600;
        YpantallaFin = 400;

        //Algoritmo que calcula los puntos del gráfico
        double minTheta = 0;
        double maxTheta = 360;
        int numPuntos = 200;
        Logica(minTheta, maxTheta, numPuntos);
    }

    public void Logica(double thetaIni, double thetaFin, int numPuntos) {
        //Calcula los puntos de la ecuación a graficar
        double pasoTheta = (thetaFin - thetaIni) / numPuntos;
        double Ymin = double.MaxValue; //El mínimo valor de Y
        double Ymax = double.MinValue; //El máximo valor de Y
        double Xmin = double.MaxValue; //El máximo valor de X
        double Xmax = double.MinValue; //El máximo valor de X

        puntos.Clear();
        for (double theta = thetaIni; theta <= thetaFin; theta += pasoTheta) {
            double valorR = Ecuacion(theta);
            double X = valorR * Math.Cos(theta * Math.PI / 180);
            double Y = -1 * valorR * Math.Sin(theta * Math.PI / 180);

            if (Y > Ymax) Ymax = Y;
            if (Y < Ymin) Ymin = Y;
            if (X > Xmax) Xmax = X;
            if (X < Xmin) Xmin = X;
            puntos.Add(new Puntos(X, Y));
        }

        //Calcula los puntos a poner en la pantalla
        double convierteX = (XpantallaFin - XpantallaIni) / (Xmax - Xmin);
        double convierteY = (YpantallaFin - YpantallaIni) / (Ymax - Ymin);

        for (int cont = 0; cont < puntos.Count; cont++) {
            puntos[cont].pantallaX = Convert.ToInt32(convierteX * (puntos[cont].valorX - Xmin) +
XpantallaIni);
            puntos[cont].pantallaY = Convert.ToInt32(convierteY * (puntos[cont].valorY - Ymin) +
YpantallaIni);
        }
    }

    //Aquí está la ecuación polar que se desee graficar con variable Theta
    public double Ecuacion(double Theta) {
        return 2 * Math.Sin(4 * (Theta * Math.PI / 180));
    }

    //Pinta la ecuación
    private void Form1_Paint(object sender, PaintEventArgs e) {
        Graphics lienzo = e.Graphics;
        Pen lapiz = new Pen(Color.Blue, 3);

        //Un recuadro para ver el área del gráfico
        lienzo.FillRectangle(Brushes.Black, XpantallaIni, YpantallaIni, XpantallaFin-XpantallaIni,
YpantallaFin-YpantallaIni);

        //Dibuja el gráfico matemático
        for (int cont = 0; cont < puntos.Count - 1; cont++) {
            lienzo.DrawLine(lapiz, puntos[cont].pantallaX, puntos[cont].pantallaY, puntos[cont +
1].pantallaX, puntos[cont + 1].pantallaY);
        }
    }
  }
}
```

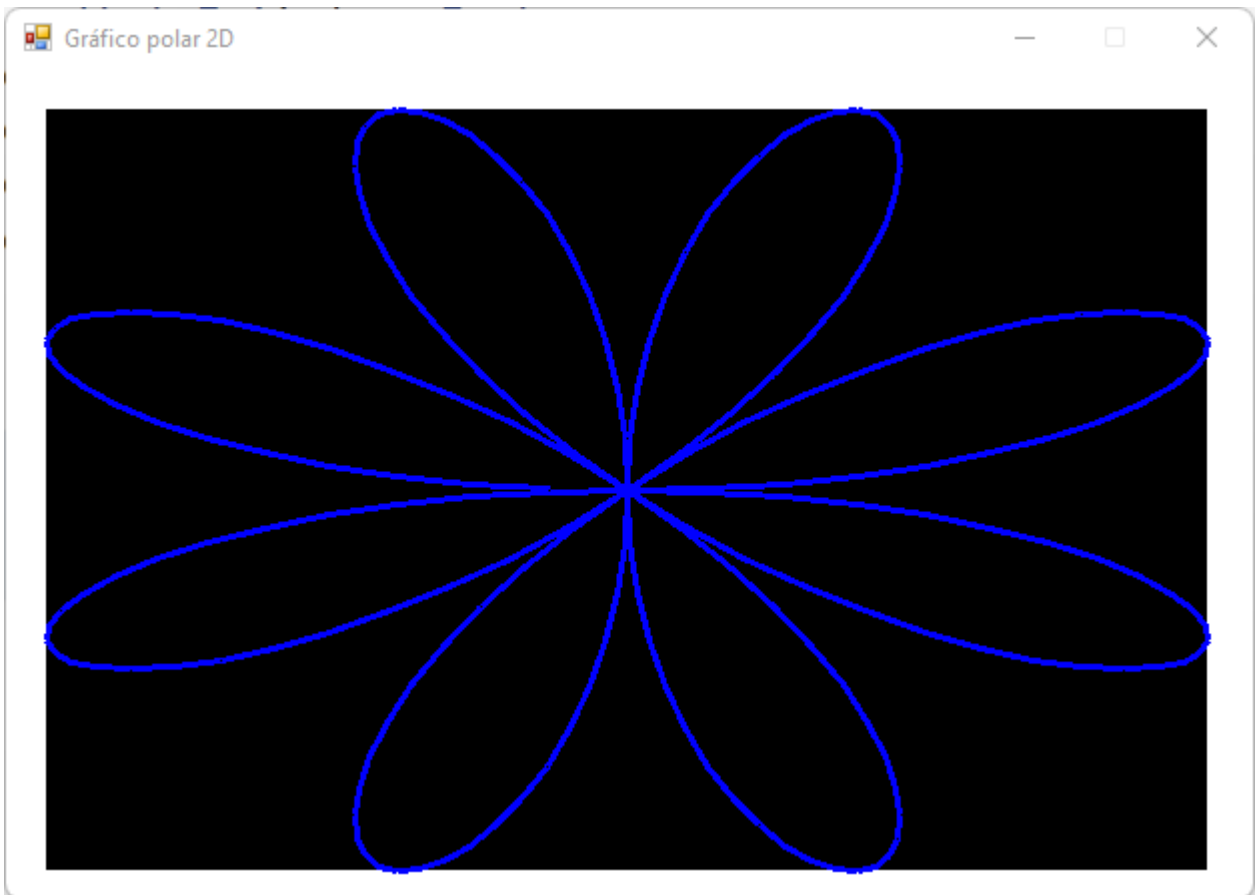


Ilustración 70: Gráfico polar

Gráfico polar animado

Dada una ecuación del tipo r=F(Θ), donde Θ es el ángulo y r el radio, por ejemplo:

$$r = 2 * seno(4 * \theta * \pi/180)$$

Se procede a graficarlo de esta forma:

1. Saber el valor de inicio de T y el valor final de T. Se llamarán Tminimo y Tmaximo respectivamente.
2. Saber la tasa de incremento del tiempo, para ir desde Tminimo hasta Tmaximo, una vez se alcance Tmaximo se devuelve decrementando a la misma tasa hasta Tminimo y vuelve a empezar.
3. Saber el valor de inicio de Θ y el valor final de Θ. Se llamarán minTheta y maxTheta respectivamente.
4. Saber cuántos puntos se van a calcular. Se llamará numPuntos.
5. Con un control Timer, en cada tick se da un valor a T y se hacen los siguientes cálculos:
    a. Con Θ desde minTheta y maxTheta se calculan los valores de r. Luego se hace esta conversión:

        r = Ecuacion(theta);

        X = r * Math.Cos(theta * Math.PI / 180);

        Y = r * Math.Sin(theta * Math.PI / 180);

    b. Debido a que el eje Y aumenta hacia abajo en una pantalla o ventana, habrá que darles la vuelta a los valores de Y calculados, es decir, multiplicarlos por -1. Luego realmente se almacena X, -Y
    c. Se requieren cuatro datos:
        i. El mínimo valor de X. Es el mismo Xini.
        ii. El máximo valor de X. Se llamará maximoXreal que muy probablemente difiera de Xfin.
        iii. El mínimo valor de Y obtenido. Se llamará Ymin.
        iv. El máximo valor de Y obtenido. Se llamará Ymax.
    d. También se requieren estos dos datos para poder ajustar el gráfico matemático a un área rectangular definida en la ventana.
        i. Coordenada superior izquierda en pantalla. Serán las coordenadas enteras positivas XpantallaIni, YpantallaIni
        ii. Coordenada inferior derecha en pantalla. Serán las coordenadas enteras positivas XpantallaFin, YpantallaFin
    e. Se calculan unas constantes de conversión con estas fórmulas:
        i. convierteX = (XpantallaFin – XpantallaIni) / (maximoXreal – Xini)
        ii. convierteY = (YpantallaFin – YpantallaIni) / (Ymax – Ymin)
    f. Tomar cada coordenada calculada de la ecuación (valor, valorY) y hacerle la conversión a pantalla plana con la siguiente fórmula:
        i. pantallaX = convierteX * (valorX - Xini) + XpantallaIni
        ii. pantallaY = convierteY * (valorY - Ymin) + YpantallaIni
    g. Se grafican esos puntos y se unen con líneas.
    h. Se borra la pantalla y se da otro valor de T, eso da la sensación de animación.

A continuación, el código que se encuentra en un archivo .zip porque es un proyecto escrito en Microsoft Visual Studio 2022. La clase Puntos es para almacenar los valores reales de la ecuación (valor, valorY) y los valores convertidos para que cuadren en pantalla (pantallaX, pantallaY)

08. PolarAnimado/PolarAnimado.zip/Puntos.cs

```
namespace Graficos {
    internal class Puntos {
        //Valor X, Y reales de la ecuación
        public double valorX, valorY;

        //Puntos convertidos a coordenadas enteras de pantalla
        public int pantallaX, pantallaY;

        public Puntos(double valorX, double valorY) {
            this.valorX = valorX;
            this.valorY = valorY;
        }
    }
}
```

```
//Gráfico polar animado
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Para la variable temporal
        private double TiempoValor, Tminimo, Tmaximo, Tincrementa;

        //Datos de la ecuación (desde donde inicia y termina Theta)
        double minTheta;
        double maxTheta;
        int numPuntos;

        //Donde almacena los puntos
        List<Puntos> puntos;

        //Datos de la pantalla
        int XpantallaIni, YpantallaIni, XpantallaFin, YpantallaFin;

        public Form1() {
            InitializeComponent();
            puntos = new List<Puntos>();

            //Área dónde se dibujará el gráfico matemático
            XpantallaIni = 20;
            YpantallaIni = 20;
            XpantallaFin = 700;
            YpantallaFin = 500;

            //Inicia el tiempo
            Tminimo = 0;
            Tmaximo = 5;
            Tincrementa = 0.05;
            TiempoValor = Tminimo;

            //Datos de la ecuación
            minTheta = 0;
            maxTheta = 360;
            numPuntos = 200;
        }

        private void timerAnimar_Tick(object sender, EventArgs e) {
            TiempoValor += Tincrementa;
            if (TiempoValor <= Tminimo || TiempoValor >= Tmaximo) Tincrementa = -Tincrementa;
            Logica(minTheta, maxTheta, numPuntos, TiempoValor);
            Refresh();
        }

        public void Logica(double thetaIni, double thetaFin, int numPuntos, double Tiempo) {
            //Calcula los puntos de la ecuación a graficar
            double pasoTheta = (thetaFin - thetaIni) / numPuntos;
            double Ymin = double.MaxValue; //El mínimo valor de Y
            double Ymax = double.MinValue; //El máximo valor de Y
            double Xmin = double.MaxValue; //El máximo valor de X
            double Xmax = double.MinValue; //El máximo valor de X

            puntos.Clear();
            for (double theta = thetaIni; theta <= thetaFin; theta += pasoTheta) {
                double valorR = Ecuacion(theta, Tiempo);
                double X = valorR * Math.Cos(theta * Math.PI / 180);
                double Y = -1 * valorR * Math.Sin(theta * Math.PI / 180);

                if (Y > Ymax) Ymax = Y;
                if (Y < Ymin) Ymin = Y;
                if (X > Xmax) Xmax = X;
                if (X < Xmin) Xmin = X;
                puntos.Add(new Puntos(X, Y));
            }

            //Calcula los puntos a poner en la pantalla
            double convierteX = (XpantallaFin - XpantallaIni) / (Xmax - Xmin);
            double convierteY = (YpantallaFin - YpantallaIni) / (Ymax - Ymin);

            for (int cont = 0; cont < puntos.Count; cont++) {
```

```
                puntos[cont].pantallaX = Convert.ToInt32(convierteX * (puntos[cont].valorX - Xmin) +
XpantallaIni);
                puntos[cont].pantallaY = Convert.ToInt32(convierteY * (puntos[cont].valorY - Ymin) +
YpantallaIni);
            }
        }

        //Aquí está la ecuación que se desee graficar con variable Theta y T (tiempo)
        public double Ecuacion(double Theta, double T) {
            return 2 * (1 + Math.Sin(Theta * T * Math.PI / 180));
        }

        //Pinta la ecuación
        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics lienzo = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 3);

            //Un recuadro para ver el área del gráfico
            lienzo.FillRectangle(Brushes.Black, XpantallaIni, YpantallaIni, XpantallaFin-XpantallaIni,
YpantallaFin-YpantallaIni);

            //Dibuja el gráfico matemático
            for (int cont = 0; cont < puntos.Count - 1; cont++) {
                lienzo.DrawLine(lapiz, puntos[cont].pantallaX, puntos[cont].pantallaY, puntos[cont +
1].pantallaX, puntos[cont + 1].pantallaY);
            }
        }
    }
}
```

Ilustración 71:Un "frame" del gráfico polar animado

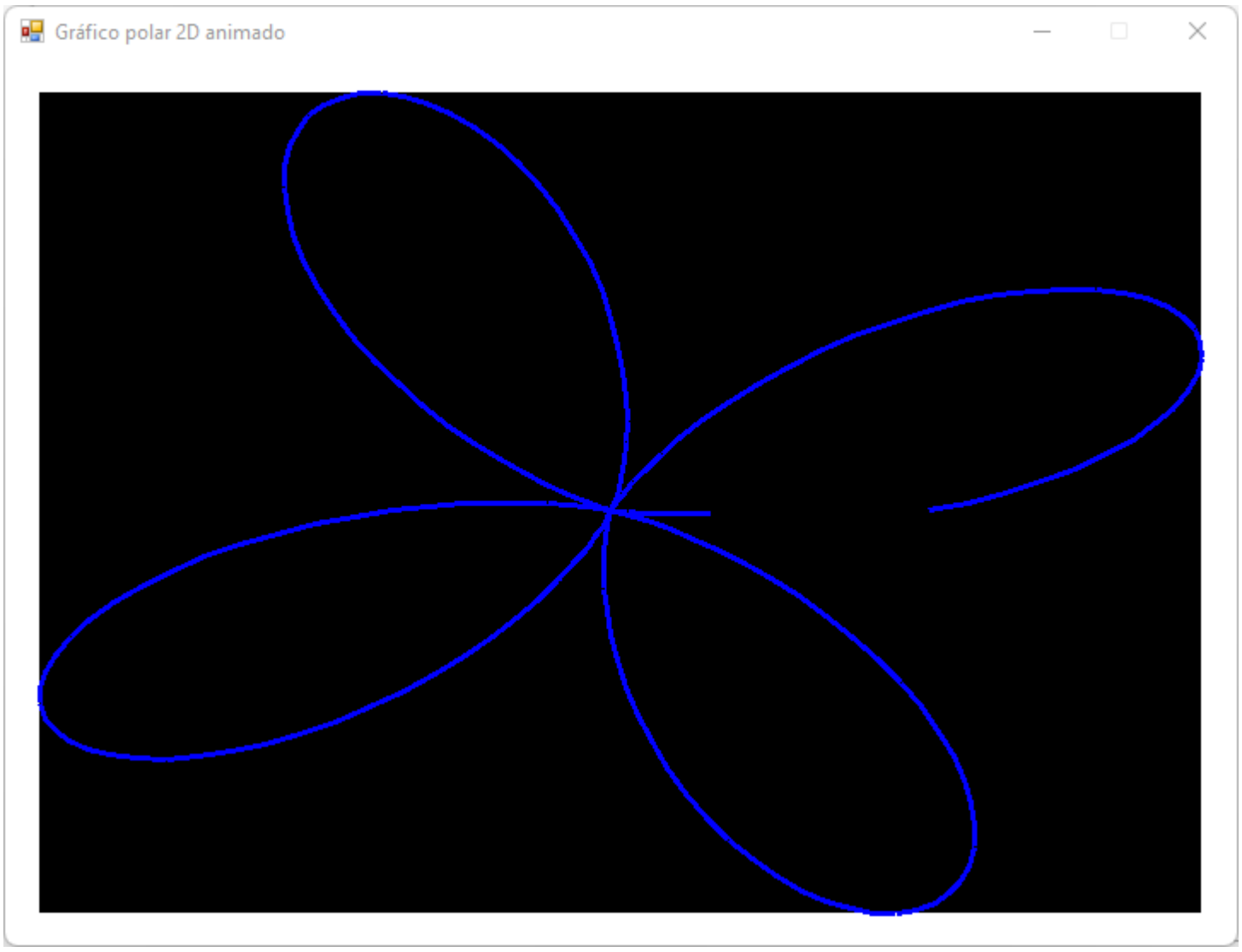


Ilustración 72:Otro "frame" del gráfico polar animado

Figuras en 3D

Proyección 3D a 2D

Dado un objeto tridimensional que flota entre la persona y una pantalla plana, ¿Cómo se proyectaría este objeto tridimensional en la pantalla? [14] Ese objeto tiene coordenadas (X,Y,Z) y deben convertirse a coordenadas planas (Xplano,Yplano). Hay que considerar la distancia de la persona u observador a ese objeto. Si la persona está muy cerca del objeto 3D, lo verá grande, si se aleja el observador, lo verá pequeño.

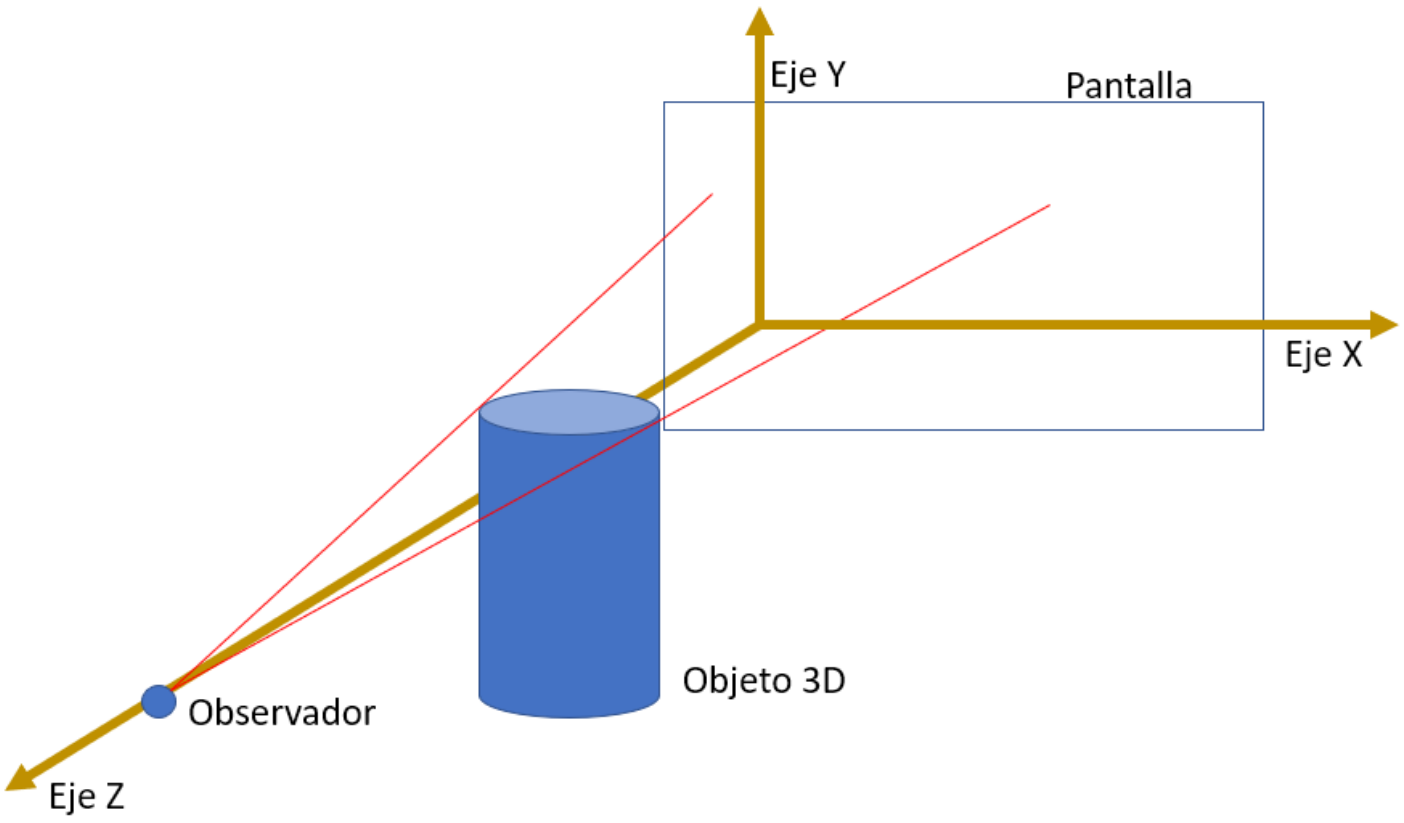


Ilustración 73: Proyección de objeto 3D en pantalla 2D

Las ecuaciones para pasar coordenadas X,Y,Z a coordenadas planas Xplano, Yplano, considerando la distancia del observador (ZPersona) es:

```
Xplano = (ZPersona * X) / (ZPersona - Z)
Yplano = (ZPersona * Y) / (ZPersona - Z)
```

Demostración, si ponemos el valor de X = 0 para ver sólo el comportamiento de Y, este sería el gráfico

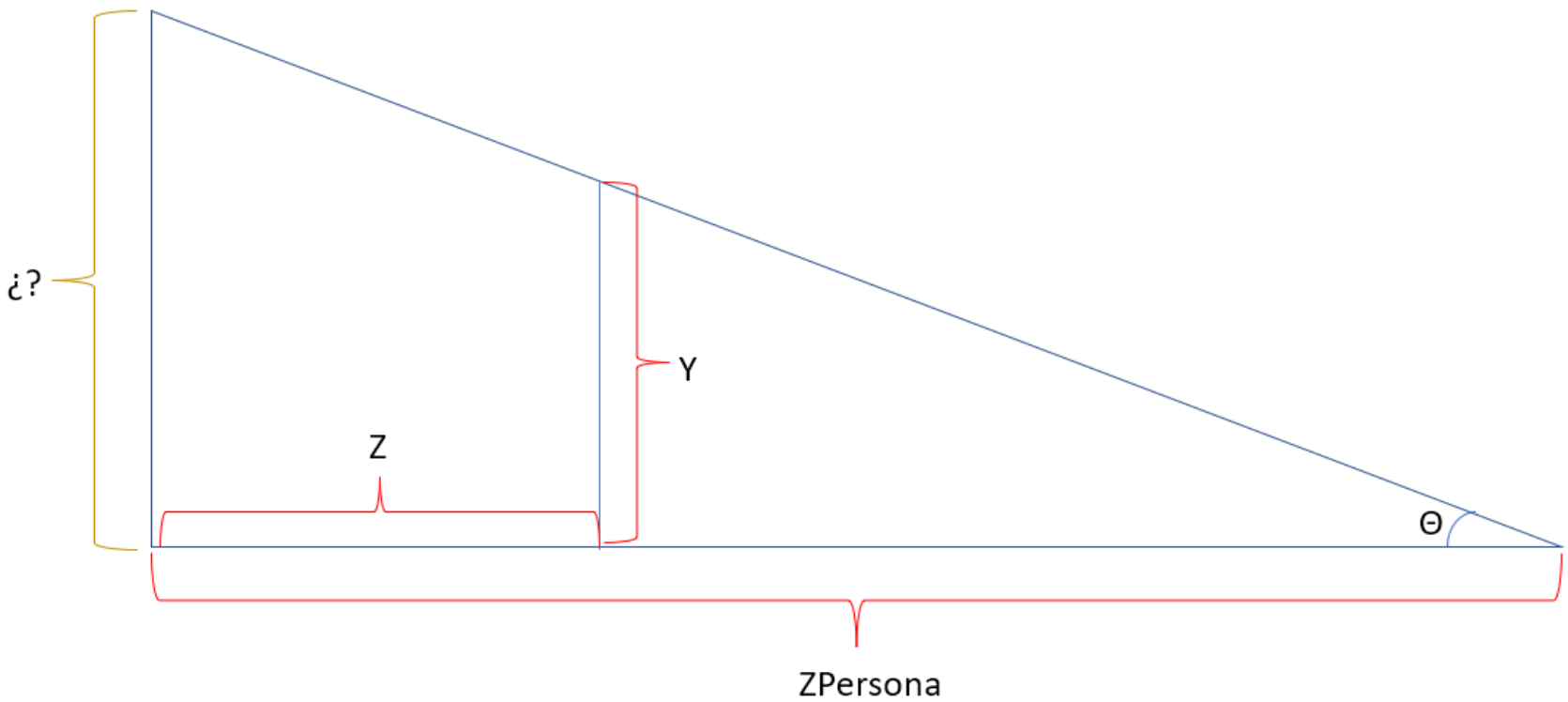


Ilustración 74: Esquema de proyección

Necesitamos hallar el valor de ¿? Que sería Yplano, ¿Cómo? La respuesta está en el ángulo Θ, más concretamente en la tangente que sería así:

$$\tan\theta = \frac{Y}{ZPersona - Z}$$

Pero también sabemos que:

$$\tan\theta = \frac{¿?}{ZPersona}$$

Luego igualamos:

$$\frac{¿?}{ZPersona} = \frac{Y}{ZPersona - Z}$$

Despejando:

$$¿? = \frac{Y * ZPersona}{ZPersona - Z}$$

Luego:

$$Yplano = \frac{Y * ZPersona}{ZPersona - Z}$$

Se hace lo mismo para saber el comportamiento de X.

Aquí una implementación:

09. 3D/01. Proyección.zip/Puntos.cs

```
namespace Graficos {
    internal class Puntos {
        //Coordenada espacial
        public int posX, posY, posZ;

        //Coordenada proyectada
        public int planoX, planoY;

        //Constructor
        public Puntos(int posX, int posY, int posZ) {
            this.posX = posX;
            this.posY = posY;
            this.posZ = posZ;
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(int ZPersona) {
            planoX = (ZPersona * posX) / (ZPersona - posZ);
            planoY = (ZPersona * posY) / (ZPersona - posZ);
        }
    }
}
```

09. 3D/01. Proyección.zip/Form1.cs

```
//Proyección 3D a 2D
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {

        //Donde almacena los puntos
        List<Puntos> puntos;

        public Form1() {
            InitializeComponent();
            puntos = new List<Puntos>();

            //Coordenadas del cubo
            puntos.Add(new Puntos(50, 50, 50));
```

```
            puntos.Add(new Puntos(100, 50, 50));
            puntos.Add(new Puntos(100, 100, 50));
            puntos.Add(new Puntos(50, 100, 50));
            puntos.Add(new Puntos(50, 50, 100));
            puntos.Add(new Puntos(100, 50, 100));
            puntos.Add(new Puntos(100, 100, 100));
            puntos.Add(new Puntos(50, 100, 100));

            //Distancia de la persona a la pantalla
            int ZPersona = 180;

            //Convierte los puntos espaciales en puntos proyectados al plano
            for (int cont = 0; cont < puntos.Count; cont++) {
                puntos[cont].Convierte3Da2D(ZPersona);
            }
        }


        //Pinta la proyección
        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics lienzo = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 3);

            lienzo.DrawLine(lapiz, puntos[0].planoX, puntos[0].planoY, puntos[1].planoX,
puntos[1].planoY);
            lienzo.DrawLine(lapiz, puntos[1].planoX, puntos[1].planoY, puntos[2].planoX,
puntos[2].planoY);
            lienzo.DrawLine(lapiz, puntos[2].planoX, puntos[2].planoY, puntos[3].planoX,
puntos[3].planoY);
            lienzo.DrawLine(lapiz, puntos[3].planoX, puntos[3].planoY, puntos[0].planoX,
puntos[0].planoY);

            lienzo.DrawLine(lapiz, puntos[4].planoX, puntos[4].planoY, puntos[5].planoX,
puntos[5].planoY);
            lienzo.DrawLine(lapiz, puntos[5].planoX, puntos[5].planoY, puntos[6].planoX,
puntos[6].planoY);
            lienzo.DrawLine(lapiz, puntos[6].planoX, puntos[6].planoY, puntos[7].planoX,
puntos[7].planoY);
            lienzo.DrawLine(lapiz, puntos[7].planoX, puntos[7].planoY, puntos[4].planoX,
puntos[4].planoY);

            lienzo.DrawLine(lapiz, puntos[0].planoX, puntos[0].planoY, puntos[4].planoX,
puntos[4].planoY);
            lienzo.DrawLine(lapiz, puntos[1].planoX, puntos[1].planoY, puntos[5].planoX,
puntos[5].planoY);
            lienzo.DrawLine(lapiz, puntos[2].planoX, puntos[2].planoY, puntos[6].planoX,
puntos[6].planoY);
            lienzo.DrawLine(lapiz, puntos[3].planoX, puntos[3].planoY, puntos[7].planoX,
puntos[7].planoY);
        }
    }
}
```

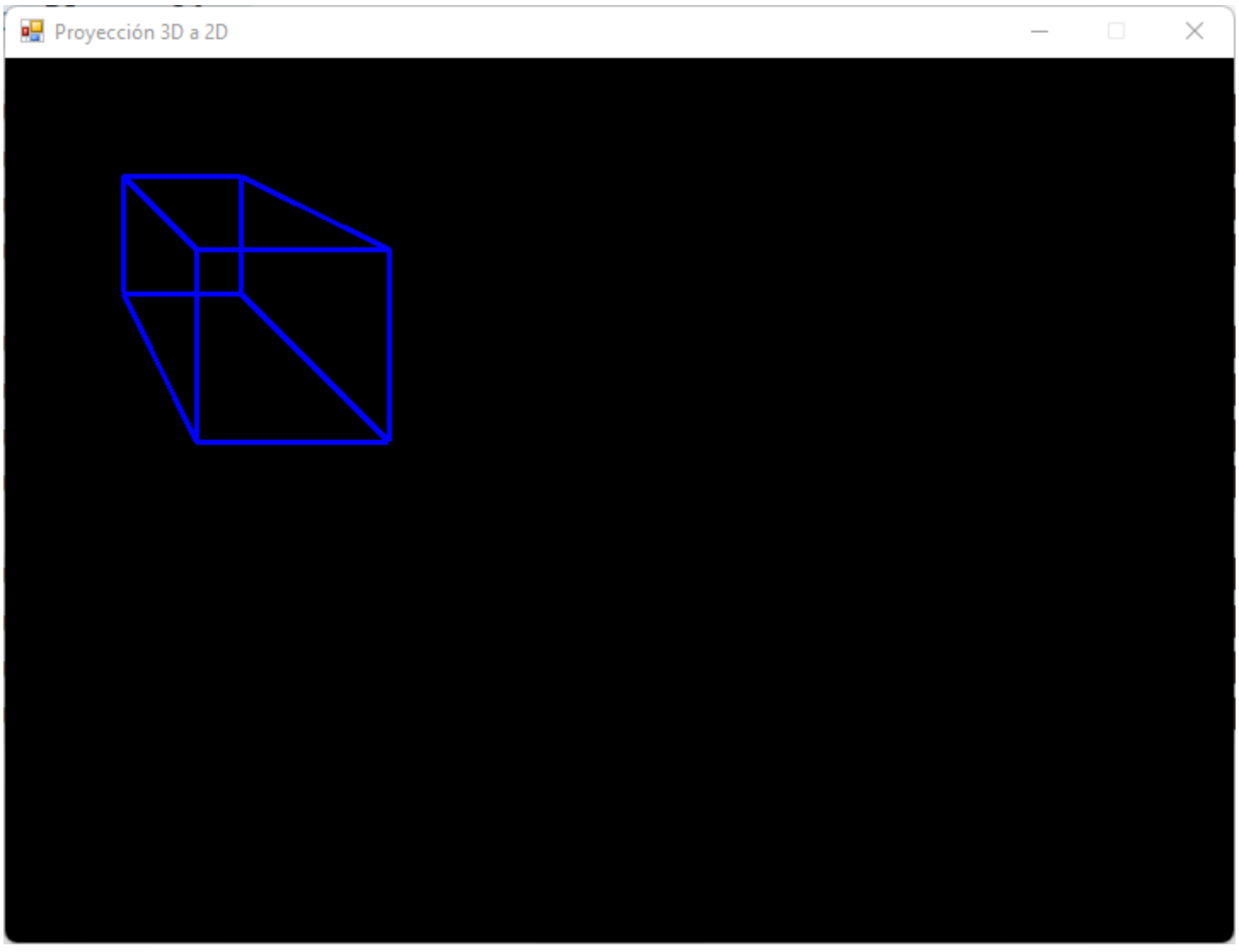


Ilustración 75: Proyección del cubo en la pantalla

Girando un objeto 3D y mostrarlo en 2D

En https://www.cs.buap.mx/~iolmos/graficacion/5_Transformaciones_geometricas_3D.pdf [15]se explican las tres rotaciones en los ejes X, Y, Z por separado. Nota: La matriz en el texto referenciado está como (columna, fila) en este libro es (columna, fila), de todos modos, las ecuaciones de giro son las mismas.

Para aplicar el giro, se requiere una matriz de transformación. Estas serían las matrices:

Matriz de Giro en X

$$\begin{bmatrix} Xgiro \\ Ygiro \\ Zgiro \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(angX) & sen(angX) \\ 0 & -sen(angX) & \cos(angX) \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Aplicándolo:

$$Xgiro = 1 * X + 0 * Y + 0 * Z$$

$$Ygiro = 0 * X + \cos(angX) * Y - sen(angX) * Z$$

$$Zgiro = 0 * X + sen(angX) * Y + \cos(angX) * Z$$

Matriz de Giro en Y

$$\begin{bmatrix} Xgiro \\ Ygiro \\ Zgiro \end{bmatrix} = \begin{bmatrix} \cos(angY) & 0 & -sen(angY) \\ 0 & 1 & 0 \\ sen(angY) & 0 & \cos(angY) \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Aplicándolo:

$$Xgiro = \cos(angY) * X + 0 * Y + sen(angY) * Z$$

$$Ygiro = 0 * X + 1 * Y + 0 * Z$$

$$Zgiro = -sen(angY) * X + 0 * Y + \cos(angY) * Z$$

Matriz de Giro en Z

$$\begin{bmatrix} Xgiro \\ Ygiro \\ Zgiro \end{bmatrix} = \begin{bmatrix} \cos(angZ) & sen(angZ) & 0 \\ -sen(angZ) & \cos(\text{angZ}) & 0 \\ 0 & 0 & 1) \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Aplicándolo:

$$Xgiro = \cos(angZ) * X - sen(angZ) * Y + 0 * Z$$

$$Ygiro = sen(angZ) * X + \cos(angZ) * Y + 0 * Z$$

$$Zgiro = 0 * X + 0 * Y + 1 * Z$$

09. 3D/02. Giros.zip/Puntos.cs

```
using System;

namespace Graficos {
    internal class Puntos {
        //Coordenada espacial original
        public double posX, posY, posZ;

        //Coordenada espacial después de girar
        public double posXg, posYg, posZg;

        //Coordenada proyectada
        public int planoX, planoY;

        //Constructor
        public Puntos(int posX, int posY, int posZ) {
```

```
        this.posX = posX;
        this.posY = posY;
        this.posZ = posZ;
    }

    //Convierte de 3D a 2D
    public void Convierte3Da2D(int ZPersona) {
        planoX = Convert.ToInt32((ZPersona * posXg) / (ZPersona - posZg));
        planoY = Convert.ToInt32((ZPersona * posYg) / (ZPersona - posZg));
    }

    //Gira en X
    public void GiroX(double angulo) {
        double ang = angulo * Math.PI / 180;

        double[,] Matriz = new double[3, 3] {
            {1, 0, 0},
            {0, Math.Cos(ang), Math.Sin(ang)},
            {0, -Math.Sin(ang), Math.Cos(ang) }
        };

        posXg = posX * Matriz[0,0] + posY * Matriz[1,0] + posZ * Matriz[2,0];
        posYg = posX * Matriz[0,1] + posY * Matriz[1,1] + posZ * Matriz[2,1];
        posZg = posX * Matriz[0,2] + posY * Matriz[1,2] + posZ * Matriz[2,2];
    }

    //Gira en Y
    public void GiroY(double angulo) {
        double ang = angulo * Math.PI / 180;

        double[,] Matriz = new double[3, 3] {
            {Math.Cos(ang), 0, -Math.Sin(ang)},
            {0, 1, 0},
            {Math.Sin(ang), 0, Math.Cos(ang) }
        };

        posXg = posX * Matriz[0, 0] + posY * Matriz[1, 0] + posZ * Matriz[2, 0];
        posYg = posX * Matriz[0, 1] + posY * Matriz[1, 1] + posZ * Matriz[2, 1];
        posZg = posX * Matriz[0, 2] + posY * Matriz[1, 2] + posZ * Matriz[2, 2];

    }

    //Gira en Z
    public void GiroZ(double angulo) {
        double ang = angulo * Math.PI / 180;

        double[,] Matriz = new double[3, 3] {
            {Math.Cos(ang), Math.Sin(ang), 0},
            {-Math.Sin(ang), Math.Cos(ang), 0},
            {0, 0, 1 }
        };

        posXg = posX * Matriz[0, 0] + posY * Matriz[1, 0] + posZ * Matriz[2, 0];
        posYg = posX * Matriz[0, 1] + posY * Matriz[1, 1] + posZ * Matriz[2, 1];
        posZg = posX * Matriz[0, 2] + posY * Matriz[1, 2] + posZ * Matriz[2, 2];
    }

  }
}
```



```
//Proyección 3D a 2D y giros
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {

        //Donde almacena los puntos
        List<Puntos> puntos;

        //Ángulos de giro
        double AnguloX, AnguloY, AnguloZ;
        int ZPersona;
```

```
        public Form1() {
            InitializeComponent();
            puntos = new List<Puntos>();

            //Coordenadas del cubo
            puntos.Add(new Puntos(50, 50, 50));
            puntos.Add(new Puntos(100, 50, 50));
            puntos.Add(new Puntos(100, 100, 50));
            puntos.Add(new Puntos(50, 100, 50));
            puntos.Add(new Puntos(50, 50, 100));
            puntos.Add(new Puntos(100, 50, 100));
            puntos.Add(new Puntos(100, 100, 100));
            puntos.Add(new Puntos(50, 100, 100));

            //Valores por defecto a los ángulos
            AnguloX = 0;
            AnguloY = 0;
            AnguloZ = 0;
            ZPersona = 180;

            Logica(0);
        }

        private void numGiroX_ValueChanged(object sender, System.EventArgs e) {
            AnguloX = Convert.ToDouble(numGiroX.Value);
            Logica(0);
            Refresh();
        }

        private void numGiroY_ValueChanged(object sender, EventArgs e) {
            AnguloY = Convert.ToDouble(numGiroY.Value);
            Logica(1);
            Refresh();
        }

        private void numGiroZ_ValueChanged(object sender, EventArgs e) {
            AnguloZ = Convert.ToDouble(numGiroZ.Value);
            Logica(2);
            Refresh();
        }

        private void numDistancia_ValueChanged(object sender, EventArgs e) {
            ZPersona = Convert.ToInt32(numDistancia.Value);
            Logica(0);
            Refresh();
        }

        public void Logica(int AnguloGira) {
            //Gira y convierte los puntos espaciales en puntos proyectados al plano
            for (int cont = 0; cont < puntos.Count; cont++) {
                switch (AnguloGira) {
                    case 0: puntos[cont].GiroX(AnguloX); break;
                    case 1: puntos[cont].GiroY(AnguloY); break;
                    case 2: puntos[cont].GiroZ(AnguloZ); break;
                }
                puntos[cont].Convierte3Da2D(ZPersona);
            }
        }


        //Pinta la proyección
        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics lienzo = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 3);

            lienzo.DrawLine(lapiz, puntos[0].planoX, puntos[0].planoY, puntos[1].planoX,
puntos[1].planoY);
            lienzo.DrawLine(lapiz, puntos[1].planoX, puntos[1].planoY, puntos[2].planoX,
puntos[2].planoY);
            lienzo.DrawLine(lapiz, puntos[2].planoX, puntos[2].planoY, puntos[3].planoX,
puntos[3].planoY);
            lienzo.DrawLine(lapiz, puntos[3].planoX, puntos[3].planoY, puntos[0].planoX,
puntos[0].planoY);

            lienzo.DrawLine(lapiz, puntos[4].planoX, puntos[4].planoY, puntos[5].planoX,
puntos[5].planoY);
            lienzo.DrawLine(lapiz, puntos[5].planoX, puntos[5].planoY, puntos[6].planoX,
puntos[6].planoY);
```

```
            lienzo.DrawLine(lapiz, puntos[6].planoX, puntos[6].planoY, puntos[7].planoX,
puntos[7].planoY);
            lienzo.DrawLine(lapiz, puntos[7].planoX, puntos[7].planoY, puntos[4].planoX,
puntos[4].planoY);

            lienzo.DrawLine(lapiz, puntos[0].planoX, puntos[0].planoY, puntos[4].planoX,
puntos[4].planoY);
            lienzo.DrawLine(lapiz, puntos[1].planoX, puntos[1].planoY, puntos[5].planoX,
puntos[5].planoY);
            lienzo.DrawLine(lapiz, puntos[2].planoX, puntos[2].planoY, puntos[6].planoX,
puntos[6].planoY);
            lienzo.DrawLine(lapiz, puntos[3].planoX, puntos[3].planoY, puntos[7].planoX,
puntos[7].planoY);
        }
    }
}
```

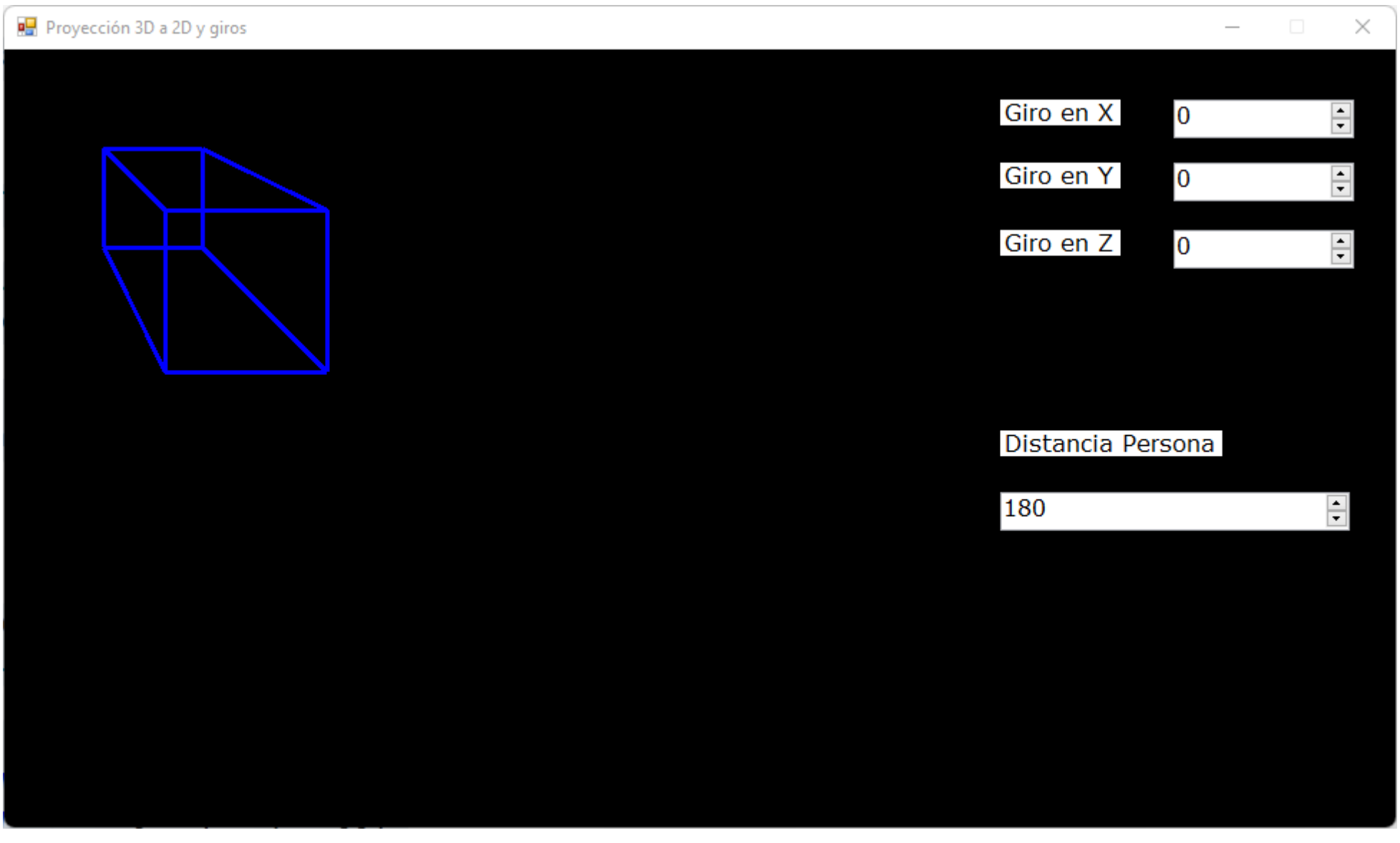


Ilustración 76: Posición del cubo original

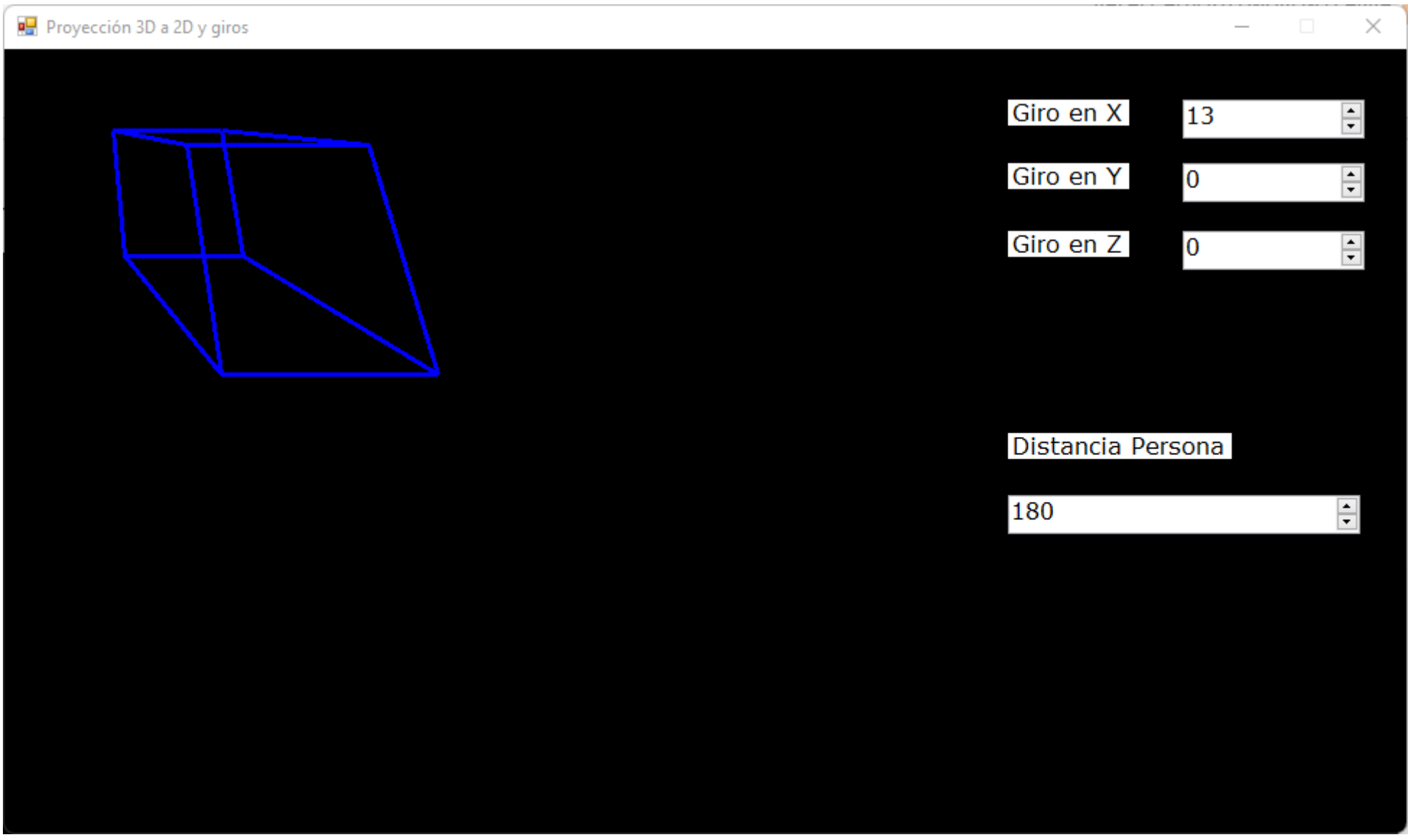


Ilustración 77: Giro en el eje X

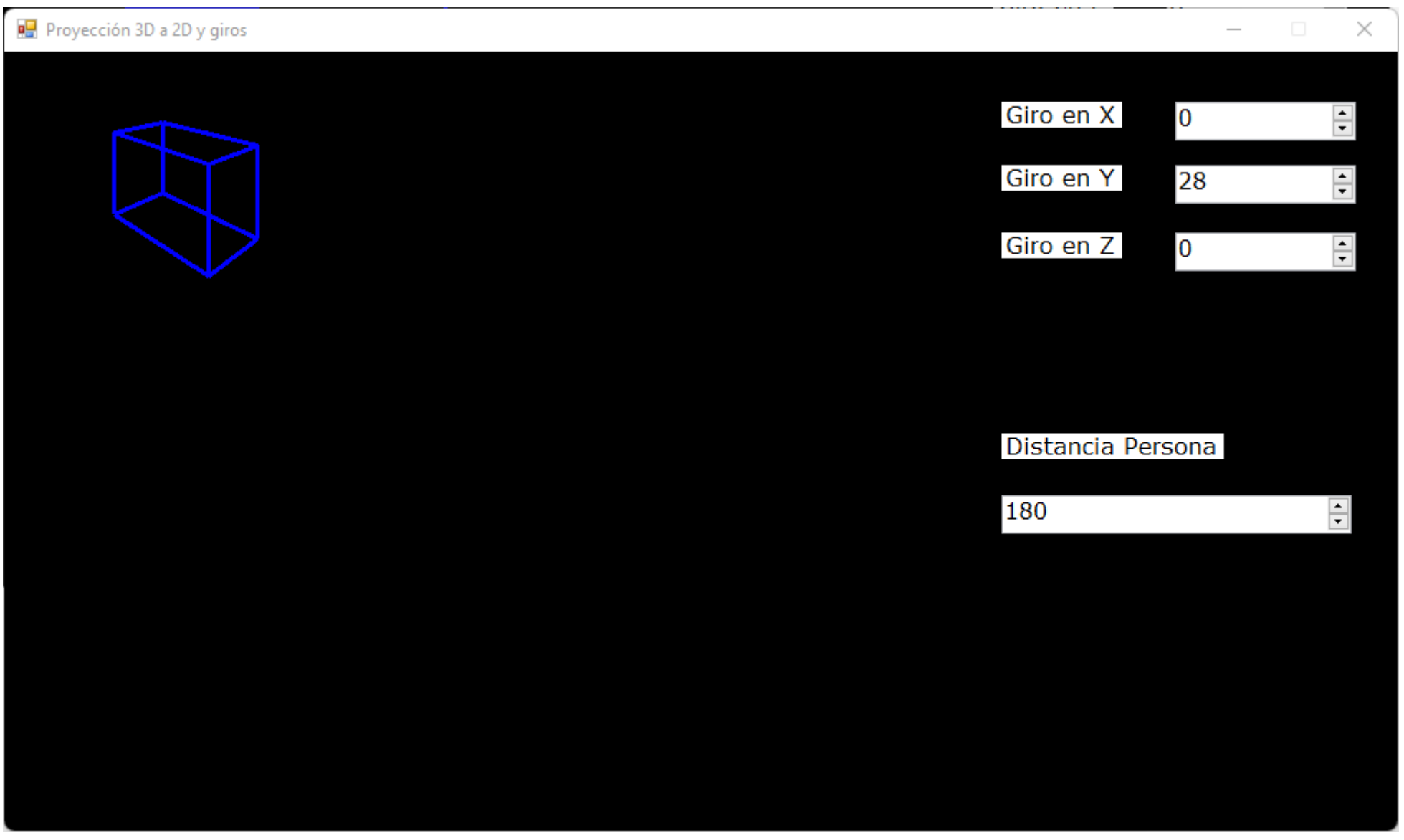


Ilustración 78: Giro en el eje Y

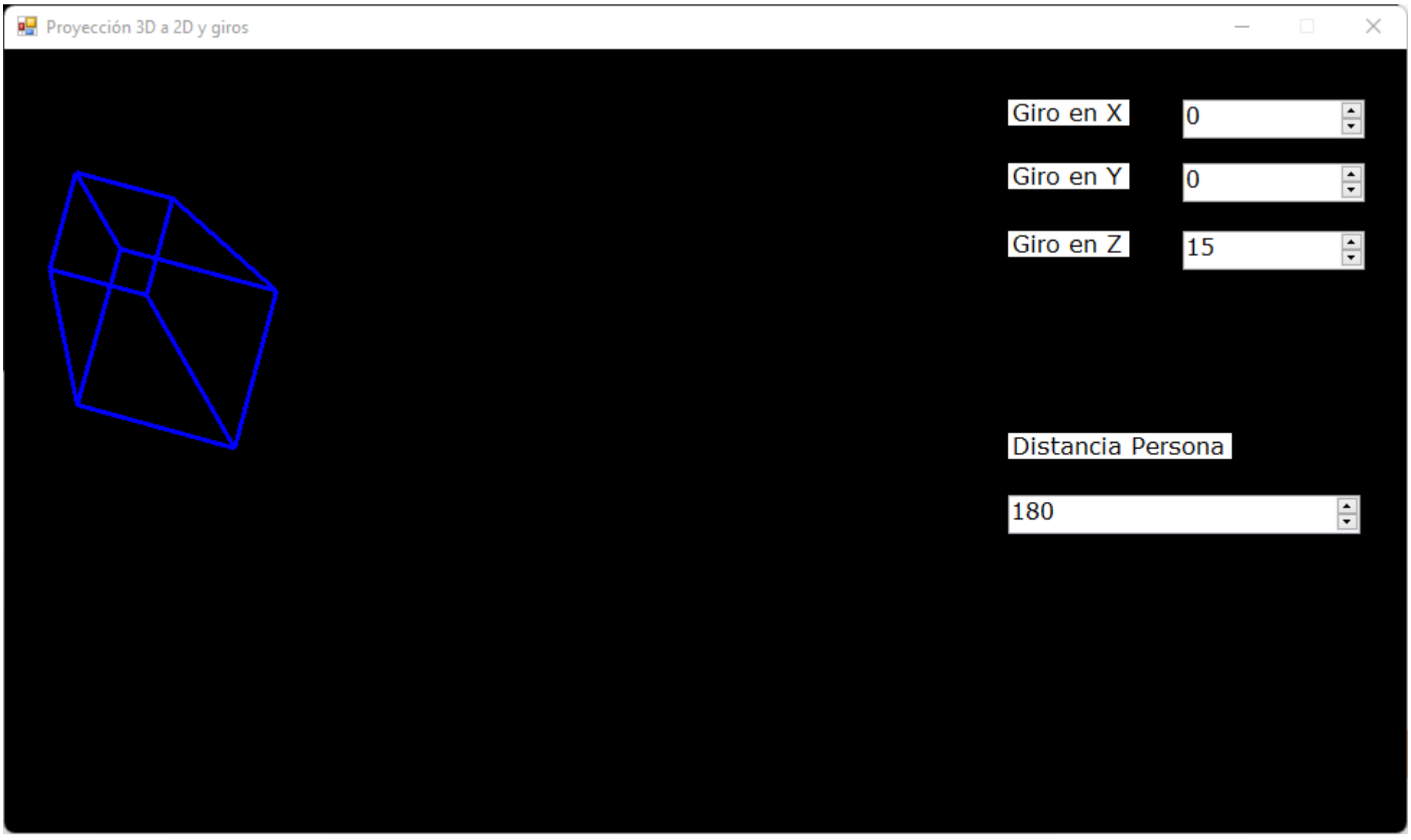


Ilustración 79: Giro en el eje Z

Giro centrado

En el ejemplo anterior, el punto (0,0,0) se encuentra en la parte superior izquierda de la ventana, el cubo está a un lado, no en el centro, por lo que girar el cubo también implica desplazarlo. Llega a un punto que se sale de la ventana y no se puede ver. Se hace un cambio al cubo para que su centro esté en la posición (0,0,0), además de mejorar la proyección en pantalla.

El pasar el cubo proyectado a la pantalla física es el que requiere más cuidado porque hay que cuadrarlo considerando la distancia del observador y los diferentes giros que puede hacer el cubo. El problema es que el cubo se puede ver bien en la posición predeterminada (ángulos en 0,0,0), pero al girarlo, se salen los vértices de los límites de la pantalla o ventana, luego hay que ajustar la conversión de plano a pantalla de tal manera que por más que lo gire en cualquier ángulo, no se salga el dibujo de la ventana. Para lograr eso, se sabe que el cubo tiene coordenadas de -0.5 a 0.5 (es decir de lado=1) tanto en X, Y, y Z. Se deja fija la distancia del observador, en este caso con un valor de 5 (suficientemente alejado de la figura). Con esos valores entonces se probaron todos los ángulos de giro en X de 0 a 360 grados, en Y de 0 a 360 grados y en Z de 0 a 360 grados, así:

Desde anguloX = 0 hasta 360 paso 1

    Desde anguloY = 0 hasta 360 paso 1

        Desde anguloZ=0 hasta 360 paso 1

Eso significa que se probaron 361*361*361 = 47.045.881 situaciones posibles. Un número muy alto que tomó tiempo en hacerlo (depende de la velocidad del computador) para saber las coordenadas máximas planas al girarlo en todos esos ángulos. Con este proceso se obtuvieron las siguientes constantes:

```
//Constantes para cuadrar el cubo en pantalla
//Se obtuvieron probando los ángulos de giro X,Y,Z de 0 a 360 y ZPersona
double XplanoMax = 0.87931543769177811;
double XplanoMin = -0.87931543769177811;
double YplanoMax = 0.87931539875237918;
double YplanoMin = -0.87931539875237918;
```

Esas constantes son las que se usan para generar las coordenadas en pantalla física.

Nota 1: Si cambia la distancia del observador al plano o cambia el tamaño de la figura 3D hay que recalcular nuevas constantes.

Nota 2: Esas constantes se usarán en futuras implementaciones.

09. 3D/03. GiroCentrado.zip/Puntos.cs

```
using System;

namespace Graficos {
    internal class Puntos {
        //Coordenada espacial original
        public double posX, posY, posZ;

        //Coordenada espacial después de girar
        public double posXg, posYg, posZg;

        //Coordenada proyectada
        public double planoX, planoY;

        //Coordenada del plano enviada a la pantalla física
        public int pantallaX, pantallaY;

        //Constructor
        public Puntos(double posX, double posY, double posZ) {
            this.posX = posX;
            this.posY = posY;
            this.posZ = posZ;
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(int ZPersona) {
            planoX = (ZPersona * posXg) / (ZPersona - posZg);
            planoY = (ZPersona * posYg) / (ZPersona - posZg);
        }

        //Gira en X
        public void GiroX(double angulo) {
            double ang = angulo * Math.PI / 180;

            double[,] Matriz = new double[3, 3] {
                {1, 0, 0},
                {0, Math.Cos(ang), Math.Sin(ang)},
                {0, -Math.Sin(ang), Math.Cos(ang) }
            };
```

```
        posXg = posX * Matriz[0,0] + posY * Matriz[1,0] + posZ * Matriz[2,0];
        posYg = posX * Matriz[0,1] + posY * Matriz[1,1] + posZ * Matriz[2,1];
        posZg = posX * Matriz[0,2] + posY * Matriz[1,2] + posZ * Matriz[2,2];
    }

    //Gira en Y
    public void GiroY(double angulo) {
        double ang = angulo * Math.PI / 180;

        double[,] Matriz = new double[3, 3] {
            {Math.Cos(ang), 0, -Math.Sin(ang)},
            {0, 1, 0},
            {Math.Sin(ang), 0, Math.Cos(ang) }
        };

        posXg = posX * Matriz[0, 0] + posY * Matriz[1, 0] + posZ * Matriz[2, 0];
        posYg = posX * Matriz[0, 1] + posY * Matriz[1, 1] + posZ * Matriz[2, 1];
        posZg = posX * Matriz[0, 2] + posY * Matriz[1, 2] + posZ * Matriz[2, 2];

    }

    //Gira en Z
    public void GiroZ(double angulo) {
        double ang = angulo * Math.PI / 180;

        double[,] Matriz = new double[3, 3] {
            {Math.Cos(ang), Math.Sin(ang), 0},
            {-Math.Sin(ang), Math.Cos(ang), 0},
            {0, 0, 1 }
        };

        posXg = posX * Matriz[0, 0] + posY * Matriz[1, 0] + posZ * Matriz[2, 0];
        posYg = posX * Matriz[0, 1] + posY * Matriz[1, 1] + posZ * Matriz[2, 1];
        posZg = posX * Matriz[0, 2] + posY * Matriz[1, 2] + posZ * Matriz[2, 2];
    }

    //Cuadra los puntos en pantalla
    public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
        pantallaX = Convert.ToInt32(ConstanteX * (planoX - XplanoMin) + pXmin);
        pantallaY = Convert.ToInt32(ConstanteY * (planoY - YplanoMin) + pYmin);
    }

}
}
```

09. 3D/03. GiroCentrado.zip/Form1.cs

```
//Proyección 3D a 2D y giros. Figura centrada.
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {

        //Donde almacena los puntos
        List<Puntos> puntos;

        //Ángulos de giro
        double AnguloX, AnguloY, AnguloZ;
        int ZPersona;

        //Datos para la pantalla
        int pXmin, pYmin, pXmax, pYmax;

        public Form1() {
            InitializeComponent();
            puntos = new List<Puntos>();

            //Coordenadas del cubo
            puntos.Add(new Puntos(-0.5, -0.5, -0.5));
            puntos.Add(new Puntos(0.5, -0.5, -0.5));
            puntos.Add(new Puntos(0.5, 0.5, -0.5));
            puntos.Add(new Puntos(-0.5, 0.5, -0.5));
            puntos.Add(new Puntos(-0.5, -0.5, 0.5));
```

```
        puntos.Add(new Puntos(0.5, -0.5, 0.5));
        puntos.Add(new Puntos(0.5, 0.5, 0.5));
        puntos.Add(new Puntos(-0.5, 0.5, 0.5));

        //Valores por defecto a los ángulos
        AnguloX = 0;
        AnguloY = 0;
        AnguloZ = 0;
        ZPersona = 5;

        //Coordenadas de pantalla
        pXmin = 20;
        pYmin = 20;
        pXmax = 500;
        pYmax = 500;

        Logica(0);
    }

    private void numGiroX_ValueChanged(object sender, System.EventArgs e) {
        AnguloX = Convert.ToDouble(numGiroX.Value);
        numGiroY.Value = 0;
        numGiroZ.Value = 0;
        Logica(0);
        Refresh();
    }

    private void numGiroY_ValueChanged(object sender, EventArgs e) {
        AnguloY = Convert.ToDouble(numGiroY.Value);
        numGiroX.Value = 0;
        numGiroZ.Value = 0;
        Logica(1);
        Refresh();
    }

    private void numGiroZ_ValueChanged(object sender, EventArgs e) {
        AnguloZ = Convert.ToDouble(numGiroZ.Value);
        numGiroX.Value = 0;
        numGiroY.Value = 0;
        Logica(2);
        Refresh();
    }

    public void Logica(int AnguloGira) {
        //Gira y convierte los puntos espaciales en puntos proyectados al plano
        for (int cont = 0; cont < puntos.Count; cont++) {
            switch (AnguloGira) {
                case 0: puntos[cont].GiroX(AnguloX); break;
                case 1: puntos[cont].GiroY(AnguloY); break;
                case 2: puntos[cont].GiroZ(AnguloZ); break;
            }
            puntos[cont].Convierte3Da2D(ZPersona);
        }

        //Constantes para cuadrar el cubo en pantalla
        //Se obtuvieron probando los ángulos de giro X,Y,Z de 0 a 360 y ZPersona
        double XplanoMax = 0.87931543769177811;
        double XplanoMin = -0.87931543769177811;
        double YplanoMax = 0.87931539875237918;
        double YplanoMin = -0.87931539875237918;

        //Cuadra los puntos en pantalla
        double ConstanteX = (pXmax - pXmin) / (XplanoMax - XplanoMin);
        double ConstanteY = (pYmax - pYmin) / (YplanoMax - YplanoMin);

        for (int cont = 0; cont < puntos.Count; cont++)
            puntos[cont].CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);

    }


    //Pinta la proyección
    private void Form1_Paint(object sender, PaintEventArgs e) {
        Graphics lienzo = e.Graphics;
        Pen lapiz = new Pen(Color.Black, 1);

        lienzo.DrawLine(lapiz, puntos[0].pantallaX, puntos[0].pantallaY, puntos[4].pantallaX,
puntos[4].pantallaY);
```

```
            lienzo.DrawLine(lapiz, puntos[1].pantallaX, puntos[1].pantallaY, puntos[5].pantallaX,
puntos[5].pantallaY);
            lienzo.DrawLine(lapiz, puntos[2].pantallaX, puntos[2].pantallaY, puntos[6].pantallaX,
puntos[6].pantallaY);
            lienzo.DrawLine(lapiz, puntos[3].pantallaX, puntos[3].pantallaY, puntos[7].pantallaX,
puntos[7].pantallaY);

            lienzo.DrawLine(lapiz, puntos[0].pantallaX, puntos[0].pantallaY, puntos[1].pantallaX,
puntos[1].pantallaY);
            lienzo.DrawLine(lapiz, puntos[1].pantallaX, puntos[1].pantallaY, puntos[2].pantallaX,
puntos[2].pantallaY);
            lienzo.DrawLine(lapiz, puntos[2].pantallaX, puntos[2].pantallaY, puntos[3].pantallaX,
puntos[3].pantallaY);
            lienzo.DrawLine(lapiz, puntos[3].pantallaX, puntos[3].pantallaY, puntos[0].pantallaX,
puntos[0].pantallaY);

            lienzo.DrawLine(lapiz, puntos[4].pantallaX, puntos[4].pantallaY, puntos[5].pantallaX,
puntos[5].pantallaY);
            lienzo.DrawLine(lapiz, puntos[5].pantallaX, puntos[5].pantallaY, puntos[6].pantallaX,
puntos[6].pantallaY);
            lienzo.DrawLine(lapiz, puntos[6].pantallaX, puntos[6].pantallaY, puntos[7].pantallaX,
puntos[7].pantallaY);
            lienzo.DrawLine(lapiz, puntos[7].pantallaX, puntos[7].pantallaY, puntos[4].pantallaX,
puntos[4].pantallaY);

        }
    }
}
```

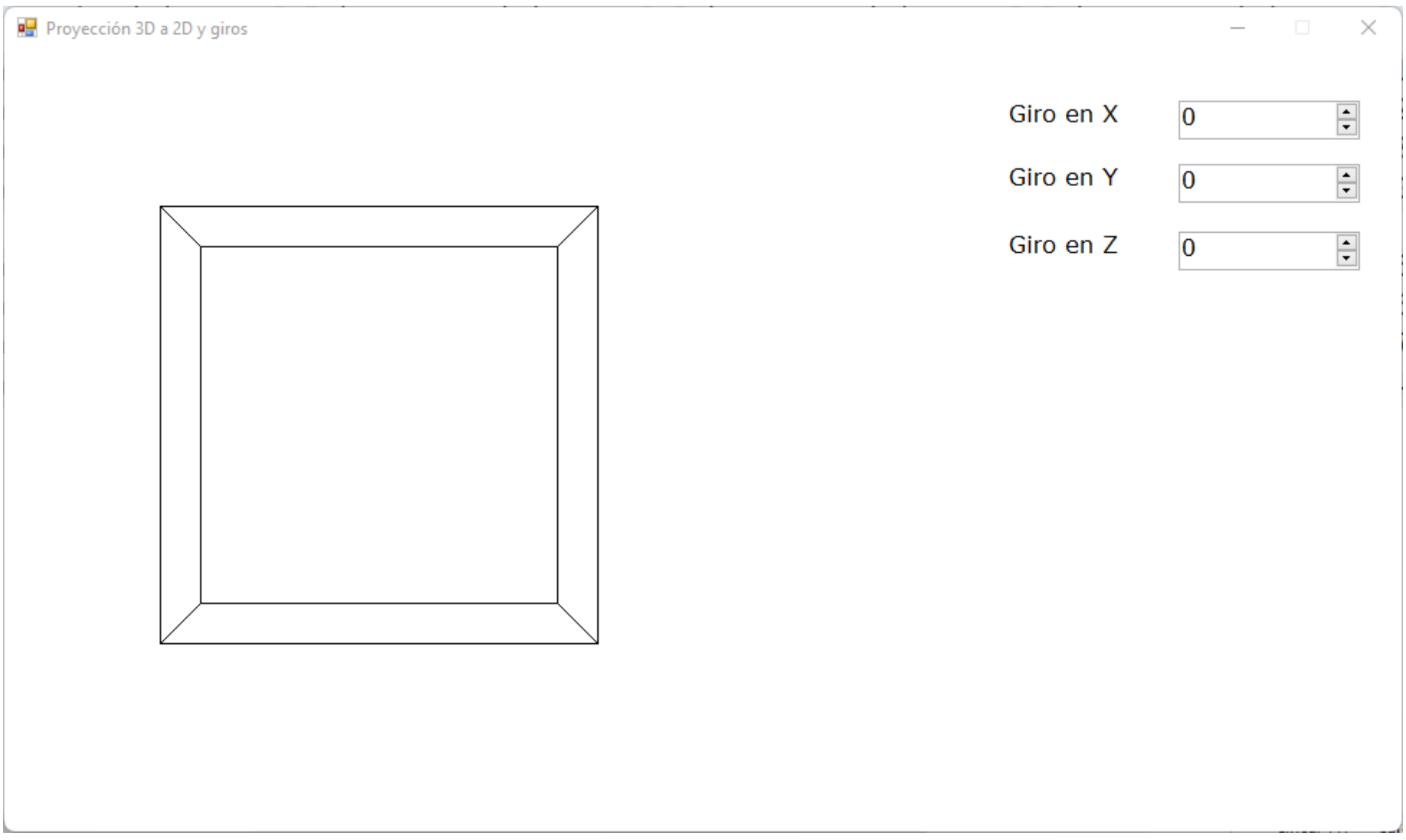


Ilustración 80: Posición inicial

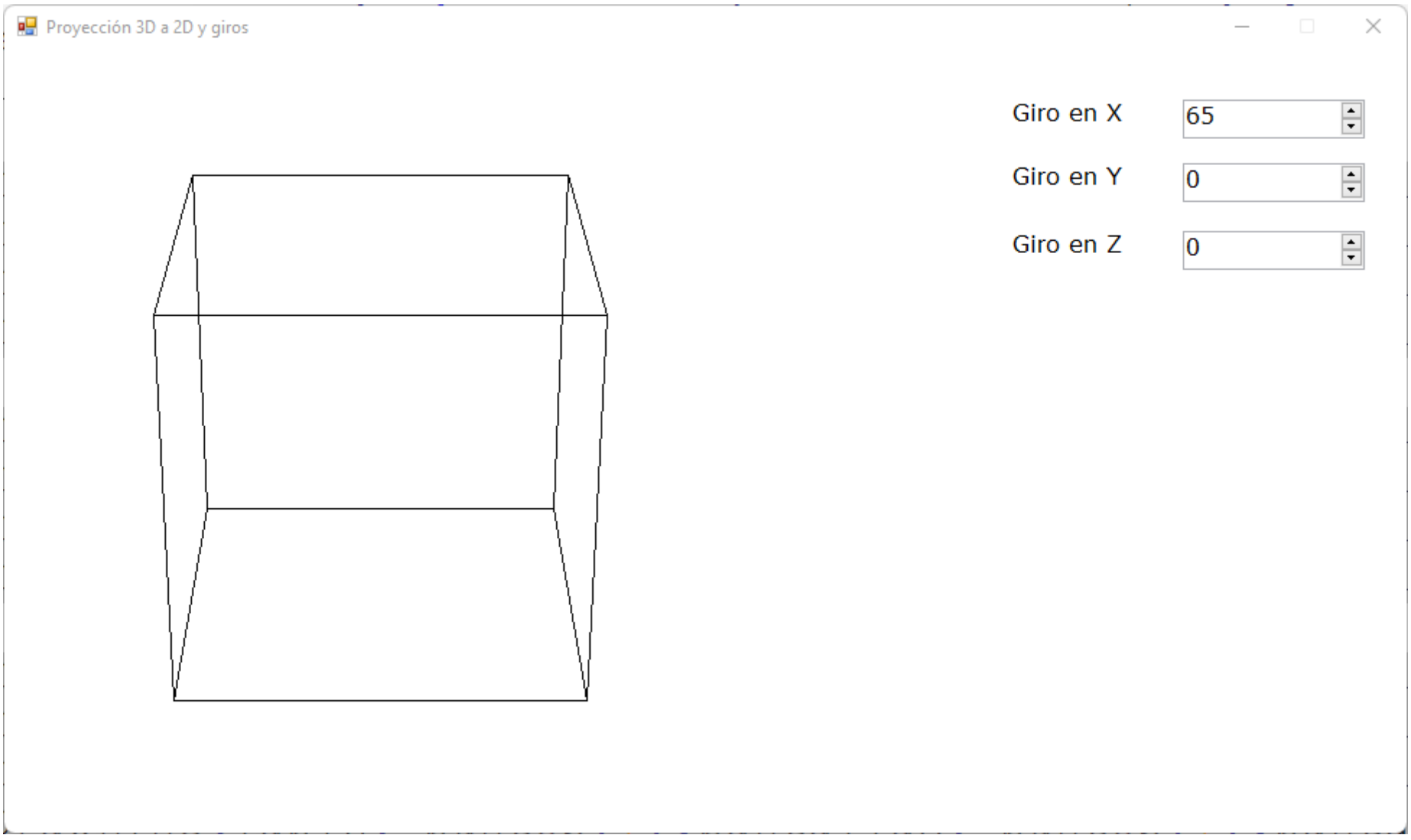


Ilustración 81: Giro en X

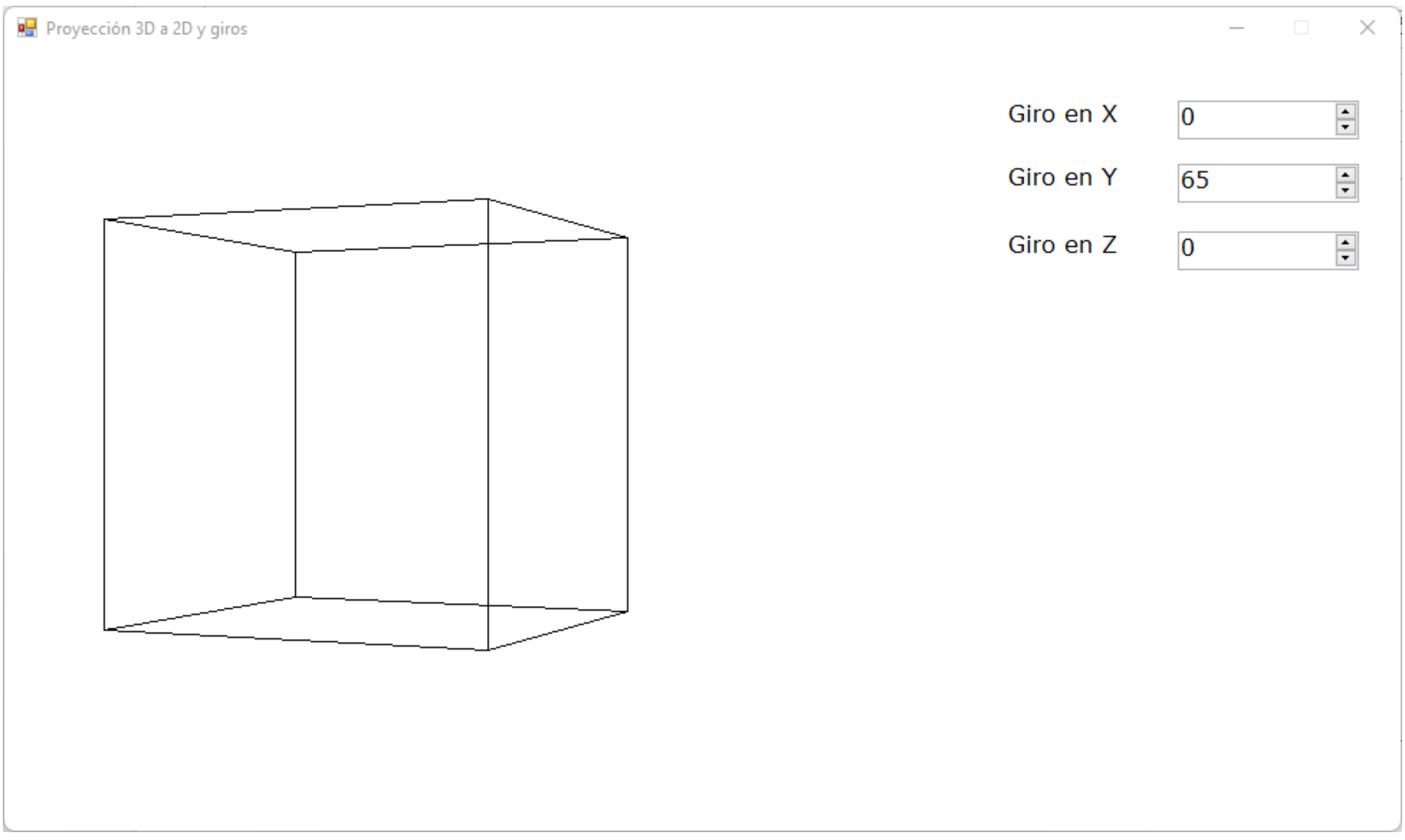


Ilustración 82: Giro en Y

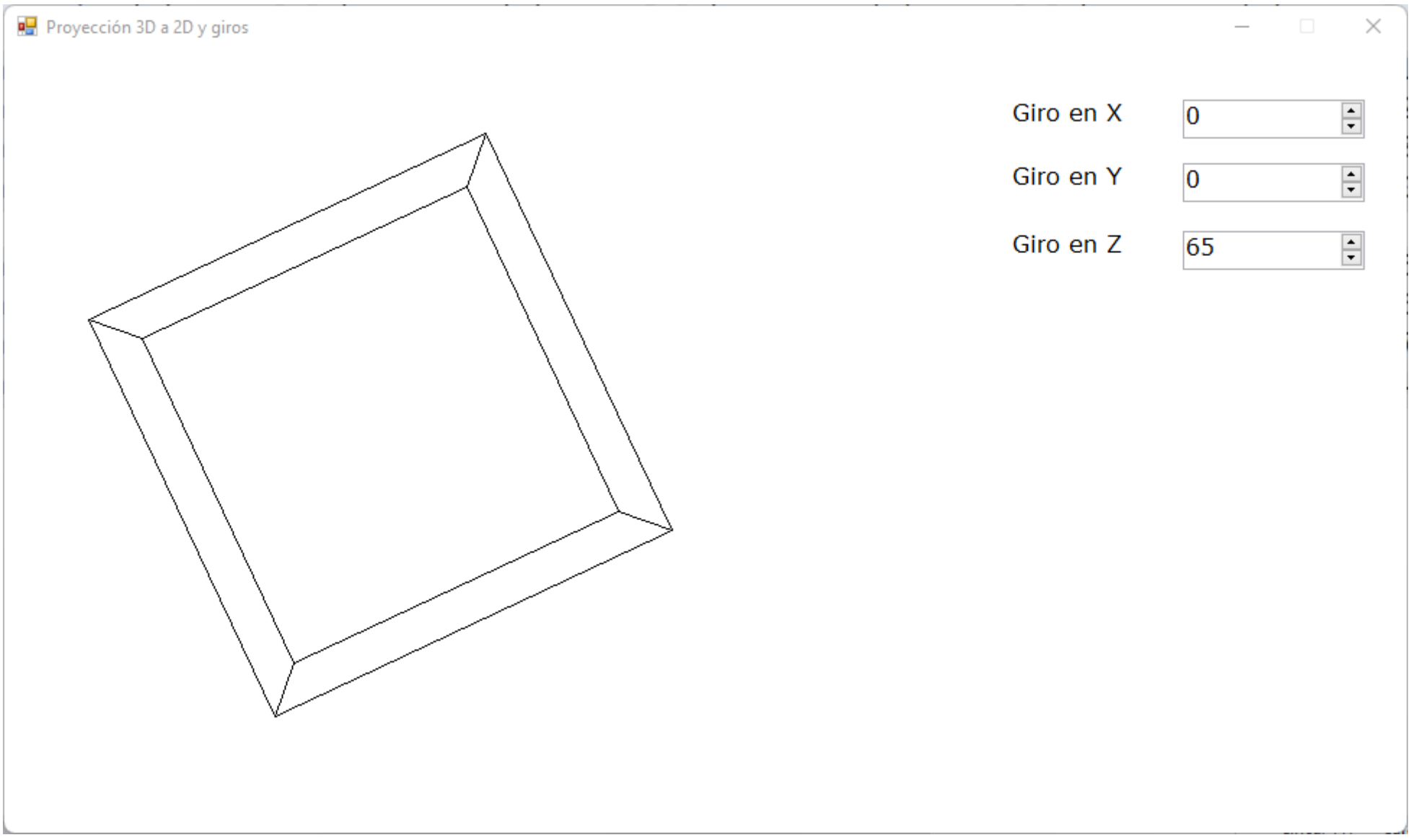


Ilustración 83: Giro en Z

Combinando los tres giros en X, Y, Z

En el programa anterior, sólo se puede girar en un ángulo, no hay forma de girarlo en los tres ángulos. ¿Cómo lograr el giro en los tres ángulos? La respuesta está en multiplicar las tres matrices de giro y con la matriz resultante se procede a hacer los giros.

$$CosX = \cos(angX)$$
$$SenX = sen(angX)$$
$$CosY = \cos(angY)$$
$$SenY = sen(angY)$$
$$CosZ = \cos(angZ)$$
$$SenZ = sen(angZ)$$

$$\begin{bmatrix} Xgiro \\ Ygiro \\ Zgiro \end{bmatrix} = \begin{bmatrix} \mathrm{CosY} * \mathrm{CosZ} & -\mathrm{CosX} * \mathrm{SinZ} + \mathrm{SinX} * \mathrm{SinY} * \mathrm{CosZ} & \mathrm{SinX} * \mathrm{SinZ} + \mathrm{CosX} * \mathrm{SinY} * \mathrm{CosZ} \\ \mathrm{CosY} * \mathrm{SinZ} & \mathrm{CosX} * \mathrm{CosZ} + \mathrm{SinX} * \mathrm{SinY} * \mathrm{SinZ} & -\mathrm{SinX} * \mathrm{CosZ} + \mathrm{CosX} * \mathrm{SinY} * \mathrm{SinZ} \\ -\mathrm{SinY} & \mathrm{SinX} * \mathrm{CosY} & \mathrm{CosX} * \mathrm{CosY} \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Los pasos para dibujar el cubo fueron los siguientes:

Paso 1: Coordenadas del cubo, donde su centro esté en la posición 0, 0, 0. Las coordenadas van de -0.5 a 0.5, luego cada lado mide 1. Cada coordenada tiene valores posX, posY, posZ

Paso 2: Se gira cada coordenada del cubo usando la matriz de giro XYZ. Cada coordenada girada es posXg, posYg, posZg.

Paso 3: Se proyecta la coordenada girada a un plano XY, el valor Z queda en cero. Esa coordenada plana es planoX, planoY

Paso 4: Ahora se debe cuadrar esa coordenada planoX, planoY (real y algunas veces con valores negativos) para que se proyecte en una pantalla física (valores enteros, limitados y positivos). Y como se mencionó antes requiere del cálculo de unas constantes:

```
//Constantes para cuadrar el cubo en pantalla
//Se obtuvieron probando los ángulos de giro X,Y,Z de 0 a 360 y ZPersona
double XplanoMax = 0.87931543769177811;
double XplanoMin = -0.87931543769177811;
double YplanoMax = 0.87931539875237918;
double YplanoMin = -0.87931539875237918;
```

Esas constantes son las que se usan para generar las coordenadas en pantalla física.

Nota 1: Si cambia la distancia del observador al plano o cambia el tamaño de la figura 3D hay que recalcular nuevas constantes.

Nota 2: Esas constantes se usarán en futuras implementaciones.

09. 3D/04. GiroXYZ.zip/Puntos.cs

```
using System;

namespace Graficos {
    internal class Puntos {
        //Coordenada espacial original
        public double posX, posY, posZ;

        //Coordenada espacial después de girar
        public double posXg, posYg, posZg;

        //Coordenada proyectada en un plano
        public double planoX, planoY;

        //Coordenada del plano enviada a la pantalla física
        public int pantallaX, pantallaY;

        //Constructor
        public Puntos(double posX, double posY, double posZ) {
            this.posX = posX;
            this.posY = posY;
            this.posZ = posZ;
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            planoX = posXg * ZPersona / (ZPersona - posZg);
            planoY = posYg * ZPersona / (ZPersona - posZg);
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
```

```
            double angXr = angX * Math.PI / 180;
            double angYr = angY * Math.PI / 180;
            double angZr = angZ * Math.PI / 180;

            double CosX = Math.Cos(angXr);
            double SinX = Math.Sin(angXr);
            double CosY = Math.Cos(angYr);
            double SinY = Math.Sin(angYr);
            double CosZ = Math.Cos(angZr);
            double SinZ = Math.Sin(angZr);

            //Matriz de Rotación
            //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
            double[,] Matriz = new double[3, 3] {
                { CosY * CosZ, -CosX * SinZ + SinX * SinY * CosZ, SinX * SinZ + CosX * SinY * CosZ},
                { CosY * SinZ, CosX * CosZ + SinX * SinY * SinZ, -SinX * CosZ + CosX * SinY * SinZ},
                {-SinY, SinX * CosY, CosX * CosY }
            };

            posXg = posX * Matriz[0, 0] + posY * Matriz[1, 0] + posZ * Matriz[2, 0];
            posYg = posX * Matriz[0, 1] + posY * Matriz[1, 1] + posZ * Matriz[2, 1];
            posZg = posX * Matriz[0, 2] + posY * Matriz[1, 2] + posZ * Matriz[2, 2];
        }

        //Cuadra los puntos en pantalla
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            pantallaX = Convert.ToInt32(ConstanteX * (planoX - XplanoMin) + pXmin);
            pantallaY = Convert.ToInt32(ConstanteY * (planoY - YplanoMin) + pYmin);
        }
    }
}
```

09. 3D/04. GiroXYZ.zip/Form1.cs

```
//Proyección 3D a 2D y giros en los tres ejes simultáneamente
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {

        //Donde almacena los puntos
        List<Puntos> puntos;

        //Ángulos de giro
        double AnguloX, AnguloY, AnguloZ;
        double ZPersona;

        //Datos para la pantalla
        int pXmin, pYmin, pXmax, pYmax;

        public Form1() {
            InitializeComponent();
            puntos = new List<Puntos>();

            //Coordenadas del cubo
            puntos.Add(new Puntos(-0.5, -0.5, -0.5));
            puntos.Add(new Puntos(0.5, -0.5, -0.5));
            puntos.Add(new Puntos(0.5, 0.5, -0.5));
            puntos.Add(new Puntos(-0.5, 0.5, -0.5));
            puntos.Add(new Puntos(-0.5, -0.5, 0.5));
            puntos.Add(new Puntos(0.5, -0.5, 0.5));
            puntos.Add(new Puntos(0.5, 0.5, 0.5));
            puntos.Add(new Puntos(-0.5, 0.5, 0.5));

            //Valores por defecto a los ángulos
            AnguloX = (double)numGiroX.Value;
            AnguloY = (double)numGiroY.Value;
            AnguloZ = (double)numGiroZ.Value;
            ZPersona = 5;

            //Coordenadas de pantalla
            pXmin = 20;
            pYmin = 20;
            pXmax = 500;
```

```
        pYmax = 500;

        Logica();
    }

    private void numGiroX_ValueChanged(object sender, System.EventArgs e) {
        AnguloX = Convert.ToDouble(numGiroX.Value);
        Logica();
        Refresh();
    }

    private void numGiroY_ValueChanged(object sender, EventArgs e) {
        AnguloY = Convert.ToDouble(numGiroY.Value);
        Logica();
        Refresh();
    }

    private void numGiroZ_ValueChanged(object sender, EventArgs e) {
        AnguloZ = Convert.ToDouble(numGiroZ.Value);
        Logica();
        Refresh();
    }

    public void Logica() {
        //Gira y convierte los puntos espaciales en puntos proyectados al plano
        for (int cont = 0; cont < puntos.Count; cont++) {
            puntos[cont].GiroXYZ(AnguloX, AnguloY, AnguloZ);
            puntos[cont].Convierte3Da2D(ZPersona);
        }

        //Constantes para cuadrar el cubo en pantalla
        //Se obtuvieron probando los ángulos de giro X,Y,Z de 0 a 360 y ZPersona
        double XplanoMax = 0.87931543769177811;
        double XplanoMin = -0.87931543769177811;
        double YplanoMax = 0.87931539875237918;
        double YplanoMin = -0.87931539875237918;

        //Cuadra los puntos en pantalla
        double ConstanteX = (pXmax - pXmin) / (XplanoMax - XplanoMin);
        double ConstanteY = (pYmax - pYmin) / (YplanoMax - YplanoMin);

        for (int cont = 0; cont < puntos.Count; cont++)
            puntos[cont].CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
    }


    //Pinta la proyección
    private void Form1_Paint(object sender, PaintEventArgs e) {
        Graphics lienzo = e.Graphics;
        Pen lapiz = new Pen(Color.Black, 1);

        lienzo.DrawLine(lapiz, puntos[0].pantallaX, puntos[0].pantallaY, puntos[4].pantallaX,
puntos[4].pantallaY);
        lienzo.DrawLine(lapiz, puntos[1].pantallaX, puntos[1].pantallaY, puntos[5].pantallaX,
puntos[5].pantallaY);
        lienzo.DrawLine(lapiz, puntos[2].pantallaX, puntos[2].pantallaY, puntos[6].pantallaX,
puntos[6].pantallaY);
        lienzo.DrawLine(lapiz, puntos[3].pantallaX, puntos[3].pantallaY, puntos[7].pantallaX,
puntos[7].pantallaY);

        lienzo.DrawLine(lapiz, puntos[0].pantallaX, puntos[0].pantallaY, puntos[1].pantallaX,
puntos[1].pantallaY);
        lienzo.DrawLine(lapiz, puntos[1].pantallaX, puntos[1].pantallaY, puntos[2].pantallaX,
puntos[2].pantallaY);
        lienzo.DrawLine(lapiz, puntos[2].pantallaX, puntos[2].pantallaY, puntos[3].pantallaX,
puntos[3].pantallaY);
        lienzo.DrawLine(lapiz, puntos[3].pantallaX, puntos[3].pantallaY, puntos[0].pantallaX,
puntos[0].pantallaY);

        lienzo.DrawLine(lapiz, puntos[4].pantallaX, puntos[4].pantallaY, puntos[5].pantallaX,
puntos[5].pantallaY);
        lienzo.DrawLine(lapiz, puntos[5].pantallaX, puntos[5].pantallaY, puntos[6].pantallaX,
puntos[6].pantallaY);
        lienzo.DrawLine(lapiz, puntos[6].pantallaX, puntos[6].pantallaY, puntos[7].pantallaX,
puntos[7].pantallaY);
        lienzo.DrawLine(lapiz, puntos[7].pantallaX, puntos[7].pantallaY, puntos[4].pantallaX,
puntos[4].pantallaY);
    }
}
```

```
}
```

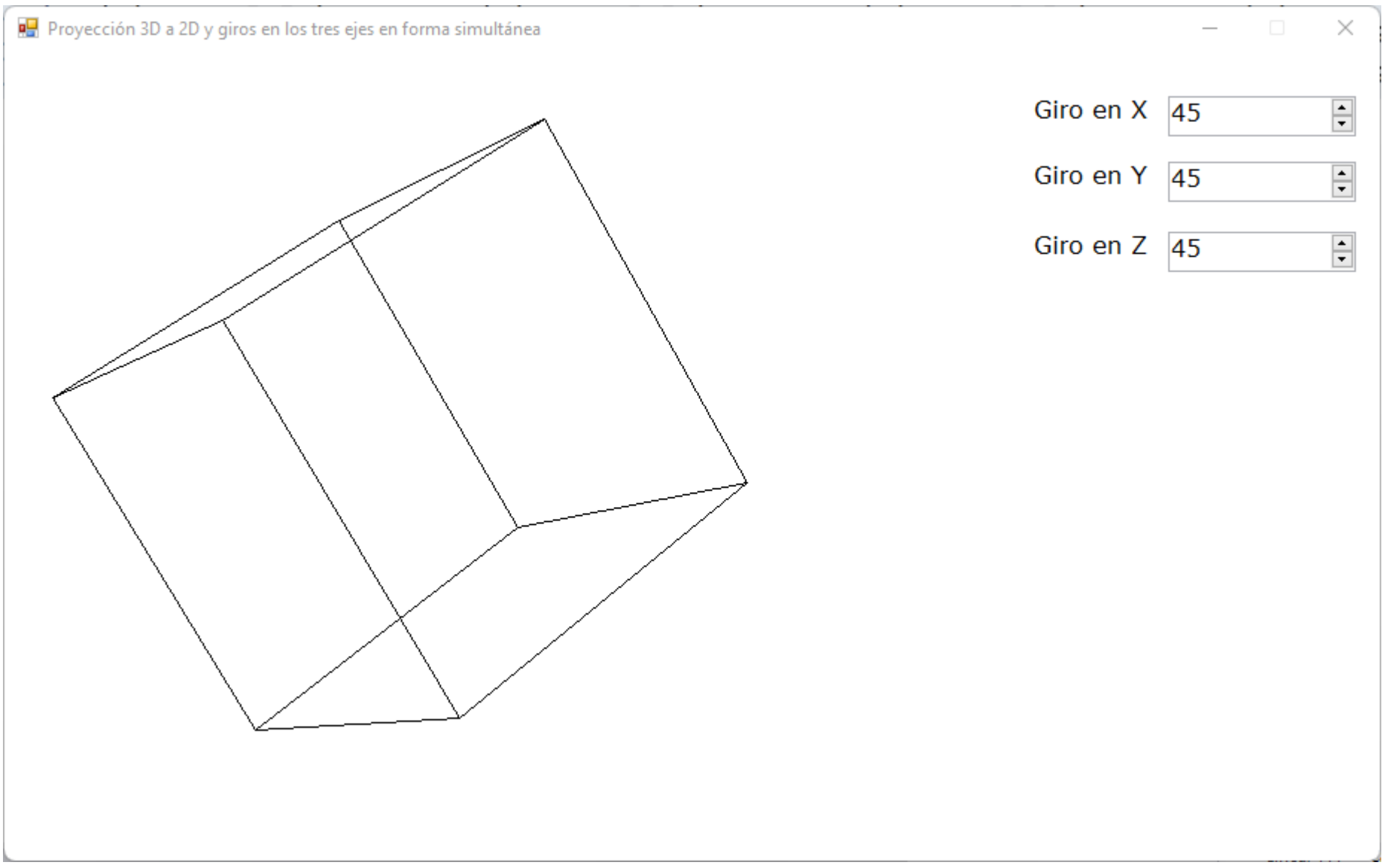


Ilustración 84: Posición inicial

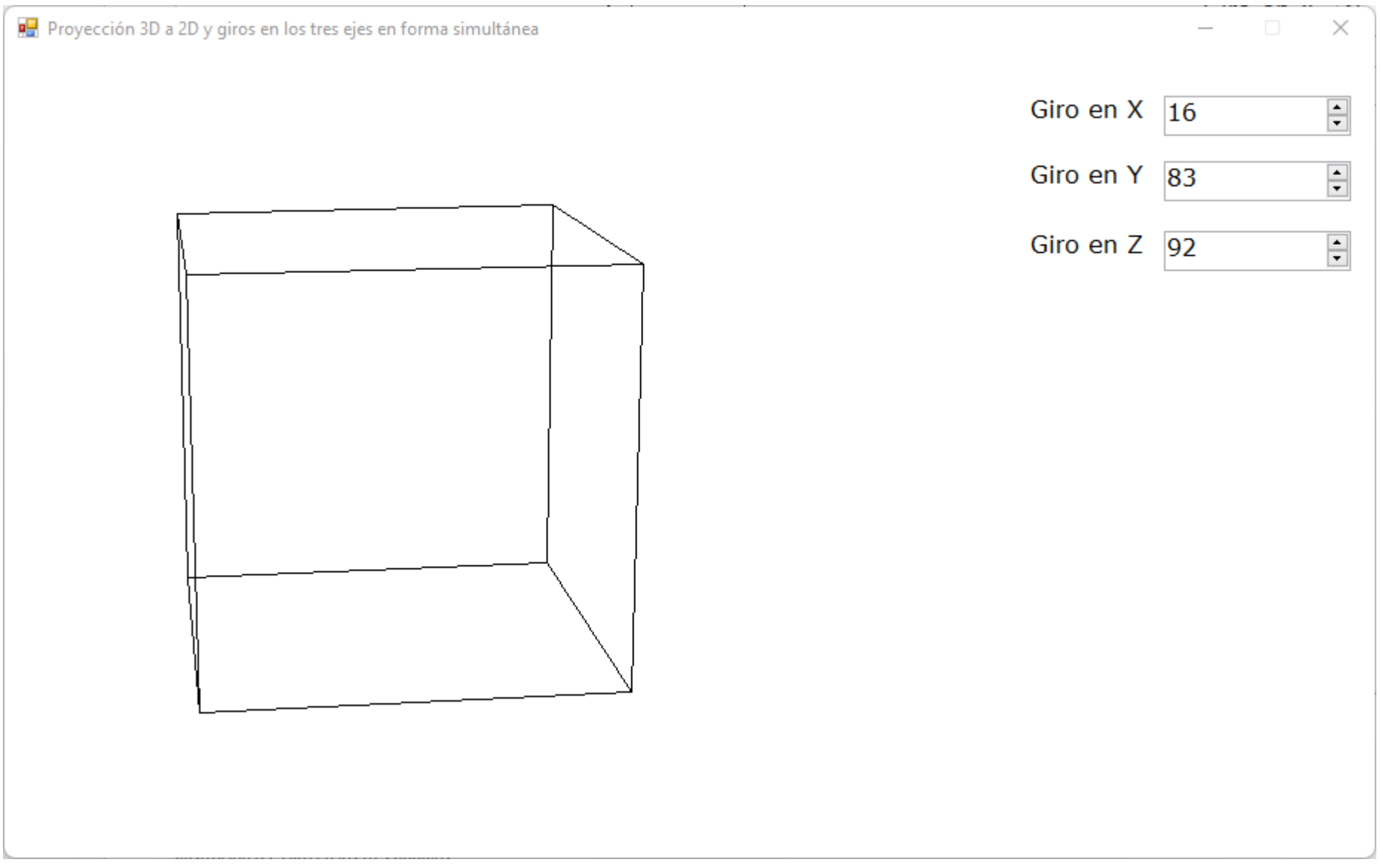


Ilustración 85: Giro en los tres ángulos

Líneas ocultas

El problema con los ejemplos anteriores de giros es nuestra vista, en vez de interpretar que el cubo gira, lo que a veces vemos es que la figura se distorsiona y es porque el cubo hecho de líneas verticales y horizontales nos confunde, y no sabemos que está atrás y que está adelante. Para evitar esto, se hace un cambio y es dibujar el cubo no con líneas sino con polígonos [16]. Un cubo tiene seis caras, luego serían seis polígonos para dibujarlo. Estos serían los pasos:

1. Escribir las cuatro coordenadas XYZ que servirán para componer cada uno de los seis polígonos.
2. Girar cada coordenada XYZ de cada polígono.
3. Determinar el valor Z promedio de cada polígono.
4. Ordenar los polígonos del Z promedio más pequeño al más grande. Eso nos daría que polígono está más lejos y cuál más cerca al observador.
5. Dibujar en ese orden cada polígono, primero rellenando el área del polígono con el color del fondo y luego dibujando el perímetro con algún color visible.

09. 3D/05. PoligonoOculto.zip/Puntos.cs

```
using System;

namespace Graficos {
    internal class Puntos {
        //Coordenada espacial original
        public double posX, posY, posZ;

        //Coordenada espacial después de girar
        public double posXg, posYg, posZg;

        //Coordenada proyectada
        public double planoX, planoY;

        //Coordenada en la pantalla
        public int pantallaX, pantallaY;

        //Constructor
        public Puntos(double posX, double posY, double posZ) {
            this.posX = posX;
            this.posY = posY;
            this.posZ = posZ;
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
            double angXr = angX * Math.PI / 180;
            double angYr = angY * Math.PI / 180;
            double angZr = angZ * Math.PI / 180;

            double CosX = Math.Cos(angXr);
            double SinX = Math.Sin(angXr);
            double CosY = Math.Cos(angYr);
            double SinY = Math.Sin(angYr);
            double CosZ = Math.Cos(angZr);
            double SinZ = Math.Sin(angZr);

            //Matriz de Rotación
            //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
            double[,] Matriz = new double[3, 3] {
                { CosY * CosZ, -CosX * SinZ + SinX * SinY * CosZ, SinX * SinZ + CosX * SinY * CosZ},
                { CosY * SinZ, CosX * CosZ + SinX * SinY * SinZ, -SinX * CosZ + CosX * SinY * SinZ},
                {-SinY, SinX * CosY, CosX * CosY }
            };

            posXg = posX * Matriz[0, 0] + posY * Matriz[1, 0] + posZ * Matriz[2, 0];
            posYg = posX * Matriz[0, 1] + posY * Matriz[1, 1] + posZ * Matriz[2, 1];
            posZg = posX * Matriz[0, 2] + posY * Matriz[1, 2] + posZ * Matriz[2, 2];
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            planoX = posXg * ZPersona / (ZPersona - posZg);
            planoY = posYg * ZPersona / (ZPersona - posZg);
        }

        //Cuadra los puntos en pantalla
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            pantallaX = Convert.ToInt32(ConstanteX * (planoX - XplanoMin) + pXmin);
            pantallaY = Convert.ToInt32(ConstanteY * (planoY - YplanoMin) + pYmin);
```

```
        }
    }
}
```

09. 3D/05. PoligonoOculto.zip/Poligono.cs

```
using System;
using System.Drawing;

namespace Graficos {
    internal class Poligono : IComparable {
        //Un polígono son cuatro(4) coordenadas espaciales
        public Puntos coordenadas1, coordenadas2, coordenadas3, coordenadas4;
        private double centro;

        public Poligono(double X1, double Y1, double Z1, double X2, double Y2, double Z2, double X3,
double Y3, double Z3, double X4, double Y4, double Z4) {
            coordenadas1 = new Puntos(X1, Y1, Z1);
            coordenadas2 = new Puntos(X2, Y2, Z2);
            coordenadas3 = new Puntos(X3, Y3, Z3);
            coordenadas4 = new Puntos(X4, Y4, Z4);
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
            coordenadas1.GiroXYZ(angX, angY, angZ);
            coordenadas2.GiroXYZ(angX, angY, angZ);
            coordenadas3.GiroXYZ(angX, angY, angZ);
            coordenadas4.GiroXYZ(angX, angY, angZ);
            centro = (coordenadas1.posZg + coordenadas2.posZg + coordenadas3.posZg + coordenadas4.posZg) /
4;
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            coordenadas1.Convierte3Da2D(ZPersona);
            coordenadas2.Convierte3Da2D(ZPersona);
            coordenadas3.Convierte3Da2D(ZPersona);
            coordenadas4.Convierte3Da2D(ZPersona);
        }

        //Cuadra los poligonos en pantalla
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            coordenadas1.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas2.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas3.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas4.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
        }

        //Hace el gráfico del polígono
        public void Dibuja(Graphics lienzo, Pen lapiz, Brush relleno) {
            //Pone un color de fondo al polígono para borrar lo que hay detrás
            Point punto1 = new Point(coordenadas1.pantallaX, coordenadas1.pantallaY);
            Point punto2 = new Point(coordenadas2.pantallaX, coordenadas2.pantallaY);
            Point punto3 = new Point(coordenadas3.pantallaX, coordenadas3.pantallaY);
            Point punto4 = new Point(coordenadas4.pantallaX, coordenadas4.pantallaY);
            Point[] poligono = { punto1, punto2, punto3, punto4 };
            lienzo.FillPolygon(relleno, poligono);

            //Pinta el perímetro del polígono
            lienzo.DrawLine(lapiz, coordenadas1.pantallaX, coordenadas1.pantallaY, coordenadas2.pantallaX,
coordenadas2.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas2.pantallaX, coordenadas2.pantallaY, coordenadas3.pantallaX,
coordenadas3.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas3.pantallaX, coordenadas3.pantallaY, coordenadas4.pantallaX,
coordenadas4.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas4.pantallaX, coordenadas4.pantallaY, coordenadas1.pantallaX,
coordenadas1.pantallaY);
        }

        public int CompareTo(object obj) {
            Poligono orderToCompare = obj as Poligono;
            if (orderToCompare.centro < centro) {
                return 1;
            }
            if (orderToCompare.centro > centro) {
```

```
                return -1;
            }

            // The orders are equivalent.
            //https://stackoverflow.com/questions/3309188/how-to-sort-a-listt-by-a-property-in-the-object
            return 0;
        }
    }
}
```

```
//Proyección 3D a 2D y giros en los tres ejes simultáneamente. Líneas ocultas.
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {

        //Donde almacena los poligonos
        List<Poligono> poligonos;

        //Ángulos de giro
        double AnguloX, AnguloY, AnguloZ;
        double ZPersona;

        //Datos para la pantalla
        int pXmin, pYmin, pXmax, pYmax;

        public Form1() {
            InitializeComponent();
            poligonos = new List<Poligono>();

            //Polígonos que forman el cubo
            poligonos.Add(new Poligono(-0.5, 0.5, -0.5, 0.5, 0.5, -0.5, 0.5, -0.5, -0.5, -0.5, -0.5, -
0.5));
            poligonos.Add(new Poligono(-0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, -0.5, 0.5, -0.5, -0.5, 0.5));

            poligonos.Add(new Poligono(-0.5, 0.5, -0.5, -0.5, 0.5, 0.5, -0.5, -0.5, 0.5, -0.5, -0.5, -
0.5));
            poligonos.Add(new Poligono(0.5, 0.5, -0.5, 0.5, 0.5, 0.5, 0.5, -0.5, 0.5, 0.5, -0.5, -0.5));

            poligonos.Add(new Poligono(-0.5, 0.5, -0.5, 0.5, 0.5, -0.5, 0.5, 0.5, 0.5, -0.5, 0.5, 0.5));
            poligonos.Add(new Poligono(-0.5, -0.5, -0.5, 0.5, -0.5, -0.5, 0.5, -0.5, 0.5, -0.5, -0.5,
0.5));

            //Valores por defecto a los ángulos
            AnguloX = (double)numGiroX.Value;
            AnguloY = (double)numGiroY.Value;
            AnguloZ = (double)numGiroZ.Value;
            ZPersona = 5;

            //Coordenadas de pantalla
            pXmin = 20;
            pYmin = 20;
            pXmax = 500;
            pYmax = 500;

            Logica();
        }

        private void numGiroX_ValueChanged(object sender, System.EventArgs e) {
            AnguloX = Convert.ToDouble(numGiroX.Value);
            Logica();
            Refresh();
        }

        private void numGiroY_ValueChanged(object sender, EventArgs e) {
            AnguloY = Convert.ToDouble(numGiroY.Value);
            Logica();
            Refresh();
        }

        private void numGiroZ_ValueChanged(object sender, EventArgs e) {
            AnguloZ = Convert.ToDouble(numGiroZ.Value);
            Logica();
            Refresh();
        }

        private void numDistancia_ValueChanged(object sender, EventArgs e) {
        }

        public void Logica() {
            //Gira y convierte los puntos espaciales en puntos proyectados al plano
            for (int cont = 0; cont < poligonos.Count; cont++) {
                poligonos[cont].GiroXYZ(AnguloX, AnguloY, AnguloZ);
                poligonos[cont].Convierte3Da2D(ZPersona);
```

```
            }

            //Constantes para cuadrar el cubo en pantalla
            //Se obtuvieron probando los ángulos de giro X,Y,Z de 0 a 360 y ZPersona
            double XplanoMax = 0.87931543769177811;
            double XplanoMin = -0.87931543769177811;
            double YplanoMax = 0.87931539875237918;
            double YplanoMin = -0.87931539875237918;

            //Cuadra los puntos en pantalla
            double ConstanteX = (pXmax - pXmin) / (XplanoMax - XplanoMin);
            double ConstanteY = (pYmax - pYmin) / (YplanoMax - YplanoMin);

            for (int cont = 0; cont < poligonos.Count; cont++)
                poligonos[cont].CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin,
pYmin);

            //Ordena del polígono más alejado al más cercano, de esa manera los polígonos de adelante
            //son visibles y los de atrás son borrados.
            poligonos.Sort();
        }


        //Pinta la proyección del cubo
        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics lienzo = e.Graphics;
            Pen lapiz = new Pen(Color.Black, 1);
            Brush relleno = new SolidBrush(Color.White);
            for (int cont = 0; cont < poligonos.Count; cont++)
                poligonos[cont].Dibuja(lienzo, lapiz, relleno);
        }
    }
}
```

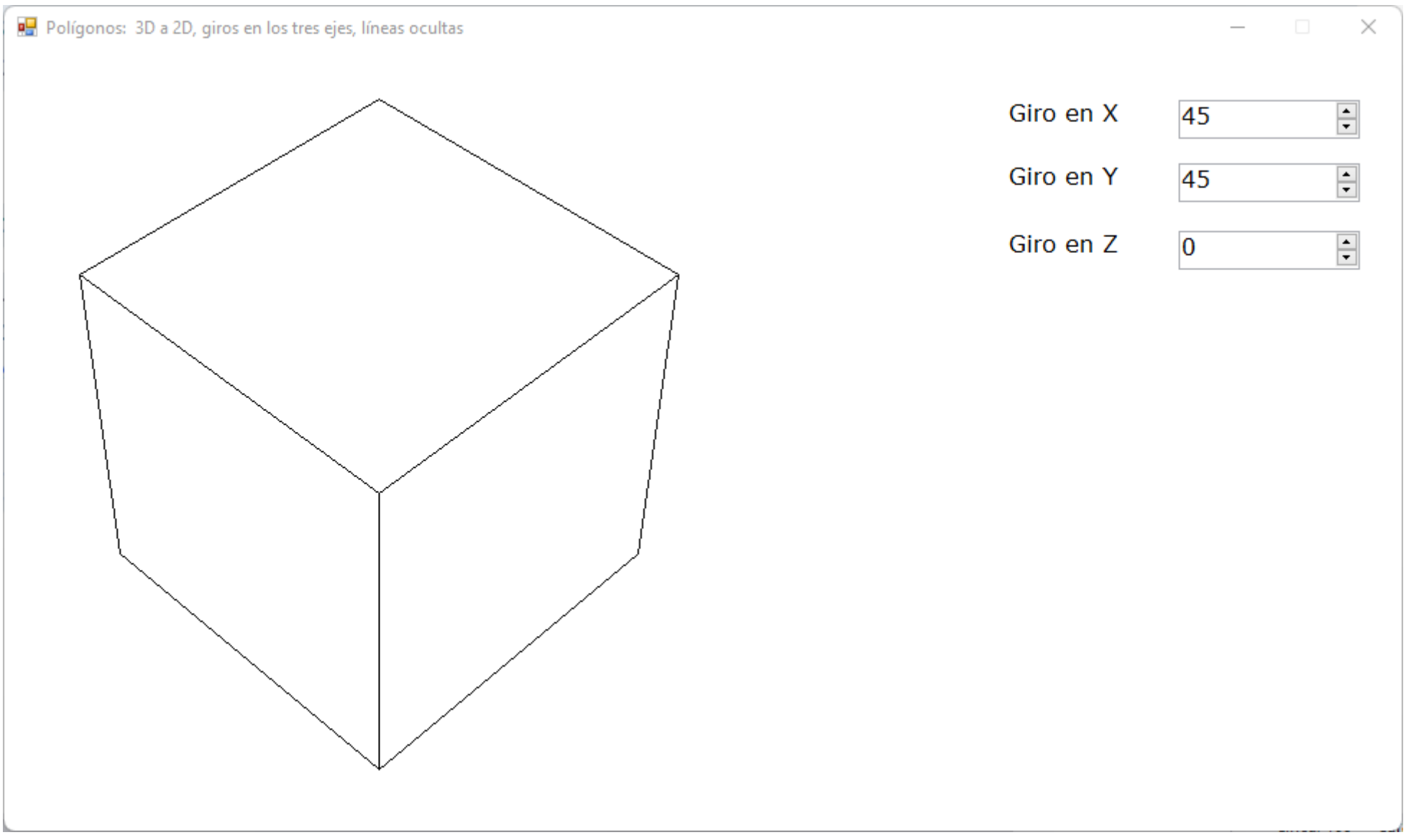


Ilustración 86: Posición inicial del cubo

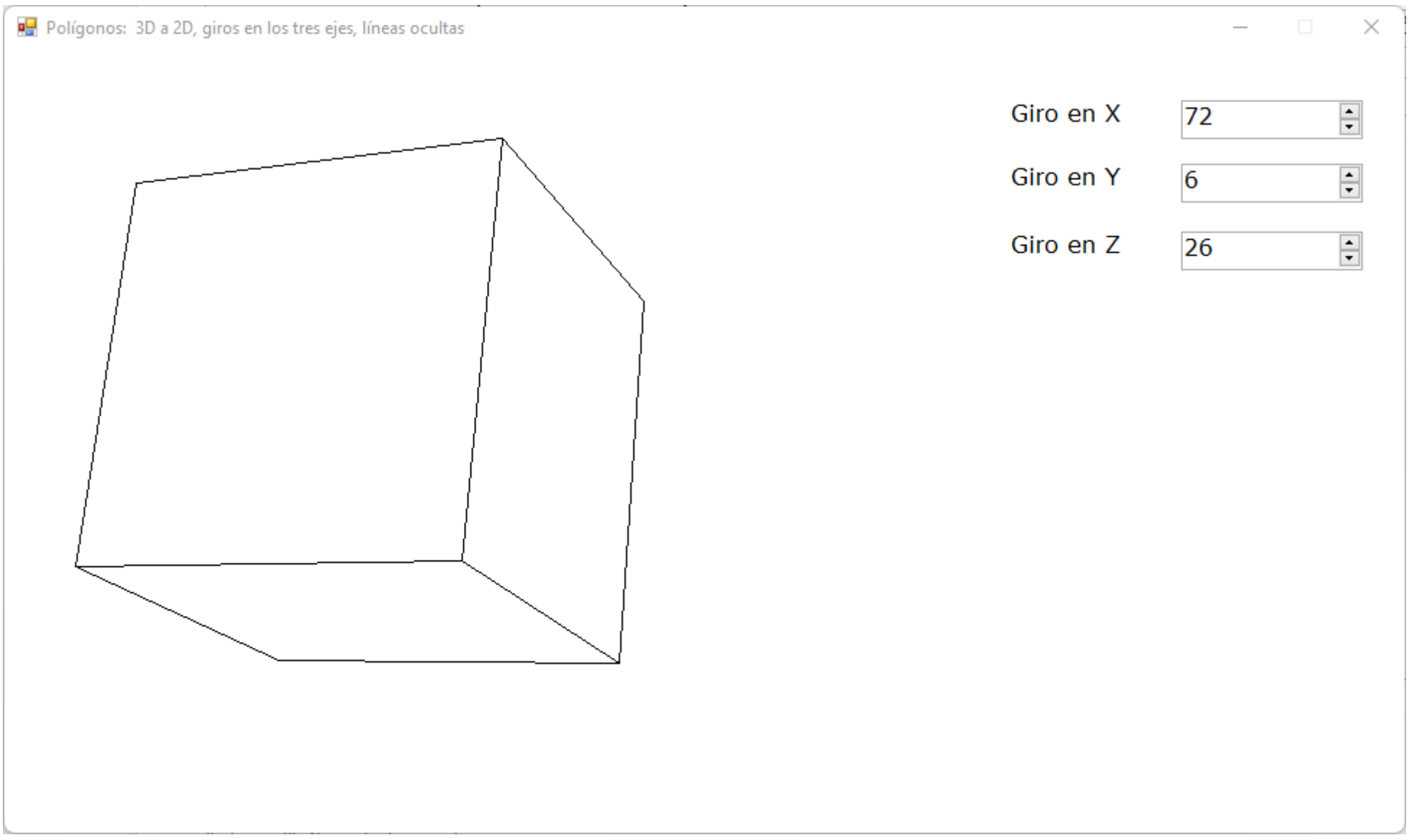


Ilustración 87: Giro en los tres ángulos

Gráfico de ecuación Z=F(X,Y)

Cuando tenemos una ecuación del tipo Z=F(X,Y), tenemos una ecuación con dos variables independientes y una variable dependiente. Su representación es en 3D. Por ejemplo:

$$Z = \sqrt[2]{X^2 + Y^2} + 3 * Cos(\sqrt[2]{X^2 + Y^2}) + 5$$

Los pasos para hacer el gráfico son los siguientes:

Paso 1: Saber el valor donde inicia X hasta donde termina. Igual sucede con Y, donde inicia Y hasta donde termina.

Paso 2: Con esos valores se calcula el valor Z. Al final se tiene un conjunto de coordenadas posX, posY, posZ. Se va a hacer uso de polígonos, por lo que se calculan cuatro coordenadas para formar el polígono.

Paso 3: Se normalizan los valores de posX, posY, posZ para que queden entre 0 y 1, luego se le resta -0.5 ¿Para qué? Para que los puntos (realmente polígonos) queden contenidos dentro de un cubo de lado=1, cuyo centro está en 0,0,0 . Ver los dos temas anteriores como se muestra el cubo.

Paso 4: Luego se aplica el giro en los tres ángulos. Se obtiene posXg, posYg, posZg

Paso 5: Se ordenan los polígonos del más profundo (menor valor de posZg) al más superficial (mayor valor de posZg). Por esa razón, la clase Polígono hereda de IComparable

Paso 6: Con Xg, Yg, Zg se proyecta a la pantalla, obteniéndose el planoX, planoY.

Paso 7: Con planoX, planoY se aplican las constantes que se calcularon anteriormente y se obtiene la proyección en pantalla, es decir, pantallaX, pantallaY

Paso 8: Se dibujan los polígonos, primero rellenándolos con el color del fondo y luego se gráfica el perímetro de ese polígono.

09. 3D/06. Grafico3D/Puntos.cs

```
using System;

namespace Graficos {
    internal class Puntos {
        //Coordenada espacial original
        public double posX, posY, posZ;

        //Coordenada espacial después de girar
        public double posXg, posYg, posZg;

        //Coordenada proyectada
        public double planoX, planoY;

        //Coordenada cuadrada en pantalla
        public int pantallaX, pantallaY;

        //Constructor
        public Puntos(double posX, double posY, double posZ) {
            this.posX = posX;
            this.posY = posY;
            this.posZ = posZ;
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
            double angXr = angX * Math.PI / 180;
            double angYr = angY * Math.PI / 180;
            double angZr = angZ * Math.PI / 180;

            double CosX = Math.Cos(angXr);
            double SinX = Math.Sin(angXr);
            double CosY = Math.Cos(angYr);
            double SinY = Math.Sin(angYr);
            double CosZ = Math.Cos(angZr);
            double SinZ = Math.Sin(angZr);

            //Matriz de Rotación
            //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
            double[,] Matriz = new double[3, 3] {
                { CosY * CosZ, -CosX * SinZ + SinX * SinY * CosZ, SinX * SinZ + CosX * SinY * CosZ},
                { CosY * SinZ, CosX * CosZ + SinX * SinY * SinZ, -SinX * CosZ + CosX * SinY * SinZ},
                {-SinY, SinX * CosY, CosX * CosY }
            };
```

```
            posXg = posX * Matriz[0, 0] + posY * Matriz[1, 0] + posZ * Matriz[2, 0];
            posYg = posX * Matriz[0, 1] + posY * Matriz[1, 1] + posZ * Matriz[2, 1];
            posZg = posX * Matriz[0, 2] + posY * Matriz[1, 2] + posZ * Matriz[2, 2];
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            planoX = posXg * ZPersona / (ZPersona - posZg);
            planoY = posYg * ZPersona / (ZPersona - posZg);
        }

        //Cuadra los puntos en pantalla
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            pantallaX = Convert.ToInt32(ConstanteX * (planoX - XplanoMin) + pXmin);
            pantallaY = Convert.ToInt32(ConstanteY * (planoY - YplanoMin) + pYmin);
        }
    }
}
```

09. 3D/06. Grafico3D/Poligono.cs

```
using System;
using System.Drawing;

namespace Graficos {
    internal class Poligono : IComparable {
        //Un polígono son cuatro(4) coordenadas espaciales
        public Puntos coordenadas1, coordenadas2, coordenadas3, coordenadas4;
        private double centro;

        public Poligono(double X1, double Y1, double Z1, double X2, double Y2, double Z2, double X3,
double Y3, double Z3, double X4, double Y4, double Z4) {
            coordenadas1 = new Puntos(X1, Y1, Z1);
            coordenadas2 = new Puntos(X2, Y2, Z2);
            coordenadas3 = new Puntos(X3, Y3, Z3);
            coordenadas4 = new Puntos(X4, Y4, Z4);
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
            coordenadas1.GiroXYZ(angX, angY, angZ);
            coordenadas2.GiroXYZ(angX, angY, angZ);
            coordenadas3.GiroXYZ(angX, angY, angZ);
            coordenadas4.GiroXYZ(angX, angY, angZ);
            centro = (coordenadas1.posZg + coordenadas2.posZg + coordenadas3.posZg + coordenadas4.posZg) /
4;
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            coordenadas1.Convierte3Da2D(ZPersona);
            coordenadas2.Convierte3Da2D(ZPersona);
            coordenadas3.Convierte3Da2D(ZPersona);
            coordenadas4.Convierte3Da2D(ZPersona);
        }

        //Cuadra en pantalla
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            coordenadas1.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas2.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas3.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas4.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
        }


        //Hace el gráfico del polígono
        public void Dibuja(Graphics lienzo, Pen lapiz, Brush relleno) {
            //Pone un color de fondo al polígono para borrar lo que hay detrás
            Point punto1 = new Point(coordenadas1.pantallaX, coordenadas1.pantallaY);
            Point punto2 = new Point(coordenadas2.pantallaX, coordenadas2.pantallaY);
            Point punto3 = new Point(coordenadas3.pantallaX, coordenadas3.pantallaY);
            Point punto4 = new Point(coordenadas4.pantallaX, coordenadas4.pantallaY);
            Point[] poligono = { punto1, punto2, punto3, punto4 };
            lienzo.FillPolygon(relleno, poligono);
```

```
            //Pinta el perímetro del polígono
            lienzo.DrawLine(lapiz, coordenadas1.pantallaX, coordenadas1.pantallaY, coordenadas2.pantallaX,
coordenadas2.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas2.pantallaX, coordenadas2.pantallaY, coordenadas3.pantallaX,
coordenadas3.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas3.pantallaX, coordenadas3.pantallaY, coordenadas4.pantallaX,
coordenadas4.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas4.pantallaX, coordenadas4.pantallaY, coordenadas1.pantallaX,
coordenadas1.pantallaY);
        }

        public int CompareTo(object obj) {
            Poligono orderToCompare = obj as Poligono;
            if (orderToCompare.centro < centro) {
                return 1;
            }
            if (orderToCompare.centro > centro) {
                return -1;
            }

            // The orders are equivalent.
            //https://stackoverflow.com/questions/3309188/how-to-sort-a-listt-by-a-property-in-the-object
            return 0;
        }
    }
}
```

09. 3D/06. Grafico3D/Form1.cs

```
//Graficar ecuaciones de doble variable independiente
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {

        //Donde almacena los poligonos
        List<Poligono> poligonos;

        //Ángulos de giro
        double AnguloX, AnguloY, AnguloZ;
        int ZPersona;

        //Datos para la ecuación
        double minX, minY, maxX, maxY;
        int numLineas;

        //Datos para la pantalla
        int pXmin, pYmin, pXmax, pYmax;

        public Form1() {
            InitializeComponent();
            poligonos = new List<Poligono>();

            //Valores por defecto a los ángulos
            AnguloX = (double)numGiroX.Value;
            AnguloY = (double)numGiroY.Value;
            AnguloZ = (double)numGiroZ.Value;
            ZPersona = 5;

            //Valores para la ecuación
            minX = -9;
            minY = -9;
            maxX = 9;
            maxY = 9;
            numLineas = 30;

            //Coordenadas de pantalla
            pXmin = 0;
            pYmin = 0;
            pXmax = 900;
            pYmax = 630;

            Logica();
        }
```

```
private void numGiroX_ValueChanged(object sender, System.EventArgs e) {
    AnguloX = Convert.ToDouble(numGiroX.Value);
    Logica();
    Refresh();
}

private void numGiroY_ValueChanged(object sender, EventArgs e) {
    AnguloY = Convert.ToDouble(numGiroY.Value);
    Logica();
    Refresh();
}

private void numGiroZ_ValueChanged(object sender, EventArgs e) {
    AnguloZ = Convert.ToDouble(numGiroZ.Value);
    Logica();
    Refresh();
}

public void Logica() {
    //Calcula cada coordenada del gráfico de la ecuación.
    poligonos.Clear();
    double incrementoX = (maxX - minX) / numLineas;
    double incrementoY = (maxY - minY) / numLineas;

    //Mínimos y máximos para normalizar
    double minimoX = double.MaxValue;
    double minimoY = double.MaxValue;
    double minimoZ = double.MaxValue;
    double maximoX = double.MinValue;
    double maximoY = double.MinValue;
    double maximoZ = double.MinValue;

    for (double valX = minX; valX <= maxX; valX += incrementoX)
        for (double valY = minY; valY <= maxY; valY += incrementoY) {
            double X1 = valX;
            double Y1 = valY;
            double Z1 = Ecuacion(X1, Y1);
            if (double.IsNaN(Z1) || double.IsInfinity(Z1)) Z1 = 0;

            double X2 = valX + incrementoX;
            double Y2 = valY;
            double Z2 = Ecuacion(X2, Y2);
            if (double.IsNaN(Z2) || double.IsInfinity(Z2)) Z2 = 0;

            double X3 = valX + incrementoX;
            double Y3 = valY + incrementoY;
            double Z3 = Ecuacion(X3, Y3);
            if (double.IsNaN(Z3) || double.IsInfinity(Z3)) Z3 = 0;

            double X4 = valX;
            double Y4 = valY + incrementoY;
            double Z4 = Ecuacion(X4, Y4);
            if (double.IsNaN(Z4) || double.IsInfinity(Z4)) Z4 = 0;

            if (X1 < minimoX) minimoX = X1;
            if (X2 < minimoX) minimoX = X2;
            if (X3 < minimoX) minimoX = X3;
            if (X4 < minimoX) minimoX = X4;

            if (Y1 < minimoY) minimoY = Y1;
            if (Y2 < minimoY) minimoY = Y2;
            if (Y3 < minimoY) minimoY = Y3;
            if (Y4 < minimoY) minimoY = Y4;

            if (Z1 < minimoZ) minimoZ = Z1;
            if (Z2 < minimoZ) minimoZ = Z2;
            if (Z3 < minimoZ) minimoZ = Z3;
            if (Z4 < minimoZ) minimoZ = Z4;

            if (X1 > maximoX) maximoX = X1;
            if (X2 > maximoX) maximoX = X2;
            if (X3 > maximoX) maximoX = X3;
            if (X4 > maximoX) maximoX = X4;

            if (Y1 > maximoY) maximoY = Y1;
            if (Y2 > maximoY) maximoY = Y2;
            if (Y3 > maximoY) maximoY = Y3;
            if (Y4 > maximoY) maximoY = Y4;
```

```
                    if (Z1 > maximoZ) maximoZ = Z1;
                    if (Z2 > maximoZ) maximoZ = Z2;
                    if (Z3 > maximoZ) maximoZ = Z3;
                    if (Z4 > maximoZ) maximoZ = Z4;

                    poligonos.Add(new Poligono(X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3, X4, Y4, Z4));
                }

            //Luego normaliza los puntos X,Y,Z para que queden entre -0.5 y 0.5
            for (int cont = 0; cont < poligonos.Count; cont++) {
                poligonos[cont].coordenadas1.posX = (poligonos[cont].coordenadas1.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas1.posY = (poligonos[cont].coordenadas1.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas1.posZ = (poligonos[cont].coordenadas1.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas2.posX = (poligonos[cont].coordenadas2.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas2.posY = (poligonos[cont].coordenadas2.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas2.posZ = (poligonos[cont].coordenadas2.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas3.posX = (poligonos[cont].coordenadas3.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas3.posY = (poligonos[cont].coordenadas3.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas3.posZ = (poligonos[cont].coordenadas3.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas4.posX = (poligonos[cont].coordenadas4.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas4.posY = (poligonos[cont].coordenadas4.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas4.posZ = (poligonos[cont].coordenadas4.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;
            }

            //Gira y convierte los puntos espaciales en puntos proyectados al plano
            for (int cont = 0; cont < poligonos.Count; cont++) {
                poligonos[cont].GiroXYZ(AnguloX, AnguloY, AnguloZ);
                poligonos[cont].Convierte3Da2D(ZPersona);
            }

            //Constantes para cuadrar el gráfico en pantalla
            //Se obtuvieron probando los ángulos de giro X,Y,Z de 0 a 360 y ZPersona
            double XplanoMax = 0.87931543769177811;
            double XplanoMin = -0.87931543769177811;
            double YplanoMax = 0.87931539875237918;
            double YplanoMin = -0.87931539875237918;

        //Constantes de transformación
            double ConstanteX = (pXmax - pXmin) / (XplanoMax - XplanoMin);
            double ConstanteY = (pYmax - pYmin) / (YplanoMax - YplanoMin);

            //Cuadra los polígonos en pantalla
            for (int cont = 0; cont < poligonos.Count; cont++)
                poligonos[cont].CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin,
pYmin);

            //Ordena del polígono más alejado al más cercano, de esa manera los polígonos de adelante
            //son visibles y los de atrás son borrados.
            poligonos.Sort();
        }

        //Aquí está la ecuación 3D que se desee graficar con variable XY
        public double Ecuacion(double x, double y) {
            return Math.Sqrt(x * x + y * y) + 3 * Math.Cos(Math.Sqrt(x * x + y * y)) + 5;
        }


        //Pinta el gráfico generado por la ecuación
        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics lienzo = e.Graphics;
            Pen lapiz = new Pen(Color.Black, 1);
            Brush relleno = new SolidBrush(Color.White);
```

```
            for (int cont = 0; cont < poligonos.Count; cont++) poligonos[cont].Dibuja(lienzo, lapiz,
relleno);
        }
    }
}
```

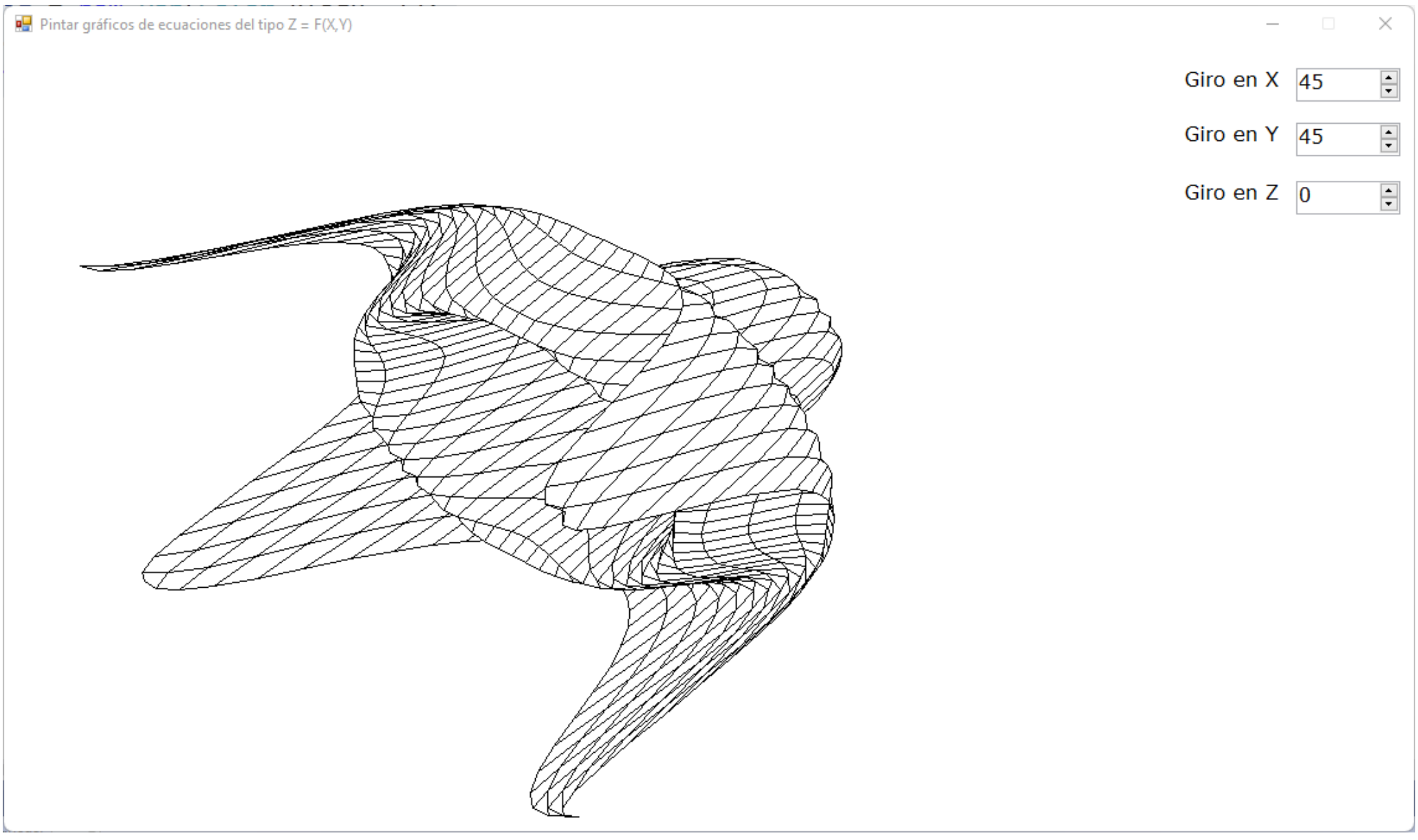


Ilustración 88: Gráfico de ecuación Z=F(X,Y)

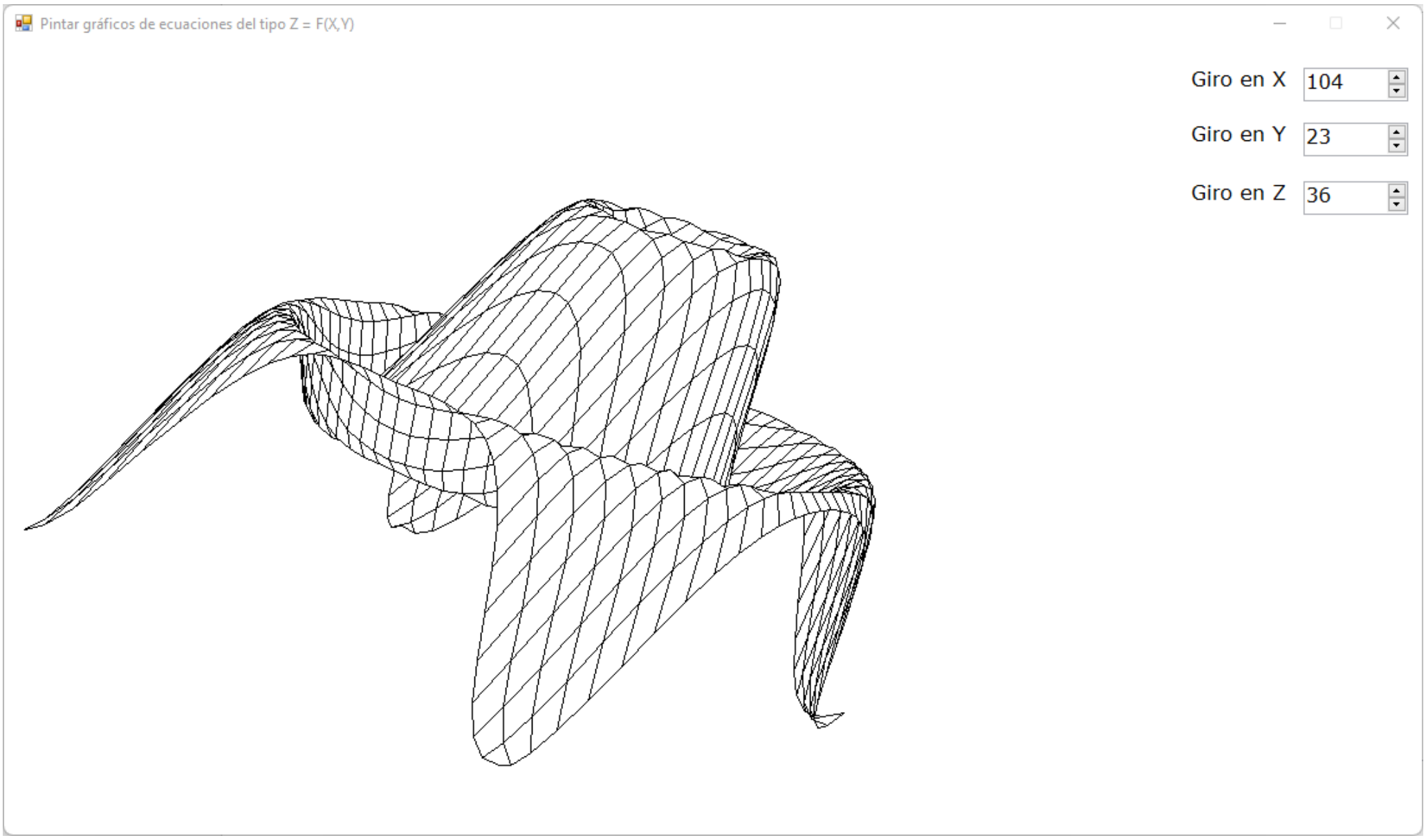


Ilustración 89: Gráfico de ecuación Z=F(X,Y)

Gráfico de ecuación Z=F(X,Y,T). Animado

Cuando tenemos una ecuación del tipo Z=F(X,Y,T), tenemos una ecuación con tres variables independientes y una variable dependiente. Su representación es en 3D animada. Por ejemplo:

$$Z = \sqrt[2]{T * X^2 + T * Y^2} + 3 * Cos(\sqrt[2]{T * X^2 + T * Y^2}) + 5$$

Los pasos para hacer el gráfico son los siguientes:

Paso 0: Saber el valor del tiempo mínimo y el valor del tiempo máximo. Se hace uso de un control "timer" que cada vez que se dispara el evento "Tick" varía el valor de T desde el mínimo hasta el máximo paso a paso, una vez alcanzado el máximo se decrementa paso a paso hasta el mínimo y repite el ciclo. De resto es igual a hacer un gráfico de ecuación tipo Z=F(X,Y).

Paso 1: Saber el valor donde inicia X hasta donde termina. Igual sucede con Y, donde inicia Y hasta donde termina.

Paso 2: Con esos valores se calcula el valor Z. Al final se tiene un conjunto de coordenadas posX, posY, posZ. Se va a hacer uso de polígonos, por lo que se calculan cuatro coordenadas para formar el polígono.

Paso 3: Se normalizan los valores de posX, posY, posZ para que queden entre 0 y 1, luego se le resta -0.5 ¿Para qué? Para que los puntos (realmente polígonos) queden contenidos dentro de un cubo de lado=1, cuyo centro está en 0,0,0 . Ver los dos temas anteriores como se muestra el cubo.

Paso 4: Luego se aplica el giro en los tres ángulos. Se obtiene posXg, posYg, posZg

Paso 5: Se ordenan los polígonos del más profundo (menor valor de posZg) al más superficial (mayor valor de posZg). Por esa razón, la clase Polígono hereda de IComparable

Paso 6: Con Xg, Yg, Zg se proyecta a la pantalla, obteniéndose el planoX, planoY.

Paso 7: Con planoX, planoY se aplican las constantes que se calcularon anteriormente y se obtiene la proyección en pantalla, es decir, pantallaX, pantallaY

Paso 8: Se dibujan los polígonos, primero rellenándolos con el color del fondo y luego se gráfica el perímetro de ese polígono.

Nota 1: El gráfico se ve animado por lo que es necesario que el formulario tenga en "true" la propiedad DoubleBuffered

Nota 2: El gráfico animado está contenido dentro de un cubo invisible de lado = 1. Eso se logra con la normalización. Sin importar el valor que tome Z, el gráfico estará contenido dentro de ese cubo, luego si Z crece mucho, el resultado visual es que el gráfico se empequeñece para que se ajuste todo a ese cubo.

09. 3D/07. Grafico3DAnimado/Puntos.cs

```
using System;

namespace Graficos {
    internal class Puntos {
        //Coordenada espacial original
        public double posX, posY, posZ;

        //Coordenada espacial después de girar
        public double posXg, posYg, posZg;

        //Coordenada proyectada
        public double planoX, planoY;

        //Coordenada cuadrada en pantalla
        public int pantallaX, pantallaY;

        //Constructor
        public Puntos(double posX, double posY, double posZ) {
            this.posX = posX;
            this.posY = posY;
            this.posZ = posZ;
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
            double angXr = angX * Math.PI / 180;
            double angYr = angY * Math.PI / 180;
            double angZr = angZ * Math.PI / 180;

            double CosX = Math.Cos(angXr);
            double SinX = Math.Sin(angXr);
            double CosY = Math.Cos(angYr);
            double SinY = Math.Sin(angYr);
            double CosZ = Math.Cos(angZr);
```

```
            double SinZ = Math.Sin(angZr);

            //Matriz de Rotación
            //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
            double[,] Matriz = new double[3, 3] {
                { CosY * CosZ, -CosX * SinZ + SinX * SinY * CosZ, SinX * SinZ + CosX * SinY * CosZ},
                { CosY * SinZ, CosX * CosZ + SinX * SinY * SinZ, -SinX * CosZ + CosX * SinY * SinZ},
                {-SinY, SinX * CosY, CosX * CosY }
            };

            posXg = posX * Matriz[0, 0] + posY * Matriz[1, 0] + posZ * Matriz[2, 0];
            posYg = posX * Matriz[0, 1] + posY * Matriz[1, 1] + posZ * Matriz[2, 1];
            posZg = posX * Matriz[0, 2] + posY * Matriz[1, 2] + posZ * Matriz[2, 2];
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            planoX = posXg * ZPersona / (ZPersona - posZg);
            planoY = posYg * ZPersona / (ZPersona - posZg);
        }

        //Cuadra los puntos en pantalla
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            pantallaX = Convert.ToInt32(ConstanteX * (planoX - XplanoMin) + pXmin);
            pantallaY = Convert.ToInt32(ConstanteY * (planoY - YplanoMin) + pYmin);
        }
    }
}
```



```
using System;
using System.Drawing;

namespace Graficos {
    internal class Poligono : IComparable {
        //Un polígono son cuatro(4) coordenadas espaciales
        public Puntos coordenadas1, coordenadas2, coordenadas3, coordenadas4;
        private double centro;

        public Poligono(double X1, double Y1, double Z1, double X2, double Y2, double Z2, double X3,
double Y3, double Z3, double X4, double Y4, double Z4) {
            coordenadas1 = new Puntos(X1, Y1, Z1);
            coordenadas2 = new Puntos(X2, Y2, Z2);
            coordenadas3 = new Puntos(X3, Y3, Z3);
            coordenadas4 = new Puntos(X4, Y4, Z4);
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
            centro = double.MinValue;
            coordenadas1.GiroXYZ(angX, angY, angZ);
            coordenadas2.GiroXYZ(angX, angY, angZ);
            coordenadas3.GiroXYZ(angX, angY, angZ);
            coordenadas4.GiroXYZ(angX, angY, angZ);
            if (coordenadas1.posZg > centro) centro = coordenadas1.posZg;
            if (coordenadas2.posZg > centro) centro = coordenadas2.posZg;
            if (coordenadas3.posZg > centro) centro = coordenadas3.posZg;
            if (coordenadas4.posZg > centro) centro = coordenadas4.posZg;
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            coordenadas1.Convierte3Da2D(ZPersona);
            coordenadas2.Convierte3Da2D(ZPersona);
            coordenadas3.Convierte3Da2D(ZPersona);
            coordenadas4.Convierte3Da2D(ZPersona);
        }

        //Cuadra en pantalla
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            coordenadas1.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas2.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas3.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas4.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
        }
```

```
        //Hace el gráfico del polígono
        public void Dibuja(Graphics lienzo, Pen lapiz, Brush relleno) {
            //Pone un color de fondo al polígono para borrar lo que hay detrás
            Point punto1 = new Point(coordenadas1.pantallaX, coordenadas1.pantallaY);
            Point punto2 = new Point(coordenadas2.pantallaX, coordenadas2.pantallaY);
            Point punto3 = new Point(coordenadas3.pantallaX, coordenadas3.pantallaY);
            Point punto4 = new Point(coordenadas4.pantallaX, coordenadas4.pantallaY);
            Point[] poligono = { punto1, punto2, punto3, punto4 };
            lienzo.FillPolygon(relleno, poligono);

            //Pinta el perímetro del polígono
            lienzo.DrawLine(lapiz, coordenadas1.pantallaX, coordenadas1.pantallaY, coordenadas2.pantallaX,
coordenadas2.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas2.pantallaX, coordenadas2.pantallaY, coordenadas3.pantallaX,
coordenadas3.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas3.pantallaX, coordenadas3.pantallaY, coordenadas4.pantallaX,
coordenadas4.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas4.pantallaX, coordenadas4.pantallaY, coordenadas1.pantallaX,
coordenadas1.pantallaY);
        }

        public int CompareTo(object obj) {
            Poligono orderToCompare = obj as Poligono;
            if (orderToCompare.centro < centro) {
                return 1;
            }
            if (orderToCompare.centro > centro) {
                return -1;
            }

            // The orders are equivalent.
            //https://stackoverflow.com/questions/3309188/how-to-sort-a-listt-by-a-property-in-the-object
            return 0;
        }
    }
}
```

09. 3D/07. Grafico3DAnimado/Form1.cs

```
//Graficar ecuaciones de doble variable independiente y una variable temporal
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Para la variable temporal
        private double Tminimo, Tmaximo, Tincrementa, Tiempo;

        //Donde almacena los poligonos
        List<Poligono> poligonos;

        //Ángulos de giro
        double AnguloX, AnguloY, AnguloZ;
        int ZPersona;

        //Datos para la ecuación
        double minX, minY, maxX, maxY;
        int numLineas;

        //Datos para la pantalla
        int pXmin, pYmin, pXmax, pYmax;

        public Form1() {
            InitializeComponent();
            poligonos = new List<Poligono>();

            //Valores por defecto a los ángulos
            AnguloX = (double)numGiroX.Value;
            AnguloY = (double)numGiroY.Value;
            AnguloZ = (double)numGiroZ.Value;
            ZPersona = 5;

            //Valores para la ecuación
            minX = -9;
```

```
        minY = -9;
        maxX = 9;
        maxY = 9;
        numLineas = 30;

        //Inicia el tiempo
        Tminimo = 0.1;
        Tmaximo = 1;
        Tincrementa = 0.01;
        Tiempo = Tminimo;

        //Valores de la pantalla (región donde se dibuja)
        pXmin = 0;
        pYmin = 0;
        pXmax = 900;
        pYmax = 630;

        Logica();
    }

    private void timerAnimar_Tick(object sender, EventArgs e) {
        Tiempo += Tincrementa;
        if (Tiempo <= Tminimo || Tiempo >= Tmaximo) Tincrementa = -Tincrementa;
        Logica();
        Refresh();
    }

    private void numGiroX_ValueChanged(object sender, System.EventArgs e) {
        AnguloX = Convert.ToDouble(numGiroX.Value);
        Logica();
        Refresh();
    }

    private void numGiroY_ValueChanged(object sender, EventArgs e) {
        AnguloY = Convert.ToDouble(numGiroY.Value);
        Logica();
        Refresh();
    }

    private void numGiroZ_ValueChanged(object sender, EventArgs e) {
        AnguloZ = Convert.ToDouble(numGiroZ.Value);
        Logica();
        Refresh();
    }

    public void Logica() {
        //Calcula cada coordenada del gráfico de la ecuación.
        poligonos.Clear();
        double incrementoX = (maxX - minX) / numLineas;
        double incrementoY = (maxY - minY) / numLineas;

        //Mínimos y máximos para normalizar
        double minimoX = double.MaxValue;
        double minimoY = double.MaxValue;
        double minimoZ = double.MaxValue;
        double maximoX = double.MinValue;
        double maximoY = double.MinValue;
        double maximoZ = double.MinValue;

        //Hace el cálculo de las 4 coordenadas de cada polígono
        for (double valX = minX; valX <= maxX; valX += incrementoX)
            for (double valY = minY; valY <= maxY; valY += incrementoY) {

                //Primera coordenada
                double X1 = valX;
                double Y1 = valY;
                double Z1 = Ecuacion(X1, Y1, Tiempo);
                if (double.IsNaN(Z1) || double.IsInfinity(Z1)) Z1 = 0;

                //Segunda coordenada
                double X2 = valX + incrementoX;
                double Y2 = valY;
                double Z2 = Ecuacion(X2, Y2, Tiempo);
                if (double.IsNaN(Z2) || double.IsInfinity(Z2)) Z2 = 0;

                //Tercera coordenada
                double X3 = valX + incrementoX;
                double Y3 = valY + incrementoY;
                double Z3 = Ecuacion(X3, Y3, Tiempo);
```

```
                if (double.IsNaN(Z3) || double.IsInfinity(Z3)) Z3 = 0;

                //Cuarta coordenada
                double X4 = valX;
                double Y4 = valY + incrementoY;
                double Z4 = Ecuacion(X4, Y4, Tiempo);
                if (double.IsNaN(Z4) || double.IsInfinity(Z4)) Z4 = 0;

                //Para más adelante hace la normalización
                if (X1 < minimoX) minimoX = X1;
                if (X2 < minimoX) minimoX = X2;
                if (X3 < minimoX) minimoX = X3;
                if (X4 < minimoX) minimoX = X4;

                if (Y1 < minimoY) minimoY = Y1;
                if (Y2 < minimoY) minimoY = Y2;
                if (Y3 < minimoY) minimoY = Y3;
                if (Y4 < minimoY) minimoY = Y4;

                if (Z1 < minimoZ) minimoZ = Z1;
                if (Z2 < minimoZ) minimoZ = Z2;
                if (Z3 < minimoZ) minimoZ = Z3;
                if (Z4 < minimoZ) minimoZ = Z4;

                if (X1 > maximoX) maximoX = X1;
                if (X2 > maximoX) maximoX = X2;
                if (X3 > maximoX) maximoX = X3;
                if (X4 > maximoX) maximoX = X4;

                if (Y1 > maximoY) maximoY = Y1;
                if (Y2 > maximoY) maximoY = Y2;
                if (Y3 > maximoY) maximoY = Y3;
                if (Y4 > maximoY) maximoY = Y4;

                if (Z1 > maximoZ) maximoZ = Z1;
                if (Z2 > maximoZ) maximoZ = Z2;
                if (Z3 > maximoZ) maximoZ = Z3;
                if (Z4 > maximoZ) maximoZ = Z4;

                //Crea el polígono
                poligonos.Add(new Poligono(X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3, X4, Y4, Z4));
            }

            //Luego normaliza los puntos X,Y,Z de cada polígono para que queden entre -0.5 y 0.5
            //Es decir, el gráfico queda confinado en un cubo de lado = 1
            for (int cont = 0; cont < poligonos.Count; cont++) {
                poligonos[cont].coordenadas1.posX = (poligonos[cont].coordenadas1.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas1.posY = (poligonos[cont].coordenadas1.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas1.posZ = (poligonos[cont].coordenadas1.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas2.posX = (poligonos[cont].coordenadas2.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas2.posY = (poligonos[cont].coordenadas2.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas2.posZ = (poligonos[cont].coordenadas2.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas3.posX = (poligonos[cont].coordenadas3.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas3.posY = (poligonos[cont].coordenadas3.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas3.posZ = (poligonos[cont].coordenadas3.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas4.posX = (poligonos[cont].coordenadas4.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas4.posY = (poligonos[cont].coordenadas4.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas4.posZ = (poligonos[cont].coordenadas4.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;
            }

            //Gira y convierte los puntos espaciales en puntos proyectados al plano
            for (int cont = 0; cont < poligonos.Count; cont++) {
                poligonos[cont].GiroXYZ(AnguloX, AnguloY, AnguloZ);
                poligonos[cont].Convierte3Da2D(ZPersona);
```

```
            }

            //Constantes para cuadrar el gráfico en pantalla
            //Se obtuvieron probando los ángulos de giro X,Y,Z de 0 a 360 y ZPersona
            double XplanoMax = 0.87931543769177811;
            double XplanoMin = -0.87931543769177811;
            double YplanoMax = 0.87931539875237918;
            double YplanoMin = -0.87931539875237918;

            double ConstanteX = (pXmax - pXmin) / (XplanoMax - XplanoMin);
            double ConstanteY = (pYmax - pYmin) / (YplanoMax - YplanoMin);

            //Cuadra los polígonos para aparecer en pantalla
            for (int cont = 0; cont < poligonos.Count; cont++)
                poligonos[cont].CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin,
pYmin);

            //Ordena del polígono más alejado al más cercano, de esa manera los polígonos de adelante
            //son visibles y los de atrás son borrados.
            poligonos.Sort();
        }

        //Ecuación con las tres variables independientes, una de ellas es el tiempo t
        public double Ecuacion(double x, double y, double t) {
            return Math.Sqrt(t * x * x + t * y * y) + 3 * Math.Cos(Math.Sqrt(t * x * x + t * y * y)) + 5;
        }

        //Pinta el gráfico generado por la ecuación
        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics lienzo = e.Graphics;
            Pen lapiz = new Pen(Color.Black, 1);
            Brush relleno = new SolidBrush(Color.White);
            for (int cont = 0; cont < poligonos.Count; cont++) poligonos[cont].Dibuja(lienzo, lapiz,
relleno);
        }
    }
}
```

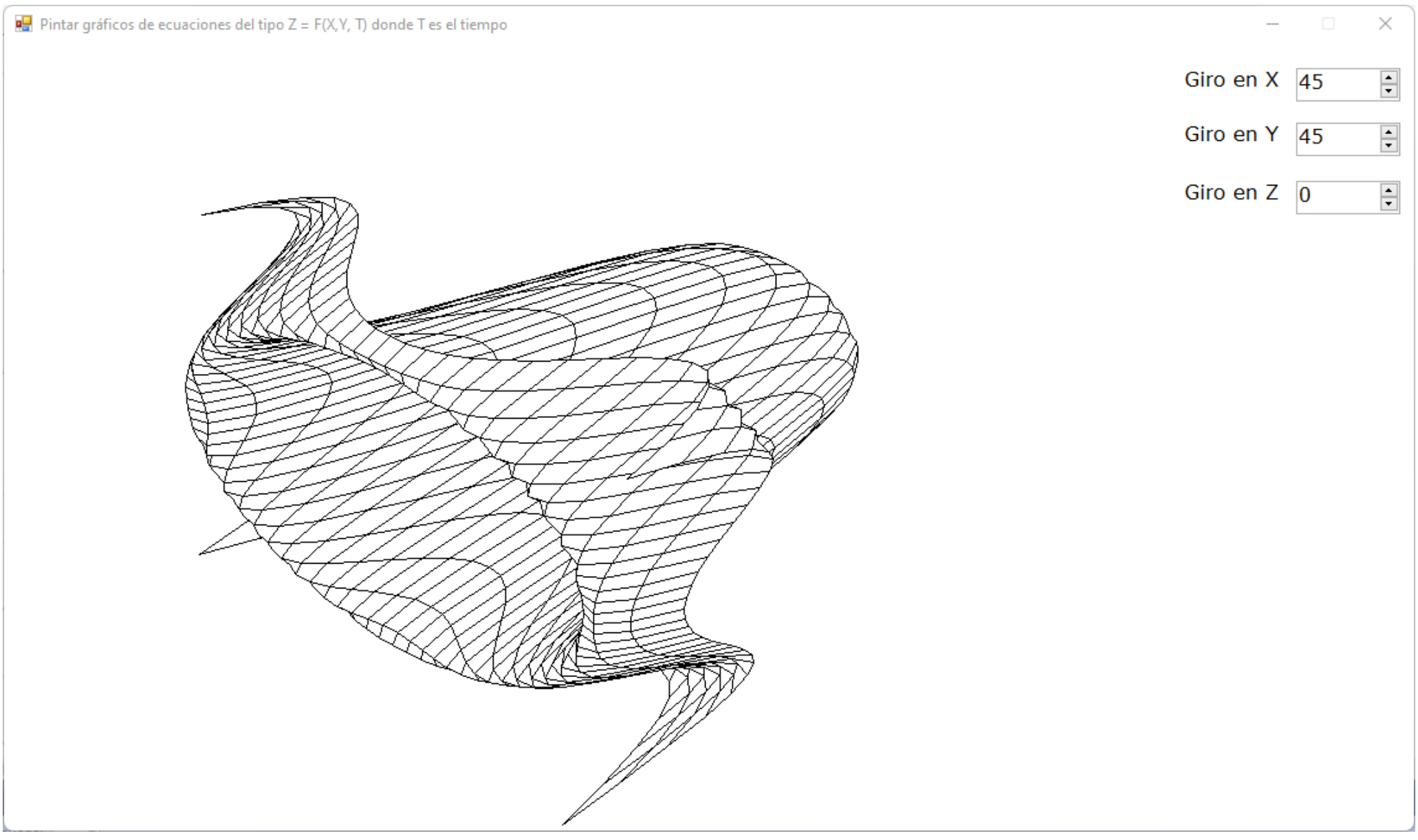


Ilustración 90: Gráfico de ecuación Z=F(X,Y) en determinado T

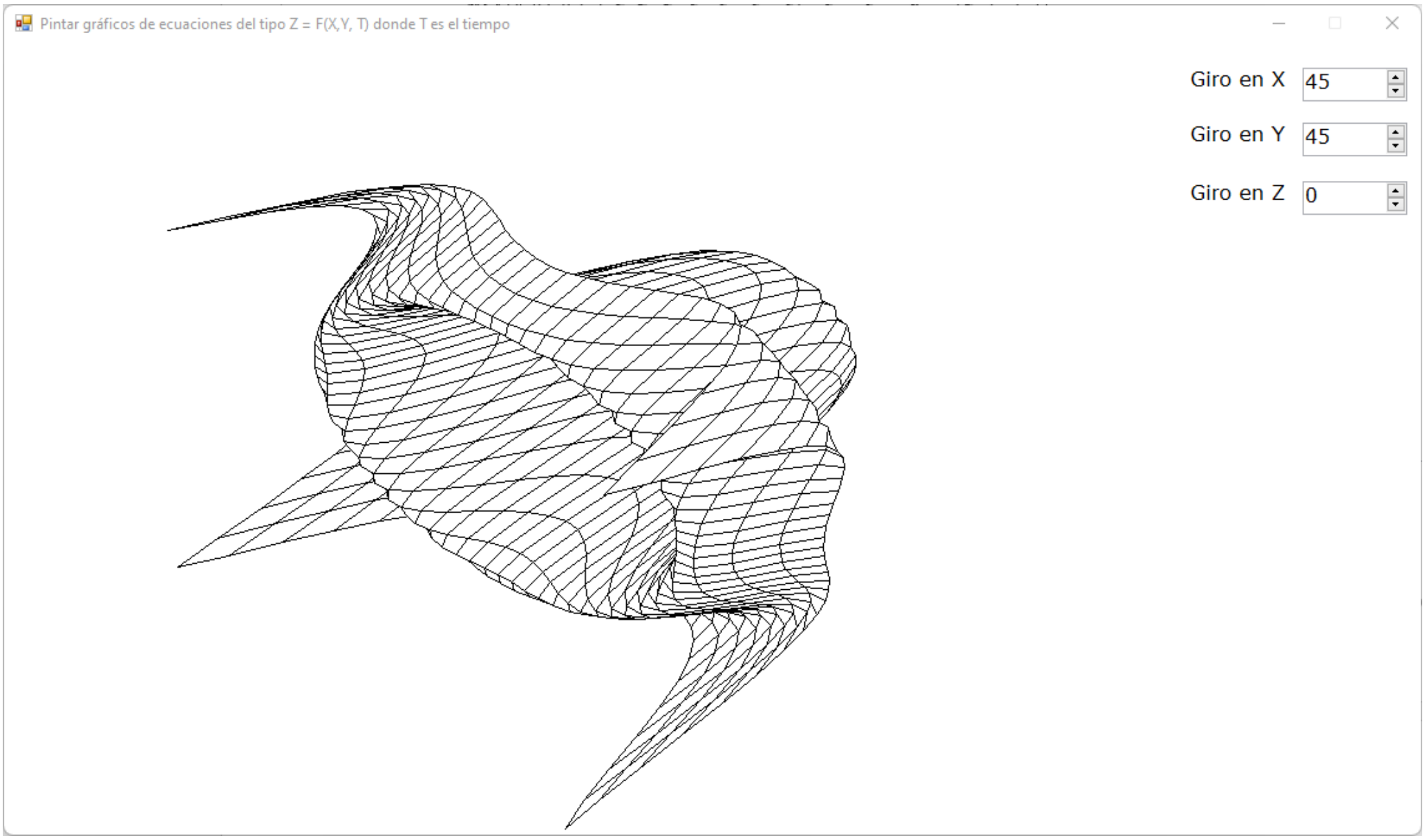


Ilustración 91: Gráfico de ecuación Z=F(X,Y) en determinado T

Gráfico Polar en 3D

Los gráficos polares en 3D trabajan con dos ángulos: Theta y Phi. Ver: https://mathematica.stackexchange.com/questions/83867/how-to-make-a-3d-plot-using-polar-coordinates o https://stackoverflow.com/questions/55031161/polar3d-plot-with-theta-phi-and-radius o https://mathworld.wolfram.com/SphericalCoordinates.html o https://en.wikipedia.org/wiki/Spherical_coordinate_system

Por ejemplo, esta ecuación:

$$r = Cos(\varphi) + Sen(\theta)$$

Los pasos para dibujar el gráfico polar 3D son:

Paso 1: φ va de 0 a 2π, Θ va de 0 a π y con eso se obtienen los valores de r

Paso 2: Luego se hace la traducción a coordenadas XYZ con estas fórmulas:

$$x = r * Cos(\varphi) * Sen(\theta)$$

$$y = r * Sen(\varphi) * Sen(\theta)$$

$$z = r * Cos(\theta)$$

Paso 3: Se hace el mismo procedimiento con los gráficos 3D de XYZ

09. 3D/08. Polar3D/Puntos.cs

```
using System;

namespace Graficos {
    internal class Puntos {
        //Coordenada espacial original
        public double posX, posY, posZ;

        //Coordenada espacial después de girar
        public double posXg, posYg, posZg;

        //Coordenada proyectada
        public double planoX, planoY;

        //Coordenada cuadrada en pantalla
        public int pantallaX, pantallaY;

        //Constructor
        public Puntos(double posX, double posY, double posZ) {
            this.posX = posX;
            this.posY = posY;
            this.posZ = posZ;
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
            double angXr = angX * Math.PI / 180;
            double angYr = angY * Math.PI / 180;
            double angZr = angZ * Math.PI / 180;

            double CosX = Math.Cos(angXr);
            double SinX = Math.Sin(angXr);
            double CosY = Math.Cos(angYr);
            double SinY = Math.Sin(angYr);
            double CosZ = Math.Cos(angZr);
            double SinZ = Math.Sin(angZr);

            //Matriz de Rotación
            //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
            double[,] Matriz = new double[3, 3] {
                { CosY * CosZ, -CosX * SinZ + SinX * SinY * CosZ, SinX * SinZ + CosX * SinY * CosZ},
                { CosY * SinZ, CosX * CosZ + SinX * SinY * SinZ, -SinX * CosZ + CosX * SinY * SinZ},
                {-SinY, SinX * CosY, CosX * CosY }
            };

            posXg = posX * Matriz[0, 0] + posY * Matriz[1, 0] + posZ * Matriz[2, 0];
            posYg = posX * Matriz[0, 1] + posY * Matriz[1, 1] + posZ * Matriz[2, 1];
            posZg = posX * Matriz[0, 2] + posY * Matriz[1, 2] + posZ * Matriz[2, 2];
    }
```

```
        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            planoX = posXg * ZPersona / (ZPersona - posZg);
            planoY = posYg * ZPersona / (ZPersona - posZg);
        }

        //Cuadra los puntos en pantalla
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            pantallaX = Convert.ToInt32(ConstanteX * (planoX - XplanoMin) + pXmin);
            pantallaY = Convert.ToInt32(ConstanteY * (planoY - YplanoMin) + pYmin);
        }
    }
}
```

09. 3D/08. Polar3D/Poligono.cs

```
using System;
using System.Drawing;

namespace Graficos {
    internal class Poligono : IComparable {
        //Un polígono son cuatro(4) coordenadas espaciales
        public Puntos coordenadas1, coordenadas2, coordenadas3, coordenadas4;
        private double Zprofundidad;

        public Poligono(double X1, double Y1, double Z1, double X2, double Y2, double Z2, double X3,
double Y3, double Z3, double X4, double Y4, double Z4) {
            coordenadas1 = new Puntos(X1, Y1, Z1);
            coordenadas2 = new Puntos(X2, Y2, Z2);
            coordenadas3 = new Puntos(X3, Y3, Z3);
            coordenadas4 = new Puntos(X4, Y4, Z4);
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
            coordenadas1.GiroXYZ(angX, angY, angZ);
            coordenadas2.GiroXYZ(angX, angY, angZ);
            coordenadas3.GiroXYZ(angX, angY, angZ);
            coordenadas4.GiroXYZ(angX, angY, angZ);
            Zprofundidad = double.MaxValue;
            if (coordenadas1.posZg < Zprofundidad) Zprofundidad = coordenadas1.posZg;
            if (coordenadas2.posZg < Zprofundidad) Zprofundidad = coordenadas2.posZg;
            if (coordenadas3.posZg < Zprofundidad) Zprofundidad = coordenadas3.posZg;
            if (coordenadas4.posZg < Zprofundidad) Zprofundidad = coordenadas4.posZg;
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            coordenadas1.Convierte3Da2D(ZPersona);
            coordenadas2.Convierte3Da2D(ZPersona);
            coordenadas3.Convierte3Da2D(ZPersona);
            coordenadas4.Convierte3Da2D(ZPersona);
        }

        //Cuadra en pantalla
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            coordenadas1.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas2.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas3.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas4.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
        }


        //Hace el gráfico del polígono
        public void Dibuja(Graphics lienzo, Pen lapiz, Brush relleno) {
            //Pone un color de fondo al polígono para borrar lo que hay detrás
            Point punto1 = new Point(coordenadas1.pantallaX, coordenadas1.pantallaY);
            Point punto2 = new Point(coordenadas2.pantallaX, coordenadas2.pantallaY);
            Point punto3 = new Point(coordenadas3.pantallaX, coordenadas3.pantallaY);
            Point punto4 = new Point(coordenadas4.pantallaX, coordenadas4.pantallaY);
            Point[] poligono = { punto1, punto2, punto3, punto4 };
            lienzo.FillPolygon(relleno, poligono);

            //Pinta el perímetro del polígono
```

```
            lienzo.DrawLine(lapiz, coordenadas1.pantallaX, coordenadas1.pantallaY, coordenadas2.pantallaX,
coordenadas2.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas2.pantallaX, coordenadas2.pantallaY, coordenadas3.pantallaX,
coordenadas3.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas3.pantallaX, coordenadas3.pantallaY, coordenadas4.pantallaX,
coordenadas4.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas4.pantallaX, coordenadas4.pantallaY, coordenadas1.pantallaX,
coordenadas1.pantallaY);
        }

        public int CompareTo(object obj) {
            Poligono orderToCompare = obj as Poligono;
            if (orderToCompare.Zprofundidad < Zprofundidad) {
                return 1;
            }
            if (orderToCompare.Zprofundidad > Zprofundidad) {
                return -1;
            }

            // The orders are equivalent.
            //https://stackoverflow.com/questions/3309188/how-to-sort-a-listt-by-a-property-in-the-object
            return 0;
        }
    }
}
```

09. 3D/08. Polar3D/Form1.cs

```
//Graficar ecuaciones polares con Radio, Theta y Phi
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Donde almacena los poligonos
        List<Poligono> poligonos;

        //Ángulos de giro
        double AnguloX, AnguloY, AnguloZ;
        int ZPersona;

        //Datos para la ecuación
        double minTheta, minPhi, maxTheta, maxPhi;
        int numLineas;

        //Datos para la pantalla
        int pXmin, pYmin, pXmax, pYmax;

        public Form1() {
            InitializeComponent();
            poligonos = new List<Poligono>();

            //Valores por defecto a los ángulos
            AnguloX = (double)numGiroX.Value;
            AnguloY = (double)numGiroY.Value;
            AnguloZ = (double)numGiroZ.Value;
            ZPersona = 5;

            //Valores para la ecuación
            minTheta = 0;
            maxTheta = 180;
            minPhi = 0;
            maxPhi = 360;
            numLineas = 40;

            //Coordenadas de pantalla
            pXmin = 0;
            pYmin = 0;
            pXmax = 900;
            pYmax = 630;

            Logica();
        }

        private void numGiroX_ValueChanged(object sender, System.EventArgs e) {
            AnguloX = Convert.ToDouble(numGiroX.Value);
```

```
        Logica();
        Refresh();
    }

    private void numGiroY_ValueChanged(object sender, EventArgs e) {
        AnguloY = Convert.ToDouble(numGiroY.Value);
        Logica();
        Refresh();
    }

    private void numGiroZ_ValueChanged(object sender, EventArgs e) {
        AnguloZ = Convert.ToDouble(numGiroZ.Value);
        Logica();
        Refresh();
    }

    public void Logica() {
        //Calcula los puntos de la ecuación y así formar los polígonos
        poligonos.Clear();
        double incTheta = (maxTheta - minTheta) / numLineas;
        double incPhi = (maxPhi - minPhi) / numLineas;

        //Mínimos y máximos para normalizar
        double minimoX = double.MaxValue;
        double minimoY = double.MaxValue;
        double minimoZ = double.MaxValue;
        double maximoX = double.MinValue;
        double maximoY = double.MinValue;
        double maximoZ = double.MinValue;

        for (double valTheta = minTheta; valTheta <= maxTheta; valTheta += incTheta)
            for (double valPhi = minPhi; valPhi <= maxPhi; valPhi += incPhi) {

                //Primer punto del polígono
                double theta1 = valTheta * Math.PI / 180;
                double phi1 = valPhi * Math.PI / 180;
                double r1 = Ecuacion(theta1, phi1);
                if (double.IsNaN(r1) || double.IsInfinity(r1)) r1 = 0;

                double x1 = r1 * Math.Cos(phi1) * Math.Sin(theta1);
                double y1 = r1 * Math.Sin(phi1) * Math.Sin(theta1);
                double z1 = r1 * Math.Cos(theta1);

                //Segundo punto del polígono
                double theta2 = (valTheta + incTheta) * Math.PI / 180;
                double phi2 = valPhi * Math.PI / 180;
                double r2 = Ecuacion(theta2, phi2);
                if (double.IsNaN(r2) || double.IsInfinity(r2)) r2 = 0;

                double x2 = r2 * Math.Cos(phi2) * Math.Sin(theta2);
                double y2 = r2 * Math.Sin(phi2) * Math.Sin(theta2);
                double z2 = r2 * Math.Cos(theta2);

                //Tercer punto del polígono
                double theta3 = (valTheta + incTheta) * Math.PI / 180;
                double phi3 = (valPhi + incPhi) * Math.PI / 180;
                double r3 = Ecuacion(theta3, phi3);
                if (double.IsNaN(r3) || double.IsInfinity(r3)) r3 = 0;

                double x3 = r3 * Math.Cos(phi3) * Math.Sin(theta3);
                double y3 = r3 * Math.Sin(phi3) * Math.Sin(theta3);
                double z3 = r3 * Math.Cos(theta3);

                //Cuarto punto del polígono
                double theta4 = valTheta * Math.PI / 180;
                double phi4 = (valPhi + incPhi) * Math.PI / 180;
                double r4 = Ecuacion(theta4, phi4);
                if (double.IsNaN(r4) || double.IsInfinity(r4)) r4 = 0;

                double x4 = r4 * Math.Cos(phi4) * Math.Sin(theta4);
                double y4 = r4 * Math.Sin(phi4) * Math.Sin(theta4);
                double z4 = r4 * Math.Cos(theta4);

                if (x1 < minimoX) minimoX = x1;
                if (x2 < minimoX) minimoX = x2;
                if (x3 < minimoX) minimoX = x3;
                if (x4 < minimoX) minimoX = x4;

                if (y1 < minimoY) minimoY = y1;
```

```
                if (y2 < minimoY) minimoY = y2;
                if (y3 < minimoY) minimoY = y3;
                if (y4 < minimoY) minimoY = y4;

                if (z1 < minimoZ) minimoZ = z1;
                if (z2 < minimoZ) minimoZ = z2;
                if (z3 < minimoZ) minimoZ = z3;
                if (z4 < minimoZ) minimoZ = z4;

                if (x1 > maximoX) maximoX = x1;
                if (x2 > maximoX) maximoX = x2;
                if (x3 > maximoX) maximoX = x3;
                if (x4 > maximoX) maximoX = x4;

                if (y1 > maximoY) maximoY = y1;
                if (y2 > maximoY) maximoY = y2;
                if (y3 > maximoY) maximoY = y3;
                if (y4 > maximoY) maximoY = y4;

                if (z1 > maximoZ) maximoZ = z1;
                if (z2 > maximoZ) maximoZ = z2;
                if (z3 > maximoZ) maximoZ = z3;
                if (z4 > maximoZ) maximoZ = z4;

                poligonos.Add(new Poligono(x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4));
            }

            //Luego normaliza los puntos X,Y,Z para que queden entre -0.5 y 0.5
            for (int cont = 0; cont < poligonos.Count; cont++) {
                poligonos[cont].coordenadas1.posX = (poligonos[cont].coordenadas1.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas1.posY = (poligonos[cont].coordenadas1.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas1.posZ = (poligonos[cont].coordenadas1.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas2.posX = (poligonos[cont].coordenadas2.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas2.posY = (poligonos[cont].coordenadas2.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas2.posZ = (poligonos[cont].coordenadas2.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas3.posX = (poligonos[cont].coordenadas3.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas3.posY = (poligonos[cont].coordenadas3.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas3.posZ = (poligonos[cont].coordenadas3.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas4.posX = (poligonos[cont].coordenadas4.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas4.posY = (poligonos[cont].coordenadas4.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas4.posZ = (poligonos[cont].coordenadas4.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;
            }

            //Gira y convierte los puntos espaciales en puntos proyectados al plano
            for (int cont = 0; cont < poligonos.Count; cont++) {
                poligonos[cont].GiroXYZ(AnguloX, AnguloY, AnguloZ);
                poligonos[cont].Convierte3Da2D(ZPersona);
            }

            //Constantes para cuadrar el gráfico en pantalla
            //Se obtuvieron probando los ángulos de giro X,Y,Z de 0 a 360 y ZPersona
            double XplanoMax = 0.87931543769177811;
            double XplanoMin = -0.87931543769177811;
            double YplanoMax = 0.87931539875237918;
            double YplanoMin = -0.87931539875237918;

            //Constantes de transformación
            double ConstanteX = (pXmax - pXmin) / (XplanoMax - XplanoMin);
            double ConstanteY = (pYmax - pYmin) / (YplanoMax - YplanoMin);

            //Cuadra los polígonos en pantalla
            for (int cont = 0; cont < poligonos.Count; cont++)
                poligonos[cont].CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin,
pYmin);
```

```
            //Ordena del polígono más alejado al más cercano, de esa manera los polígonos de adelante
            //son visibles y los de atrás son borrados.
            poligonos.Sort();
        }

        //Aquí está la ecuación polar 3D que se desee graficar con variable Theta y PHI
        public double Ecuacion(double Theta, double Phi) {
            return Math.Cos(Phi)+Math.Sin(Theta);
        }

        //Pinta el gráfico generado por la ecuación
        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics lienzo = e.Graphics;
            Pen lapiz = new Pen(Color.Black, 1);
            Brush relleno = new SolidBrush(Color.White);
            for (int cont = 0; cont < poligonos.Count; cont++)
                poligonos[cont].Dibuja(lienzo, lapiz, relleno);
        }
    }
}
```

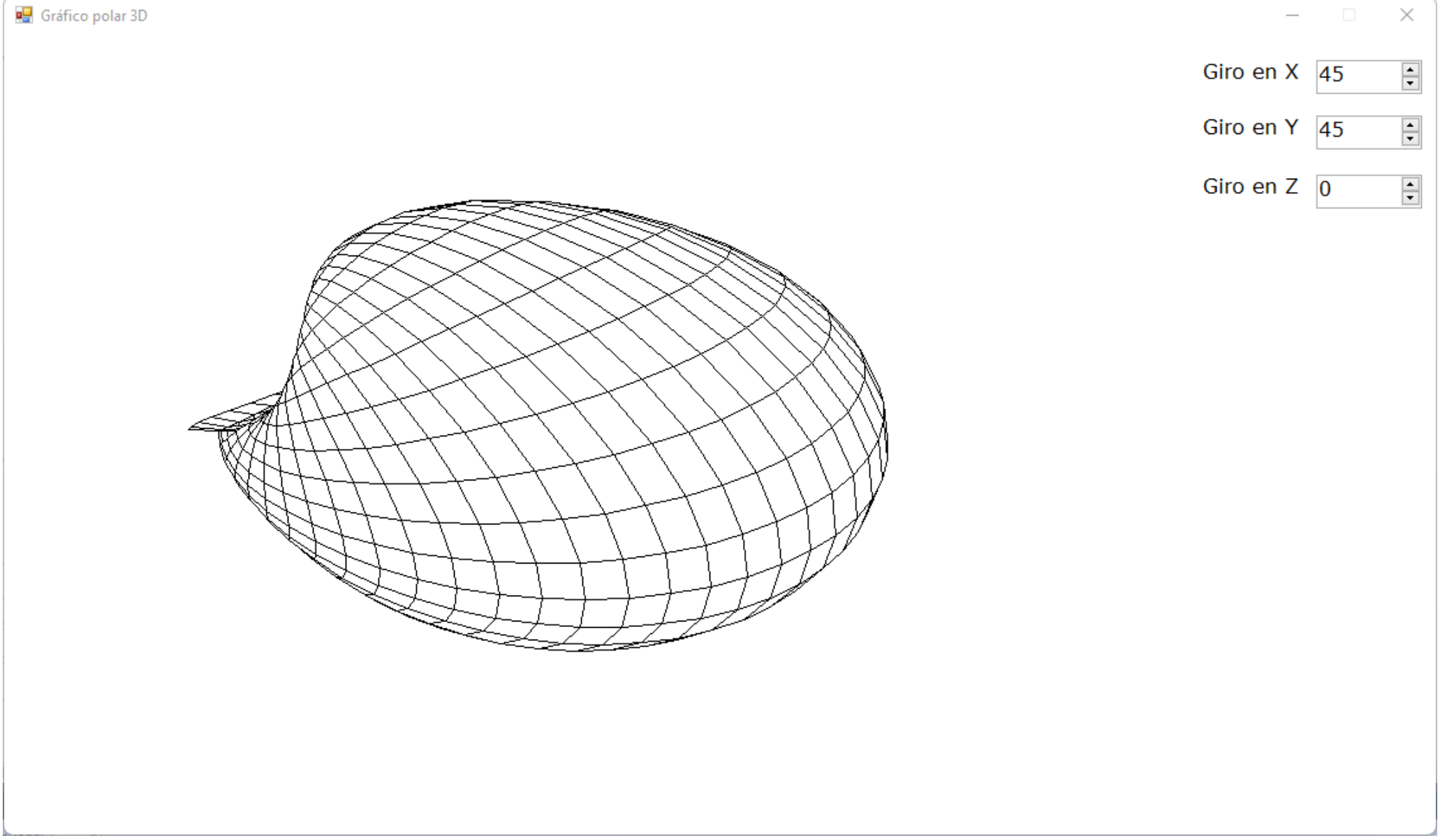


Ilustración 92: Gráfico polar 3D

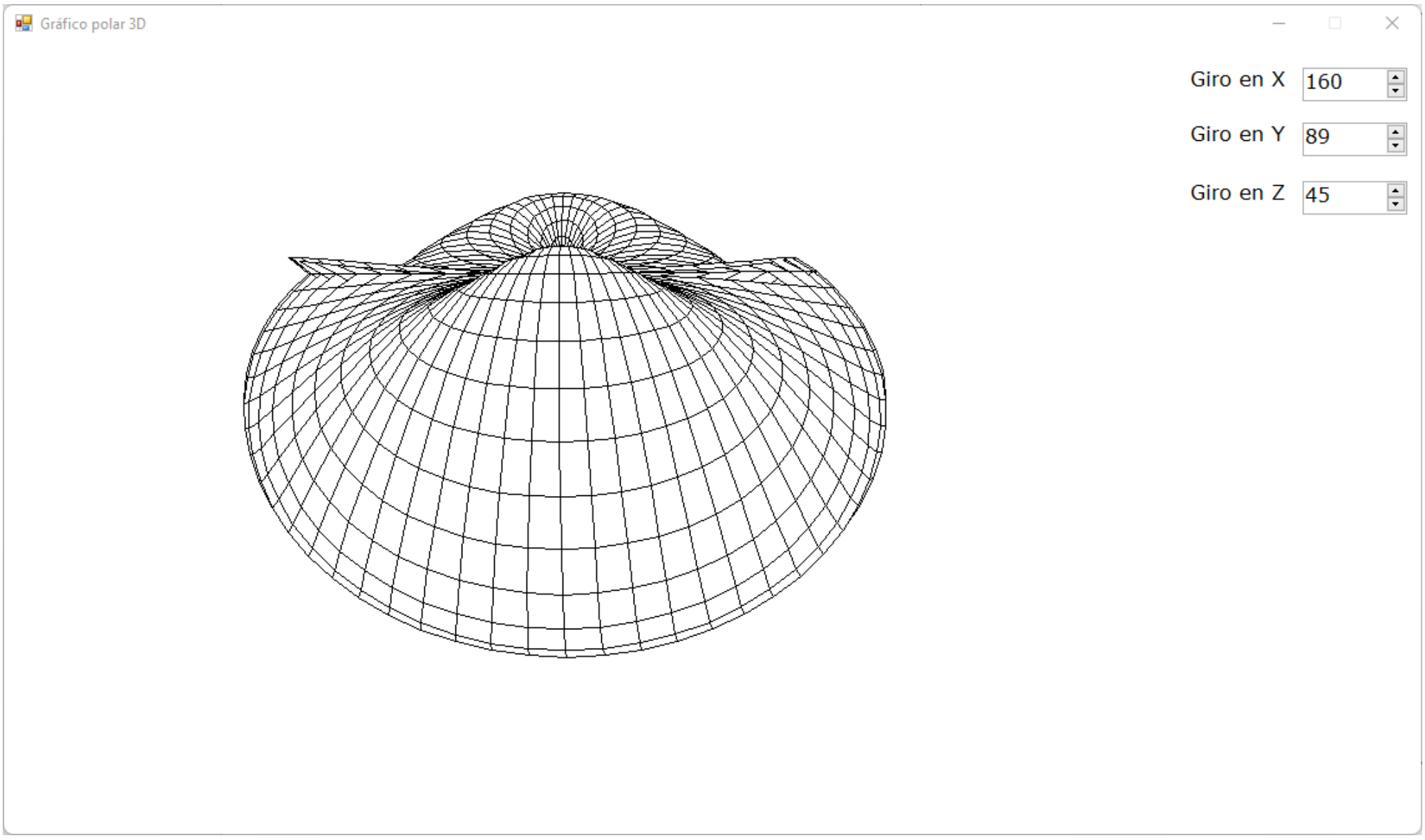


Ilustración 93: Gráfico Polar 3D

Gráfico Polar en 3D animado

Cuando tenemos una ecuación polar del tipo r=F(Θ,φ,t), tenemos una ecuación con tres variables independientes (una de ellas es tiempo) y una variable dependiente. Su representación es en 3D animada. Un ejemplo de este tipo de ecuación:

$$r = Cos(\varphi * t) + Sen(\theta * t)$$

Los pasos para hacer el gráfico son los siguientes:

Paso 0: Saber el valor del tiempo mínimo y el valor del tiempo máximo. Se hace uso de un control "timer" que cada vez que se dispara el evento "Tick" varía el valor de T desde el mínimo hasta el máximo paso a paso, una vez alcanzado el máximo se decrementa paso a paso hasta el mínimo y repite el ciclo. De resto es igual a hacer un gráfico de ecuación tipo r=F(Θ,φ).

09. 3D/09. Polar3DAnimado/Puntos.cs

```
using System;

namespace Graficos {
    internal class Puntos {
        //Coordenada espacial original
        public double posX, posY, posZ;

        //Coordenada espacial después de girar
        public double posXg, posYg, posZg;

        //Coordenada proyectada
        public double planoX, planoY;

        //Coordenada cuadrada en pantalla
        public int pantallaX, pantallaY;

        //Constructor
        public Puntos(double posX, double posY, double posZ) {
            this.posX = posX;
            this.posY = posY;
            this.posZ = posZ;
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
            double angXr = angX * Math.PI / 180;
            double angYr = angY * Math.PI / 180;
            double angZr = angZ * Math.PI / 180;

            double CosX = Math.Cos(angXr);
            double SinX = Math.Sin(angXr);
            double CosY = Math.Cos(angYr);
            double SinY = Math.Sin(angYr);
            double CosZ = Math.Cos(angZr);
            double SinZ = Math.Sin(angZr);

            //Matriz de Rotación
            //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
            double[,] Matriz = new double[3, 3] {
                { CosY * CosZ, -CosX * SinZ + SinX * SinY * CosZ, SinX * SinZ + CosX * SinY * CosZ},
                { CosY * SinZ, CosX * CosZ + SinX * SinY * SinZ, -SinX * CosZ + CosX * SinY * SinZ},
                {-SinY, SinX * CosY, CosX * CosY }
            };

            posXg = posX * Matriz[0, 0] + posY * Matriz[1, 0] + posZ * Matriz[2, 0];
            posYg = posX * Matriz[0, 1] + posY * Matriz[1, 1] + posZ * Matriz[2, 1];
            posZg = posX * Matriz[0, 2] + posY * Matriz[1, 2] + posZ * Matriz[2, 2];
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            planoX = posXg * ZPersona / (ZPersona - posZg);
            planoY = posYg * ZPersona / (ZPersona - posZg);
        }

        //Cuadra los puntos en pantalla
```

```
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            pantallaX = Convert.ToInt32(ConstanteX * (planoX - XplanoMin) + pXmin);
            pantallaY = Convert.ToInt32(ConstanteY * (planoY - YplanoMin) + pYmin);
        }
    }
}
```

09. 3D/09. Polar3DAnimado/Poligono.cs

```
using System;
using System.Drawing;

namespace Graficos {
    internal class Poligono : IComparable {
        //Un polígono son cuatro(4) coordenadas espaciales
        public Puntos coordenadas1, coordenadas2, coordenadas3, coordenadas4;
        private double Zprofundidad;

        public Poligono(double X1, double Y1, double Z1, double X2, double Y2, double Z2, double X3,
double Y3, double Z3, double X4, double Y4, double Z4) {
            coordenadas1 = new Puntos(X1, Y1, Z1);
            coordenadas2 = new Puntos(X2, Y2, Z2);
            coordenadas3 = new Puntos(X3, Y3, Z3);
            coordenadas4 = new Puntos(X4, Y4, Z4);
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
            coordenadas1.GiroXYZ(angX, angY, angZ);
            coordenadas2.GiroXYZ(angX, angY, angZ);
            coordenadas3.GiroXYZ(angX, angY, angZ);
            coordenadas4.GiroXYZ(angX, angY, angZ);
            Zprofundidad = double.MaxValue;
            if (coordenadas1.posZg < Zprofundidad) Zprofundidad = coordenadas1.posZg;
            if (coordenadas2.posZg < Zprofundidad) Zprofundidad = coordenadas2.posZg;
            if (coordenadas3.posZg < Zprofundidad) Zprofundidad = coordenadas3.posZg;
            if (coordenadas4.posZg < Zprofundidad) Zprofundidad = coordenadas4.posZg;
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            coordenadas1.Convierte3Da2D(ZPersona);
            coordenadas2.Convierte3Da2D(ZPersona);
            coordenadas3.Convierte3Da2D(ZPersona);
            coordenadas4.Convierte3Da2D(ZPersona);
        }

        //Cuadra en pantalla
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            coordenadas1.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas2.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas3.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas4.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
        }


        //Hace el gráfico del polígono
        public void Dibuja(Graphics lienzo, Pen lapiz, Brush relleno) {
            //Pone un color de fondo al polígono para borrar lo que hay detrás
            Point punto1 = new Point(coordenadas1.pantallaX, coordenadas1.pantallaY);
            Point punto2 = new Point(coordenadas2.pantallaX, coordenadas2.pantallaY);
            Point punto3 = new Point(coordenadas3.pantallaX, coordenadas3.pantallaY);
            Point punto4 = new Point(coordenadas4.pantallaX, coordenadas4.pantallaY);
            Point[] poligono = { punto1, punto2, punto3, punto4 };
            lienzo.FillPolygon(relleno, poligono);

            //Pinta el perímetro del polígono
            lienzo.DrawLine(lapiz, coordenadas1.pantallaX, coordenadas1.pantallaY, coordenadas2.pantallaX,
coordenadas2.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas2.pantallaX, coordenadas2.pantallaY, coordenadas3.pantallaX,
coordenadas3.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas3.pantallaX, coordenadas3.pantallaY, coordenadas4.pantallaX,
coordenadas4.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas4.pantallaX, coordenadas4.pantallaY, coordenadas1.pantallaX,
coordenadas1.pantallaY);
        }
```

```
        public int CompareTo(object obj) {
            Poligono orderToCompare = obj as Poligono;
            if (orderToCompare.Zprofundidad < Zprofundidad) {
                return 1;
            }
            if (orderToCompare.Zprofundidad > Zprofundidad) {
                return -1;
            }

            // The orders are equivalent.
            //https://stackoverflow.com/questions/3309188/how-to-sort-a-listt-by-a-property-in-the-object
            return 0;
        }
    }
}
```

09. 3D/09. Polar3DAnimado/Form1.cs

```
//Graficar ecuaciones polares de dos ángulos theta y phi, añadiendo la variable tiempo

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;


namespace Graficos {
    public partial class Form1 : Form {
        //Para la variable temporal
        private double Tminimo, Tmaximo, Tincrementa, Tiempo;

        //Donde almacena los poligonos
        List<Poligono> poligonos;

        //Ángulos de giro
        double AnguloX, AnguloY, AnguloZ;
        int ZPersona;

        //Datos para la ecuación
        double minTheta, minPhi, maxTheta, maxPhi;
        int numLineas;

        //Datos para la pantalla
        int pXmin, pYmin, pXmax, pYmax;

        public Form1() {
            InitializeComponent();
            poligonos = new List<Poligono>();

            //Valores por defecto a los ángulos
            AnguloX = (double)numGiroX.Value;
            AnguloY = (double)numGiroY.Value;
            AnguloZ = (double)numGiroZ.Value;
            ZPersona = 5;

            //Valores para la ecuación
            minTheta = 0;
            maxTheta = 180;
            minPhi = 0;
            maxPhi = 360;
            numLineas = 40;

            //Inicia el tiempo
            Tminimo = 0.1;
            Tmaximo = 1;
            Tincrementa = 0.01;
            Tiempo = Tminimo;

            //Coordenadas de pantalla
            pXmin = 0;
            pYmin = 0;
            pXmax = 900;
            pYmax = 630;

            Logica();
    }
```

```
private void timerAnimar_Tick(object sender, EventArgs e) {
    Tiempo += Tincrementa;
    if (Tiempo <= Tminimo || Tiempo >= Tmaximo) Tincrementa = -Tincrementa;
    Logica();
    Refresh();
}

private void numGiroX_ValueChanged(object sender, System.EventArgs e) {
    AnguloX = Convert.ToDouble(numGiroX.Value);
    Logica();
    Refresh();
}

private void numGiroY_ValueChanged(object sender, EventArgs e) {
    AnguloY = Convert.ToDouble(numGiroY.Value);
    Logica();
    Refresh();
}

private void numGiroZ_ValueChanged(object sender, EventArgs e) {
    AnguloZ = Convert.ToDouble(numGiroZ.Value);
    Logica();
    Refresh();
}

public void Logica() {
    //Calcula los puntos de la ecuación y así formar los polígonos
    poligonos.Clear();
    double incTheta = (maxTheta - minTheta) / numLineas;
    double incPhi = (maxPhi - minPhi) / numLineas;

    //Mínimos y máximos para normalizar
    double minimoX = double.MaxValue;
    double minimoY = double.MaxValue;
    double minimoZ = double.MaxValue;
    double maximoX = double.MinValue;
    double maximoY = double.MinValue;
    double maximoZ = double.MinValue;

    for (double valTheta = minTheta; valTheta <= maxTheta; valTheta += incTheta)
        for (double valPhi = minPhi; valPhi <= maxPhi; valPhi += incPhi) {

            //Primer punto del polígono
            double theta1 = valTheta * Math.PI / 180;
            double phi1 = valPhi * Math.PI / 180;
            double r1 = Ecuacion(theta1, phi1, Tiempo);
            if (double.IsNaN(r1) || double.IsInfinity(r1)) r1 = 0;

            double x1 = r1 * Math.Cos(phi1) * Math.Sin(theta1);
            double y1 = r1 * Math.Sin(phi1) * Math.Sin(theta1);
            double z1 = r1 * Math.Cos(theta1);

            //Segundo punto del polígono
            double theta2 = (valTheta + incTheta) * Math.PI / 180;
            double phi2 = valPhi * Math.PI / 180;
            double r2 = Ecuacion(theta2, phi2, Tiempo);
            if (double.IsNaN(r2) || double.IsInfinity(r2)) r2 = 0;

            double x2 = r2 * Math.Cos(phi2) * Math.Sin(theta2);
            double y2 = r2 * Math.Sin(phi2) * Math.Sin(theta2);
            double z2 = r2 * Math.Cos(theta2);

            //Tercer punto del polígono
            double theta3 = (valTheta + incTheta) * Math.PI / 180;
            double phi3 = (valPhi + incPhi) * Math.PI / 180;
            double r3 = Ecuacion(theta3, phi3, Tiempo);
            if (double.IsNaN(r3) || double.IsInfinity(r3)) r3 = 0;

            double x3 = r3 * Math.Cos(phi3) * Math.Sin(theta3);
            double y3 = r3 * Math.Sin(phi3) * Math.Sin(theta3);
            double z3 = r3 * Math.Cos(theta3);

            //Cuarto punto del polígono
            double theta4 = valTheta * Math.PI / 180;
            double phi4 = (valPhi + incPhi) * Math.PI / 180;
            double r4 = Ecuacion(theta4, phi4, Tiempo);
            if (double.IsNaN(r4) || double.IsInfinity(r4)) r4 = 0;

            double x4 = r4 * Math.Cos(phi4) * Math.Sin(theta4);
```

```
                double y4 = r4 * Math.Sin(phi4) * Math.Sin(theta4);
                double z4 = r4 * Math.Cos(theta4);
                if (x1 < minimoX) minimoX = x1;
                if (x2 < minimoX) minimoX = x2;
                if (x3 < minimoX) minimoX = x3;
                if (x4 < minimoX) minimoX = x4;

                if (y1 < minimoY) minimoY = y1;
                if (y2 < minimoY) minimoY = y2;
                if (y3 < minimoY) minimoY = y3;
                if (y4 < minimoY) minimoY = y4;

                if (z1 < minimoZ) minimoZ = z1;
                if (z2 < minimoZ) minimoZ = z2;
                if (z3 < minimoZ) minimoZ = z3;
                if (z4 < minimoZ) minimoZ = z4;

                if (x1 > maximoX) maximoX = x1;
                if (x2 > maximoX) maximoX = x2;
                if (x3 > maximoX) maximoX = x3;
                if (x4 > maximoX) maximoX = x4;

                if (y1 > maximoY) maximoY = y1;
                if (y2 > maximoY) maximoY = y2;
                if (y3 > maximoY) maximoY = y3;
                if (y4 > maximoY) maximoY = y4;

                if (z1 > maximoZ) maximoZ = z1;
                if (z2 > maximoZ) maximoZ = z2;
                if (z3 > maximoZ) maximoZ = z3;
                if (z4 > maximoZ) maximoZ = z4;

                poligonos.Add(new Poligono(x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4));
            }

            //Luego normaliza los puntos X,Y,Z para que queden entre -0.5 y 0.5
            for (int cont = 0; cont < poligonos.Count; cont++) {
                poligonos[cont].coordenadas1.posX = (poligonos[cont].coordenadas1.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas1.posY = (poligonos[cont].coordenadas1.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas1.posZ = (poligonos[cont].coordenadas1.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas2.posX = (poligonos[cont].coordenadas2.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas2.posY = (poligonos[cont].coordenadas2.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas2.posZ = (poligonos[cont].coordenadas2.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas3.posX = (poligonos[cont].coordenadas3.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas3.posY = (poligonos[cont].coordenadas3.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas3.posZ = (poligonos[cont].coordenadas3.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas4.posX = (poligonos[cont].coordenadas4.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas4.posY = (poligonos[cont].coordenadas4.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas4.posZ = (poligonos[cont].coordenadas4.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;
            }

            //Gira y convierte los puntos espaciales en puntos proyectados al plano
            for (int cont = 0; cont < poligonos.Count; cont++) {
                poligonos[cont].GiroXYZ(AnguloX, AnguloY, AnguloZ);
                poligonos[cont].Convierte3Da2D(ZPersona);
            }

            //Constantes para cuadrar el gráfico en pantalla
            //Se obtuvieron probando los ángulos de giro X,Y,Z de 0 a 360 y ZPersona
            double XplanoMax = 0.87931543769177811;
            double XplanoMin = -0.87931543769177811;
            double YplanoMax = 0.87931539875237918;
            double YplanoMin = -0.87931539875237918;
```

```
            //Constantes de transformación
            double ConstanteX = (pXmax - pXmin) / (XplanoMax - XplanoMin);
            double ConstanteY = (pYmax - pYmin) / (YplanoMax - YplanoMin);

            //Cuadra los polígonos en pantalla
            for (int cont = 0; cont < poligonos.Count; cont++)
                poligonos[cont].CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin,
pYmin);

            //Ordena del polígono más alejado al más cercano, de esa manera los polígonos de adelante
            //son visibles y los de atrás son borrados.
            poligonos.Sort();
        }

        //Aquí está la ecuación polar 3D que se desee graficar con variable Theta, PHI y Tiempo
        public double Ecuacion(double Theta, double Phi, double t) {
            return Math.Cos(Phi * t) + Math.Sin(Theta * t);
        }

        //Pinta el gráfico generado por la ecuación
        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics lienzo = e.Graphics;
            Pen lapiz = new Pen(Color.Black, 1);
            Brush relleno = new SolidBrush(Color.White);
            for (int cont = 0; cont < poligonos.Count; cont++)
                poligonos[cont].Dibuja(lienzo, lapiz, relleno);
        }
    }
}
```

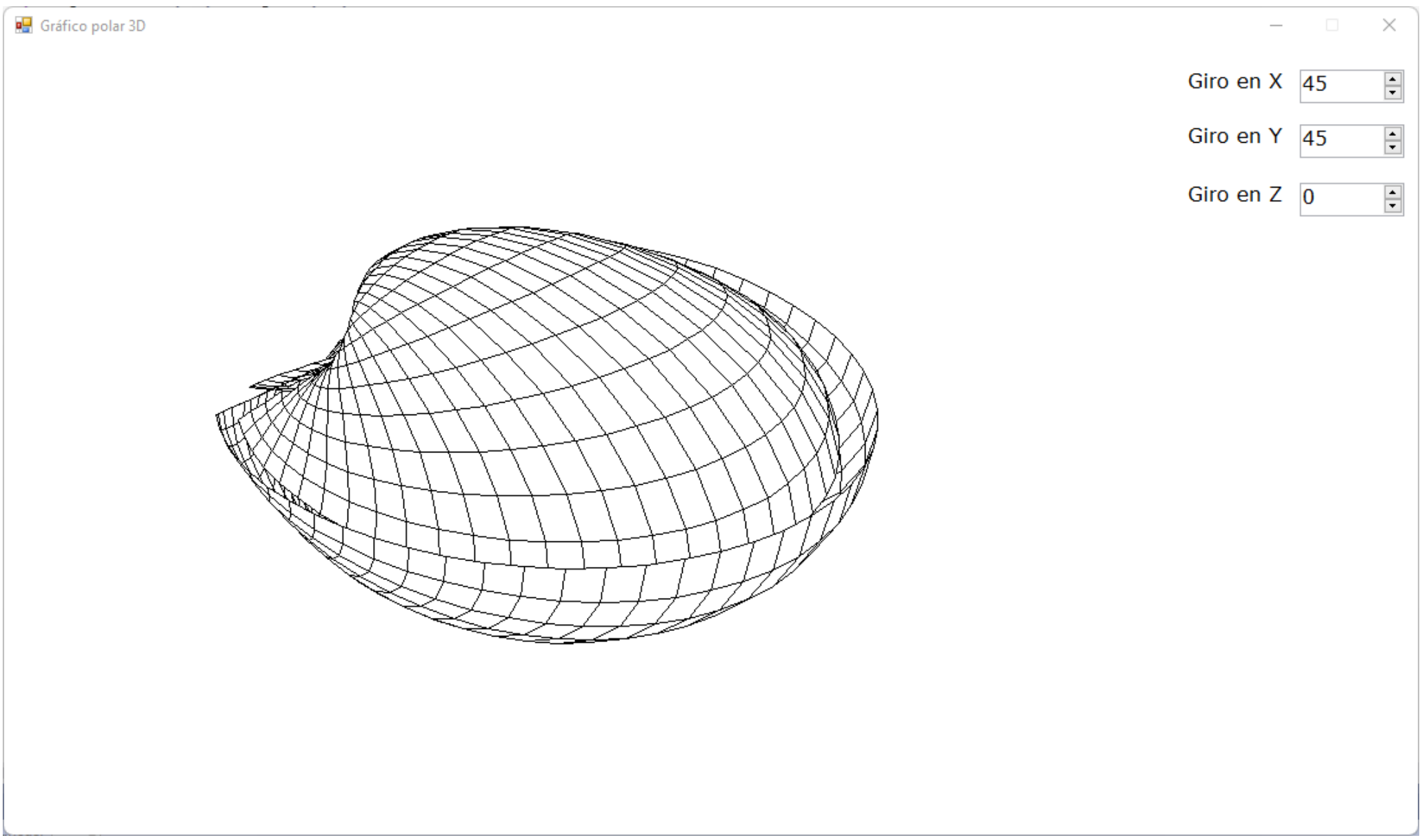


Ilustración 94: Un "frame" de la animación del gráfico polar en 3D

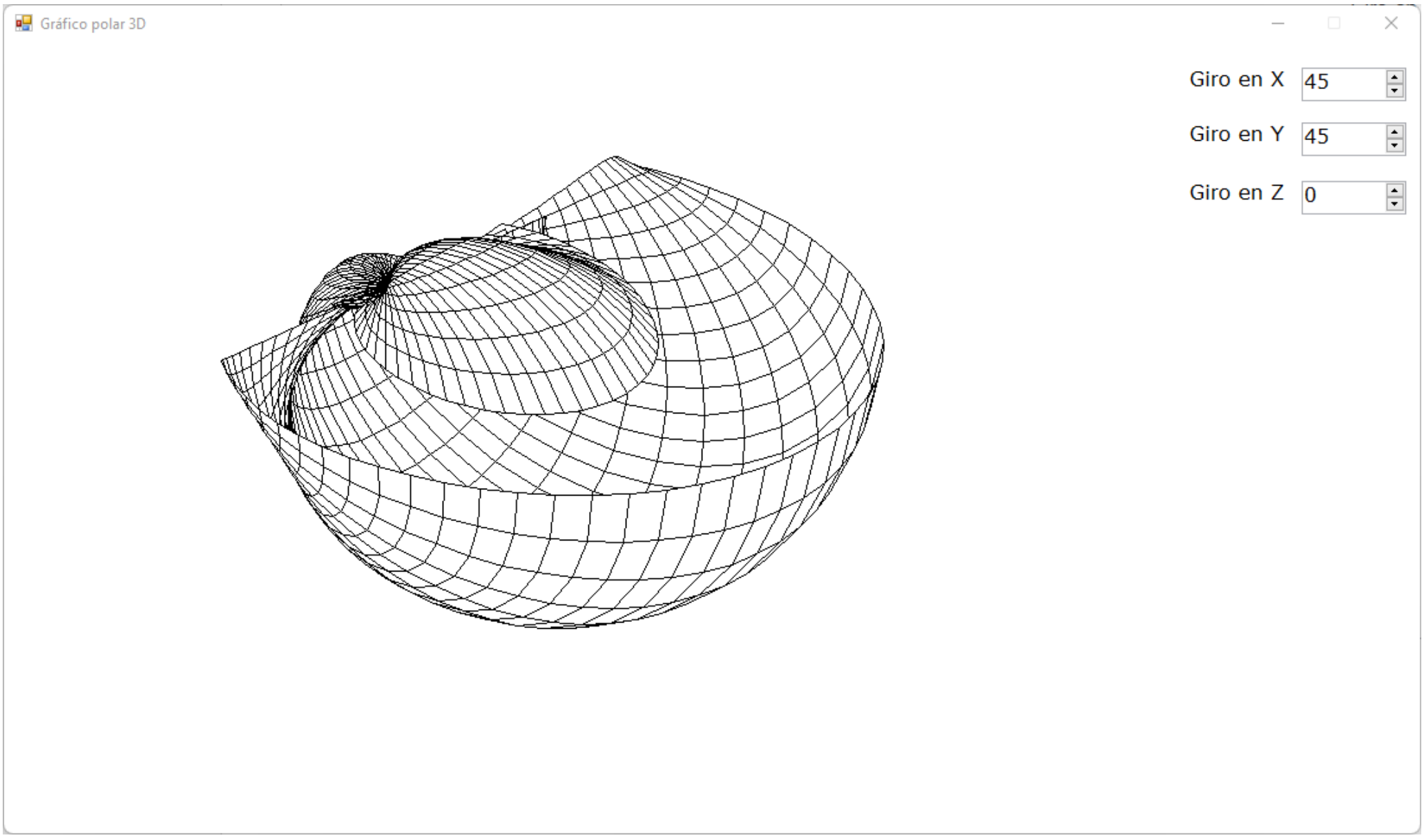


Ilustración 95: Un "frame" de la animación del gráfico polar en 3D

Sólido de revolución

Un sólido de revolución puede nacer de tomar una ecuación del tipo Y=F(X), y luego poner a girar el gráfico resultante en el eje X. Ver mas en: https://es.wikipedia.org/wiki/S%C3%B3lido_de_revoluci%C3%B3n

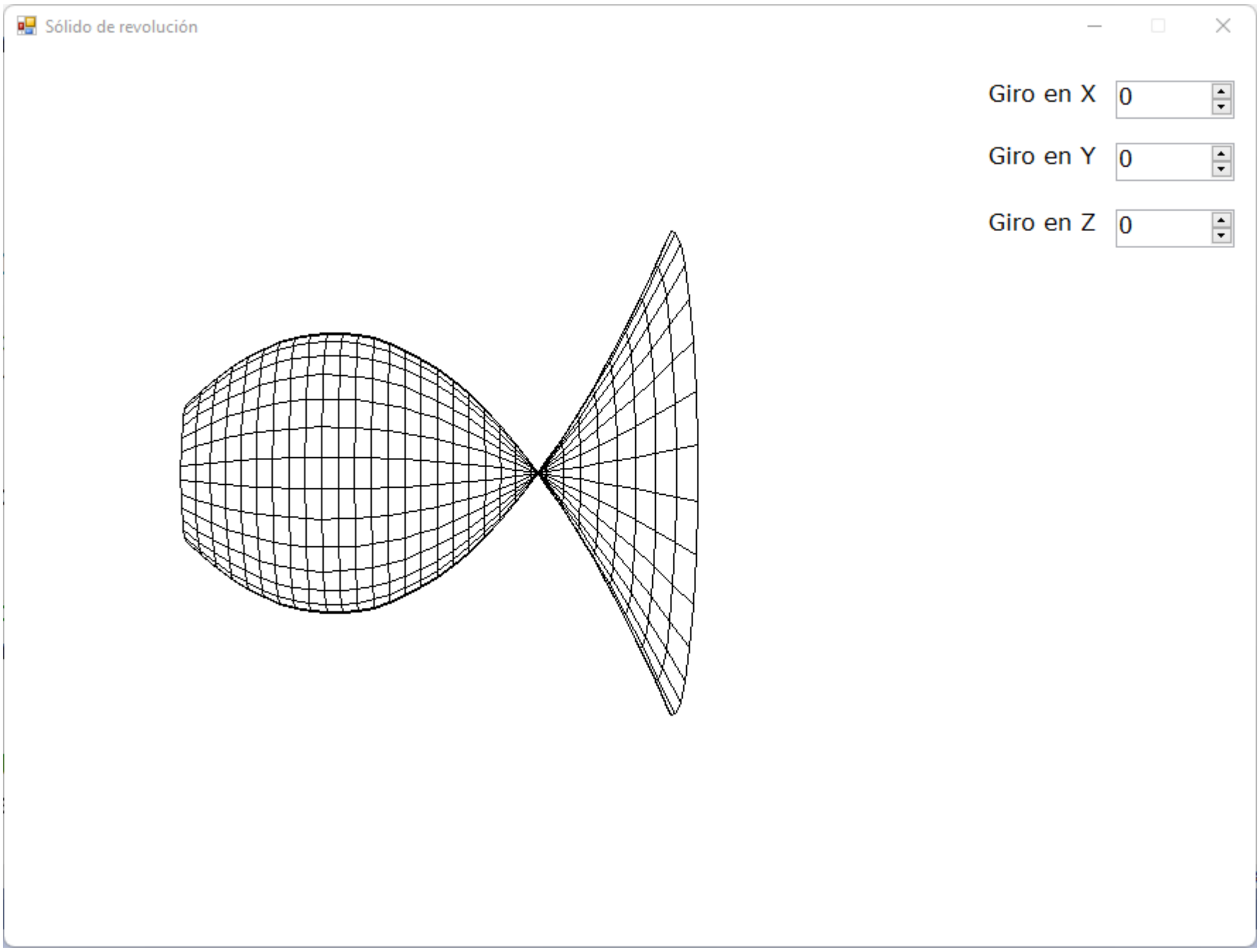


Ilustración 96: Sólido de revolución

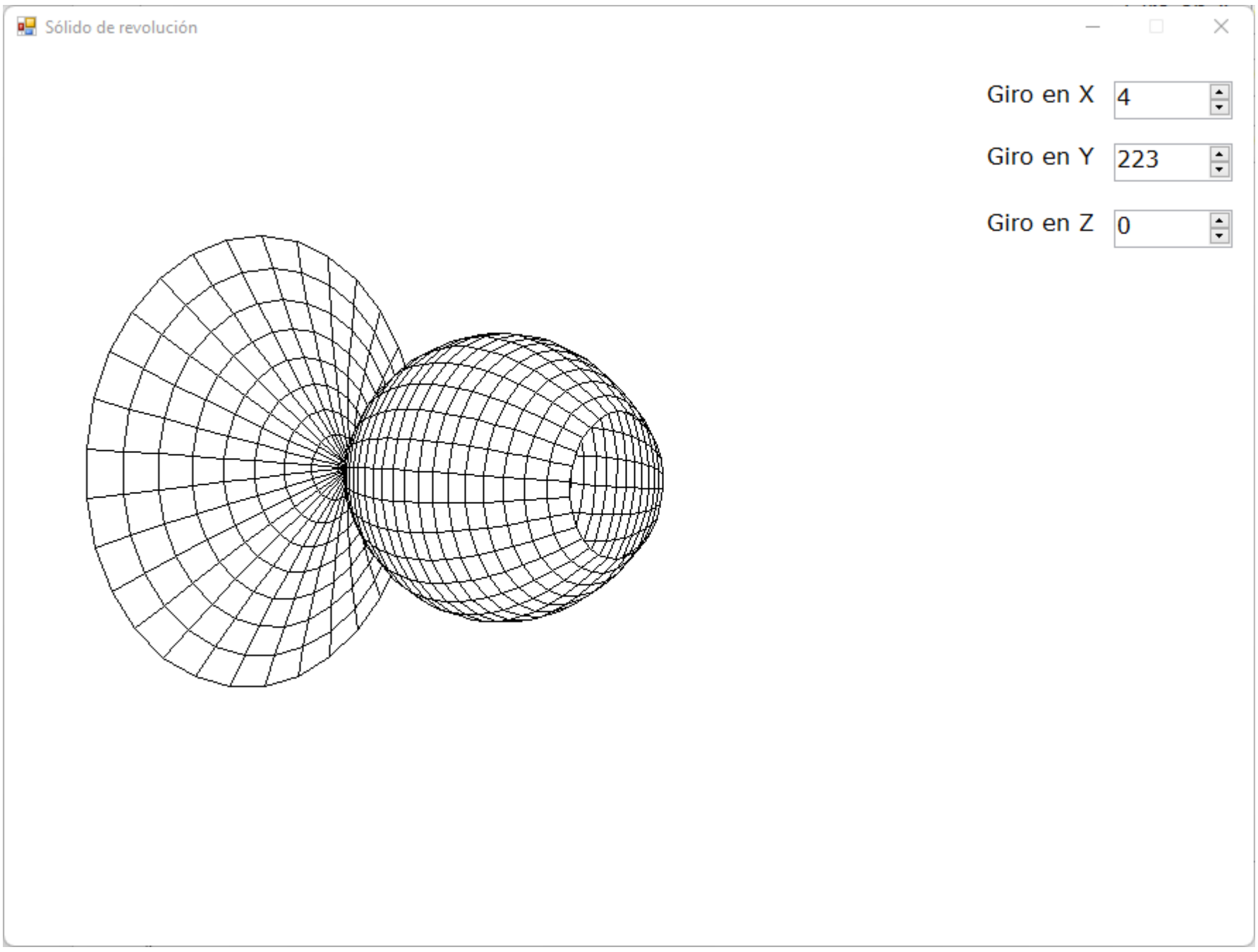


Ilustración 97: Girando el sólido de revolución

Los pasos para hacer el gráfico son los siguientes:

Paso 1: Saber el valor donde inicia X hasta donde termina. Así se calcula Y.

Paso 2: Ahora se toma desde un ángulo de giro en el eje X de 0 hasta 360.

Paso 3: Se aplica el giro en el eje X y con eso se obtienen nuevos valores de Y y Z. Al final se tiene un conjunto de coordenadas posX, posY, posZ. Se va a hacer uso de polígonos, por lo que se calculan cuatro coordenadas para formar el polígono.

Paso 4: Se normalizan los valores de posX, posY, posZ para que queden entre 0 y 1, luego se le resta -0.5 ¿Para qué? Para que los puntos (realmente polígonos) queden contenidos dentro de un cubo de lado=1, cuyo centro está en 0,0,0 . Ver los dos temas anteriores como se muestra el cubo.

Paso 5: Luego se aplica el giro en los tres ángulos. Se obtiene posXg, posYg, posZg

Paso 6: Se ordenan los polígonos del más profundo (menor valor de posZg) al más superficial (mayor valor de posZg). Por esa razón, la clase Polígono hereda de IComparable

Paso 7: Con Xg, Yg, Zg se proyecta a la pantalla, obteniéndose el planoX, planoY.

Paso 8: Con planoX, planoY se aplican las constantes que se calcularon para proyectar el cubo y se obtiene los datos de pantalla, es decir, pantallaX, pantallaY

Paso 9: Se dibujan los polígonos, primero rellenándolos con el color del fondo y luego se gráfica el perímetro de ese polígono.

09. 3D/10. SolidoRevolucion/Puntos.cs

```
using System;

namespace Graficos {
    internal class Puntos {
        //Coordenada espacial original
        public double posX, posY, posZ;

        //Coordenada espacial después de girar
        public double posXg, posYg, posZg;

        //Coordenada proyectada
        public double planoX, planoY;
```

```
        //Coordenada cuadrada en pantalla
        public int pantallaX, pantallaY;

        //Constructor
        public Puntos(double posX, double posY, double posZ) {
            this.posX = posX;
            this.posY = posY;
            this.posZ = posZ;
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
            double angXr = angX * Math.PI / 180;
            double angYr = angY * Math.PI / 180;
            double angZr = angZ * Math.PI / 180;

            double CosX = Math.Cos(angXr);
            double SinX = Math.Sin(angXr);
            double CosY = Math.Cos(angYr);
            double SinY = Math.Sin(angYr);
            double CosZ = Math.Cos(angZr);
            double SinZ = Math.Sin(angZr);

            //Matriz de Rotación
            //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
            double[,] Matriz = new double[3, 3] {
                { CosY * CosZ, -CosX * SinZ + SinX * SinY * CosZ, SinX * SinZ + CosX * SinY * CosZ},
                { CosY * SinZ, CosX * CosZ + SinX * SinY * SinZ, -SinX * CosZ + CosX * SinY * SinZ},
                {-SinY, SinX * CosY, CosX * CosY }
            };

            posXg = posX * Matriz[0, 0] + posY * Matriz[1, 0] + posZ * Matriz[2, 0];
            posYg = posX * Matriz[0, 1] + posY * Matriz[1, 1] + posZ * Matriz[2, 1];
            posZg = posX * Matriz[0, 2] + posY * Matriz[1, 2] + posZ * Matriz[2, 2];
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            planoX = posXg * ZPersona / (ZPersona - posZg);
            planoY = posYg * ZPersona / (ZPersona - posZg);
        }

        //Cuadra los puntos en pantalla
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            pantallaX = Convert.ToInt32(ConstanteX * (planoX - XplanoMin) + pXmin);
            pantallaY = Convert.ToInt32(ConstanteY * (planoY - YplanoMin) + pYmin);
        }
    }
}
```

09. 3D/10. SolidoRevolucion/Poligono.cs

```
using System;
using System.Drawing;

namespace Graficos {
    internal class Poligono : IComparable {
        //Un polígono son cuatro(4) coordenadas espaciales
        public Puntos coordenadas1, coordenadas2, coordenadas3, coordenadas4;
        private double centro;

        public Poligono(double X1, double Y1, double Z1, double X2, double Y2, double Z2, double X3,
double Y3, double Z3, double X4, double Y4, double Z4) {
            coordenadas1 = new Puntos(X1, Y1, Z1);
            coordenadas2 = new Puntos(X2, Y2, Z2);
            coordenadas3 = new Puntos(X3, Y3, Z3);
            coordenadas4 = new Puntos(X4, Y4, Z4);
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
            coordenadas1.GiroXYZ(angX, angY, angZ);
            coordenadas2.GiroXYZ(angX, angY, angZ);
            coordenadas3.GiroXYZ(angX, angY, angZ);
            coordenadas4.GiroXYZ(angX, angY, angZ);
            centro = (coordenadas1.posZg + coordenadas2.posZg + coordenadas3.posZg + coordenadas4.posZg) /
4;
```

```
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            coordenadas1.Convierte3Da2D(ZPersona);
            coordenadas2.Convierte3Da2D(ZPersona);
            coordenadas3.Convierte3Da2D(ZPersona);
            coordenadas4.Convierte3Da2D(ZPersona);
        }

        //Cuadra en pantalla
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            coordenadas1.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas2.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas3.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas4.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
        }


        //Hace el gráfico del polígono
        public void Dibuja(Graphics lienzo, Pen lapiz, Brush relleno) {
            //Pone un color de fondo al polígono para borrar lo que hay detrás
            Point punto1 = new Point(coordenadas1.pantallaX, coordenadas1.pantallaY);
            Point punto2 = new Point(coordenadas2.pantallaX, coordenadas2.pantallaY);
            Point punto3 = new Point(coordenadas3.pantallaX, coordenadas3.pantallaY);
            Point punto4 = new Point(coordenadas4.pantallaX, coordenadas4.pantallaY);
            Point[] poligono = { punto1, punto2, punto3, punto4 };
            lienzo.FillPolygon(relleno, poligono);

            //Pinta el perímetro del polígono
            lienzo.DrawLine(lapiz, coordenadas1.pantallaX, coordenadas1.pantallaY, coordenadas2.pantallaX,
coordenadas2.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas2.pantallaX, coordenadas2.pantallaY, coordenadas3.pantallaX,
coordenadas3.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas3.pantallaX, coordenadas3.pantallaY, coordenadas4.pantallaX,
coordenadas4.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas4.pantallaX, coordenadas4.pantallaY, coordenadas1.pantallaX,
coordenadas1.pantallaY);
        }

        public int CompareTo(object obj) {
            Poligono orderToCompare = obj as Poligono;
            if (orderToCompare.centro < centro) {
                return 1;
            }
            if (orderToCompare.centro > centro) {
                return -1;
            }

            // The orders are equivalent.
            //https://stackoverflow.com/questions/3309188/how-to-sort-a-listt-by-a-property-in-the-object
            return 0;
        }
    }
}
```



```
//Sólidos de revolución
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {

        //Donde almacena los poligonos
        List<Poligono> poligonos;

        //Ángulos de giro
        double AnguloX, AnguloY, AnguloZ;
        int ZPersona;

        //Datos para la ecuación
        double minXreal, maxXreal;
        int numLineas;
```

```
//Datos para la pantalla
int pXmin, pYmin, pXmax, pYmax;

public Form1() {
    InitializeComponent();
    poligonos = new List<Poligono>();

    //Valores por defecto a los ángulos
    AnguloX = (double)numGiroX.Value;
    AnguloY = (double)numGiroY.Value;
    AnguloZ = (double)numGiroZ.Value;
    ZPersona = 5;

    //Valores para la ecuación
    minXreal = -2;
    maxXreal = 2;
    numLineas = 30;

    //Coordenadas de pantalla
    pXmin = 0;
    pYmin = 0;
    pXmax = 600;
    pYmax = 600;

    Logica();
}

private void numGiroX_ValueChanged(object sender, System.EventArgs e) {
    AnguloX = Convert.ToDouble(numGiroX.Value);
    Logica();
    Refresh();
}

private void numGiroY_ValueChanged(object sender, EventArgs e) {
    AnguloY = Convert.ToDouble(numGiroY.Value);
    Logica();
    Refresh();
}

private void numGiroZ_ValueChanged(object sender, EventArgs e) {
    AnguloZ = Convert.ToDouble(numGiroZ.Value);
    Logica();
    Refresh();
}

public void Logica() {
    //Calcula los puntos de la ecuación y así formar los polígonos
    poligonos.Clear();
    double incrAngulo = 360 / numLineas;
    double incrXreal = (maxXreal - minXreal) / numLineas;

    //Mínimos y máximos para normalizar
    double minimoX = double.MaxValue;
    double minimoY = double.MaxValue;
    double minimoZ = double.MaxValue;
    double maximoX = double.MinValue;
    double maximoY = double.MinValue;
    double maximoZ = double.MinValue;

    for (double angulo = 0; angulo < 360; angulo += incrAngulo)
        for (double valXreal = minXreal; valXreal <= maxXreal; valXreal += incrXreal) {

            //Primer punto
            double X1 = valXreal;
            double Y1 = Ecuacion(X1);
            if (double.IsNaN(Y1) || double.IsInfinity(Y1)) Y1 = 0;

            //Hace giro
            double X1g = X1;
            double Y1g = Y1 * Math.Cos(angulo* Math.PI / 180);
            double Z1g = Y1 * -Math.Sin(angulo * Math.PI / 180);

            //Segundo punto
            double X2 = valXreal + incrXreal;
            double Y2 = Ecuacion(X2);
            if (double.IsNaN(Y2) || double.IsInfinity(Y2)) Y2 = 0;

            //Hace giro
```

```
                double X2g = X2;
                double Y2g = Y2 * Math.Cos(angulo * Math.PI / 180);
                double Z2g = Y2 * -Math.Sin(angulo * Math.PI / 180);

                //Tercer punto ya girado
                double X3g = X2;
                double Y3g = Y2 * Math.Cos((angulo+incrAngulo) * Math.PI / 180);
                double Z3g = Y2 * -Math.Sin((angulo + incrAngulo) * Math.PI / 180);

                //Cuarto punto ya girado
                double X4g = X1;
                double Y4g = Y1 * Math.Cos((angulo + incrAngulo) * Math.PI / 180);
                double Z4g = Y1 * -Math.Sin((angulo + incrAngulo) * Math.PI / 180);

                //Obtener los valores extremos para poder normalizar
                if (X1g < minimoX) minimoX = X1g;
                if (X2g < minimoX) minimoX = X2g;
                if (X3g < minimoX) minimoX = X3g;
                if (X4g < minimoX) minimoX = X4g;

                if (Y1g < minimoY) minimoY = Y1g;
                if (Y2g < minimoY) minimoY = Y2g;
                if (Y3g < minimoY) minimoY = Y3g;
                if (Y4g < minimoY) minimoY = Y4g;

                if (Z1g < minimoZ) minimoZ = Z1g;
                if (Z2g < minimoZ) minimoZ = Z2g;
                if (Z3g < minimoZ) minimoZ = Z3g;
                if (Z4g < minimoZ) minimoZ = Z4g;

                if (X1g > maximoX) maximoX = X1g;
                if (X2g > maximoX) maximoX = X2g;
                if (X3g > maximoX) maximoX = X3g;
                if (X4g > maximoX) maximoX = X4g;

                if (Y1g > maximoY) maximoY = Y1g;
                if (Y2g > maximoY) maximoY = Y2g;
                if (Y3g > maximoY) maximoY = Y3g;
                if (Y4g > maximoY) maximoY = Y4g;

                if (Z1g > maximoZ) maximoZ = Z1g;
                if (Z2g > maximoZ) maximoZ = Z2g;
                if (Z3g > maximoZ) maximoZ = Z3g;
                if (Z4g > maximoZ) maximoZ = Z4g;

                poligonos.Add(new Poligono(X1g, Y1g, Z1g, X2g, Y2g, Z2g, X3g, Y3g, Z3g, X4g, Y4g,
Z4g));
            }

            //Luego normaliza los puntos X,Y,Z para que queden entre -0.5 y 0.5
            for (int cont = 0; cont < poligonos.Count; cont++) {
                poligonos[cont].coordenadas1.posX = (poligonos[cont].coordenadas1.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas1.posY = (poligonos[cont].coordenadas1.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas1.posZ = (poligonos[cont].coordenadas1.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas2.posX = (poligonos[cont].coordenadas2.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas2.posY = (poligonos[cont].coordenadas2.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas2.posZ = (poligonos[cont].coordenadas2.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas3.posX = (poligonos[cont].coordenadas3.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas3.posY = (poligonos[cont].coordenadas3.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas3.posZ = (poligonos[cont].coordenadas3.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas4.posX = (poligonos[cont].coordenadas4.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas4.posY = (poligonos[cont].coordenadas4.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas4.posZ = (poligonos[cont].coordenadas4.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;
            }
```

```
            //Gira y convierte los puntos espaciales en puntos proyectados al plano
            for (int cont = 0; cont < poligonos.Count; cont++) {
                poligonos[cont].GiroXYZ(AnguloX, AnguloY, AnguloZ);
                poligonos[cont].Convierte3Da2D(ZPersona);
            }

            //Constantes para cuadrar el gráfico en pantalla
            //Se obtuvieron probando los ángulos de giro X,Y,Z de 0 a 360 y ZPersona
            double XplanoMax = 0.87931543769177811;
            double XplanoMin = -0.87931543769177811;
            double YplanoMax = 0.87931539875237918;
            double YplanoMin = -0.87931539875237918;

            //Constantes de transformación
            double ConstanteX = (pXmax - pXmin) / (XplanoMax - XplanoMin);
            double ConstanteY = (pYmax - pYmin) / (YplanoMax - YplanoMin);

            //Cuadra los polígonos en pantalla
            for (int cont = 0; cont < poligonos.Count; cont++)
                poligonos[cont].CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin,
pYmin);

            //Ordena del polígono más alejado al más cercano, de esa manera los polígonos de adelante
            //son visibles y los de atrás son borrados.
            poligonos.Sort();
        }

        public double Ecuacion(double x) {
            return -2*x*x-3*x+5;
        }


        //Pinta el gráfico generado por la ecuación
        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics lienzo = e.Graphics;
            Pen lapiz = new Pen(Color.Black, 1);
            Brush relleno = new SolidBrush(Color.White);
            for (int cont = 0; cont < poligonos.Count; cont++)  poligonos[cont].Dibuja(lienzo, lapiz,
relleno);
        }
    }
}
```

Sólido de revolución animado

Una ecuación del tipo Y=F(X,t) donde t es el tiempo. Y similar al anterior, con esa ecuación al girarla en el eje X se obtiene el sólido de revolución. Por cada t se cambia el gráfico. Un control timer se encarga de cambiar t y luego evaluar toda la ecuación con ese valor de t en particular, generar el sólido de revolución y mostrarlo en pantalla.

09. 3D/11. SolidoRevolucionAnimado/Puntos.cs

```
using System;

namespace Graficos {
    internal class Puntos {
        //Coordenada espacial original
        public double posX, posY, posZ;

        //Coordenada espacial después de girar
        public double posXg, posYg, posZg;

        //Coordenada proyectada
        public double planoX, planoY;

        //Coordenada cuadrada en pantalla
        public int pantallaX, pantallaY;

        //Constructor
        public Puntos(double posX, double posY, double posZ) {
            this.posX = posX;
            this.posY = posY;
            this.posZ = posZ;
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
            double angXr = angX * Math.PI / 180;
            double angYr = angY * Math.PI / 180;
            double angZr = angZ * Math.PI / 180;

            double CosX = Math.Cos(angXr);
            double SinX = Math.Sin(angXr);
            double CosY = Math.Cos(angYr);
            double SinY = Math.Sin(angYr);
            double CosZ = Math.Cos(angZr);
            double SinZ = Math.Sin(angZr);

            //Matriz de Rotación
            //https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions
            double[,] Matriz = new double[3, 3] {
                { CosY * CosZ, -CosX * SinZ + SinX * SinY * CosZ, SinX * SinZ + CosX * SinY * CosZ},
                { CosY * SinZ, CosX * CosZ + SinX * SinY * SinZ, -SinX * CosZ + CosX * SinY * SinZ},
                {-SinY, SinX * CosY, CosX * CosY }
            };

            posXg = posX * Matriz[0, 0] + posY * Matriz[1, 0] + posZ * Matriz[2, 0];
            posYg = posX * Matriz[0, 1] + posY * Matriz[1, 1] + posZ * Matriz[2, 1];
            posZg = posX * Matriz[0, 2] + posY * Matriz[1, 2] + posZ * Matriz[2, 2];
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            planoX = posXg * ZPersona / (ZPersona - posZg);
            planoY = posYg * ZPersona / (ZPersona - posZg);
        }

        //Cuadra los puntos en pantalla
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            pantallaX = Convert.ToInt32(ConstanteX * (planoX - XplanoMin) + pXmin);
            pantallaY = Convert.ToInt32(ConstanteY * (planoY - YplanoMin) + pYmin);
        }
    }
}
```

09. 3D/11. SolidoRevolucionAnimado/Poligono.cs

```
using System;
using System.Drawing;

namespace Graficos {
```

```
    internal class Poligono : IComparable {
        //Un polígono son cuatro(4) coordenadas espaciales
        public Puntos coordenadas1, coordenadas2, coordenadas3, coordenadas4;
        private double centro;

        public Poligono(double X1, double Y1, double Z1, double X2, double Y2, double Z2, double X3,
double Y3, double Z3, double X4, double Y4, double Z4) {
            coordenadas1 = new Puntos(X1, Y1, Z1);
            coordenadas2 = new Puntos(X2, Y2, Z2);
            coordenadas3 = new Puntos(X3, Y3, Z3);
            coordenadas4 = new Puntos(X4, Y4, Z4);
        }

        //Gira en XYZ
        public void GiroXYZ(double angX, double angY, double angZ) {
            coordenadas1.GiroXYZ(angX, angY, angZ);
            coordenadas2.GiroXYZ(angX, angY, angZ);
            coordenadas3.GiroXYZ(angX, angY, angZ);
            coordenadas4.GiroXYZ(angX, angY, angZ);
            centro = (coordenadas1.posZg + coordenadas2.posZg + coordenadas3.posZg + coordenadas4.posZg) /
4;
        }

        //Convierte de 3D a 2D
        public void Convierte3Da2D(double ZPersona) {
            coordenadas1.Convierte3Da2D(ZPersona);
            coordenadas2.Convierte3Da2D(ZPersona);
            coordenadas3.Convierte3Da2D(ZPersona);
            coordenadas4.Convierte3Da2D(ZPersona);
        }

        //Cuadra en pantalla
        public void CuadraPantalla(double ConstanteX, double ConstanteY, double XplanoMin, double
YplanoMin, int pXmin, int pYmin) {
            coordenadas1.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas2.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas3.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
            coordenadas4.CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin, pYmin);
        }


        //Hace el gráfico del polígono
        public void Dibuja(Graphics lienzo, Pen lapiz, Brush relleno) {
            //Pone un color de fondo al polígono para borrar lo que hay detrás
            Point punto1 = new Point(coordenadas1.pantallaX, coordenadas1.pantallaY);
            Point punto2 = new Point(coordenadas2.pantallaX, coordenadas2.pantallaY);
            Point punto3 = new Point(coordenadas3.pantallaX, coordenadas3.pantallaY);
            Point punto4 = new Point(coordenadas4.pantallaX, coordenadas4.pantallaY);
            Point[] poligono = { punto1, punto2, punto3, punto4 };
            lienzo.FillPolygon(relleno, poligono);

            //Pinta el perímetro del polígono
            lienzo.DrawLine(lapiz, coordenadas1.pantallaX, coordenadas1.pantallaY, coordenadas2.pantallaX,
coordenadas2.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas2.pantallaX, coordenadas2.pantallaY, coordenadas3.pantallaX,
coordenadas3.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas3.pantallaX, coordenadas3.pantallaY, coordenadas4.pantallaX,
coordenadas4.pantallaY);
            lienzo.DrawLine(lapiz, coordenadas4.pantallaX, coordenadas4.pantallaY, coordenadas1.pantallaX,
coordenadas1.pantallaY);
        }

        public int CompareTo(object obj) {
            Poligono orderToCompare = obj as Poligono;
            if (orderToCompare.centro < centro) {
                return 1;
            }
            if (orderToCompare.centro > centro) {
                return -1;
            }

            // The orders are equivalent.
            //https://stackoverflow.com/questions/3309188/how-to-sort-a-listt-by-a-property-in-the-object
            return 0;
        }
    }
}
```

```
//Sólidos de revolución animados
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Para la variable temporal
        private double Tminimo, Tmaximo, Tincrementa, Tiempo;

        //Donde almacena los poligonos
        List<Poligono> poligonos;

        //Ángulos de giro
        double AnguloX, AnguloY, AnguloZ;
        int ZPersona;

        //Datos para la ecuación
        double minXreal, maxXreal;
        int numLineas;

        //Datos para la pantalla
        int pXmin, pYmin, pXmax, pYmax;

        public Form1() {
            InitializeComponent();
            poligonos = new List<Poligono>();

            //Valores por defecto a los ángulos
            AnguloX = (double)numGiroX.Value;
            AnguloY = (double)numGiroY.Value;
            AnguloZ = (double)numGiroZ.Value;
            ZPersona = 5;

            //Valores para la ecuación
            minXreal = -2;
            maxXreal = 2;
            numLineas = 30;

            //Inicia el tiempo
            Tminimo = 0.1;
            Tmaximo = 1;
            Tincrementa = 0.01;
            Tiempo = Tminimo;

            //Coordenadas de pantalla
            pXmin = 0;
            pYmin = 0;
            pXmax = 600;
            pYmax = 600;

            Logica();
        }

        private void timerAnimar_Tick(object sender, EventArgs e) {
            Tiempo += Tincrementa;
            if (Tiempo <= Tminimo || Tiempo >= Tmaximo) Tincrementa = -Tincrementa;
            Logica();
            Refresh();
        }

        private void numGiroX_ValueChanged(object sender, System.EventArgs e) {
            AnguloX = Convert.ToDouble(numGiroX.Value);
            Logica();
            Refresh();
        }

        private void numGiroY_ValueChanged(object sender, EventArgs e) {
            AnguloY = Convert.ToDouble(numGiroY.Value);
            Logica();
            Refresh();
        }

        private void numGiroZ_ValueChanged(object sender, EventArgs e) {
            AnguloZ = Convert.ToDouble(numGiroZ.Value);
```

```
        Logica();
        Refresh();
    }

    public void Logica() {
        //Calcula los puntos de la ecuación y así formar los polígonos
        poligonos.Clear();
        double incrAngulo = 360 / numLineas;
        double incrXreal = (maxXreal - minXreal) / numLineas;

        //Mínimos y máximos para normalizar
        double minimoX = double.MaxValue;
        double minimoY = double.MaxValue;
        double minimoZ = double.MaxValue;
        double maximoX = double.MinValue;
        double maximoY = double.MinValue;
        double maximoZ = double.MinValue;

        for (double angulo = 0; angulo < 360; angulo += incrAngulo)
            for (double valXreal = minXreal; valXreal <= maxXreal; valXreal += incrXreal) {

                //Primer punto
                double X1 = valXreal;
                double Y1 = Ecuacion(X1, Tiempo);
                if (double.IsNaN(Y1) || double.IsInfinity(Y1)) Y1 = 0;

                //Aplica giro en el eje X
                double X1g = X1;
                double Y1g = Y1 * Math.Cos(angulo * Math.PI / 180);
                double Z1g = Y1 * -Math.Sin(angulo * Math.PI / 180);

                //Segundo punto
                double X2 = valXreal + incrXreal;
                double Y2 = Ecuacion(X2, Tiempo);
                if (double.IsNaN(Y2) || double.IsInfinity(Y2)) Y2 = 0;

                //Aplica giro en el eje X
                double X2g = X2;
                double Y2g = Y2 * Math.Cos(angulo * Math.PI / 180);
                double Z2g = Y2 * -Math.Sin(angulo * Math.PI / 180);

                //Tercer punto
                double X3g = X2;
                double Y3g = Y2 * Math.Cos((angulo + incrAngulo) * Math.PI / 180);
                double Z3g = Y2 * -Math.Sin((angulo + incrAngulo) * Math.PI / 180);

                //Cuarto punto
                double X4g = X1;
                double Y4g = Y1 * Math.Cos((angulo + incrAngulo) * Math.PI / 180);
                double Z4g = Y1 * -Math.Sin((angulo + incrAngulo) * Math.PI / 180);

                //Cálculo de los extremos para normalizar
                if (X1g < minimoX) minimoX = X1g;
                if (X2g < minimoX) minimoX = X2g;
                if (X3g < minimoX) minimoX = X3g;
                if (X4g < minimoX) minimoX = X4g;

                if (Y1g < minimoY) minimoY = Y1g;
                if (Y2g < minimoY) minimoY = Y2g;
                if (Y3g < minimoY) minimoY = Y3g;
                if (Y4g < minimoY) minimoY = Y4g;

                if (Z1g < minimoZ) minimoZ = Z1g;
                if (Z2g < minimoZ) minimoZ = Z2g;
                if (Z3g < minimoZ) minimoZ = Z3g;
                if (Z4g < minimoZ) minimoZ = Z4g;

                if (X1g > maximoX) maximoX = X1g;
                if (X2g > maximoX) maximoX = X2g;
                if (X3g > maximoX) maximoX = X3g;
                if (X4g > maximoX) maximoX = X4g;

                if (Y1g > maximoY) maximoY = Y1g;
                if (Y2g > maximoY) maximoY = Y2g;
                if (Y3g > maximoY) maximoY = Y3g;
                if (Y4g > maximoY) maximoY = Y4g;

                if (Z1g > maximoZ) maximoZ = Z1g;
                if (Z2g > maximoZ) maximoZ = Z2g;
```

```
                    if (Z3g > maximoZ) maximoZ = Z3g;
                    if (Z4g > maximoZ) maximoZ = Z4g;

                    poligonos.Add(new Poligono(X1g, Y1g, Z1g, X2g, Y2g, Z2g, X3g, Y3g, Z3g, X4g, Y4g,
Z4g));
                }

            //Luego normaliza los puntos X,Y,Z para que queden entre -0.5 y 0.5
            for (int cont = 0; cont < poligonos.Count; cont++) {
                poligonos[cont].coordenadas1.posX = (poligonos[cont].coordenadas1.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas1.posY = (poligonos[cont].coordenadas1.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas1.posZ = (poligonos[cont].coordenadas1.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas2.posX = (poligonos[cont].coordenadas2.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas2.posY = (poligonos[cont].coordenadas2.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas2.posZ = (poligonos[cont].coordenadas2.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas3.posX = (poligonos[cont].coordenadas3.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas3.posY = (poligonos[cont].coordenadas3.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas3.posZ = (poligonos[cont].coordenadas3.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;

                poligonos[cont].coordenadas4.posX = (poligonos[cont].coordenadas4.posX - minimoX) /
(maximoX - minimoX) - 0.5;
                poligonos[cont].coordenadas4.posY = (poligonos[cont].coordenadas4.posY - minimoY) /
(maximoY - minimoY) - 0.5;
                poligonos[cont].coordenadas4.posZ = (poligonos[cont].coordenadas4.posZ - minimoZ) /
(maximoZ - minimoZ) - 0.5;
            }

            //Gira y convierte los puntos espaciales en puntos proyectados al plano
            for (int cont = 0; cont < poligonos.Count; cont++) {
                poligonos[cont].GiroXYZ(AnguloX, AnguloY, AnguloZ);
                poligonos[cont].Convierte3Da2D(ZPersona);
            }

            //Constantes para cuadrar el gráfico en pantalla
            //Se obtuvieron probando los ángulos de giro X,Y,Z de 0 a 360 y ZPersona
            double XplanoMax = 0.87931543769177811;
            double XplanoMin = -0.87931543769177811;
            double YplanoMax = 0.87931539875237918;
            double YplanoMin = -0.87931539875237918;

            //Constantes de transformación
            double ConstanteX = (pXmax - pXmin) / (XplanoMax - XplanoMin);
            double ConstanteY = (pYmax - pYmin) / (YplanoMax - YplanoMin);

            //Cuadra los polígonos en pantalla
            for (int cont = 0; cont < poligonos.Count; cont++)
                poligonos[cont].CuadraPantalla(ConstanteX, ConstanteY, XplanoMin, YplanoMin, pXmin,
pYmin);

            //Ordena del polígono más alejado al más cercano, de esa manera los polígonos de adelante
            //son visibles y los de atrás son borrados.
            poligonos.Sort();
        }

        public double Ecuacion(double x, double t) {
            return -2 * x * x * t - 3 * x * t + 5;
        }


        //Pinta el gráfico generado por la ecuación
        private void Form1_Paint(object sender, PaintEventArgs e) {
            Graphics lienzo = e.Graphics;
            Pen lapiz = new Pen(Color.Black, 1);
            Brush relleno = new SolidBrush(Color.White);
            for (int cont = 0; cont < poligonos.Count; cont++) poligonos[cont].Dibuja(lienzo, lapiz,
relleno);
        }
    }
```

```
}
```

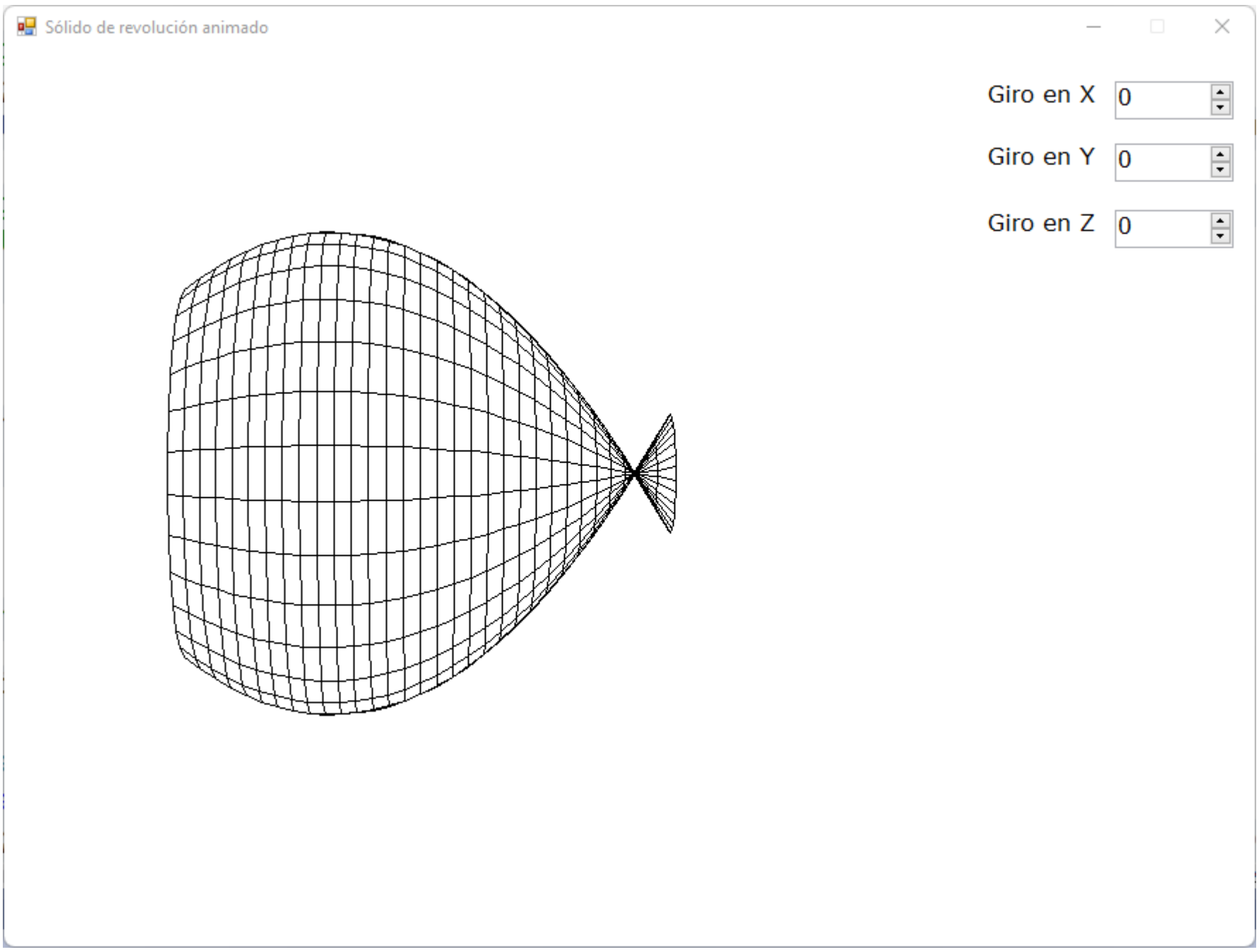


Ilustración 98: Un "frame" de un sólido de revolución animado

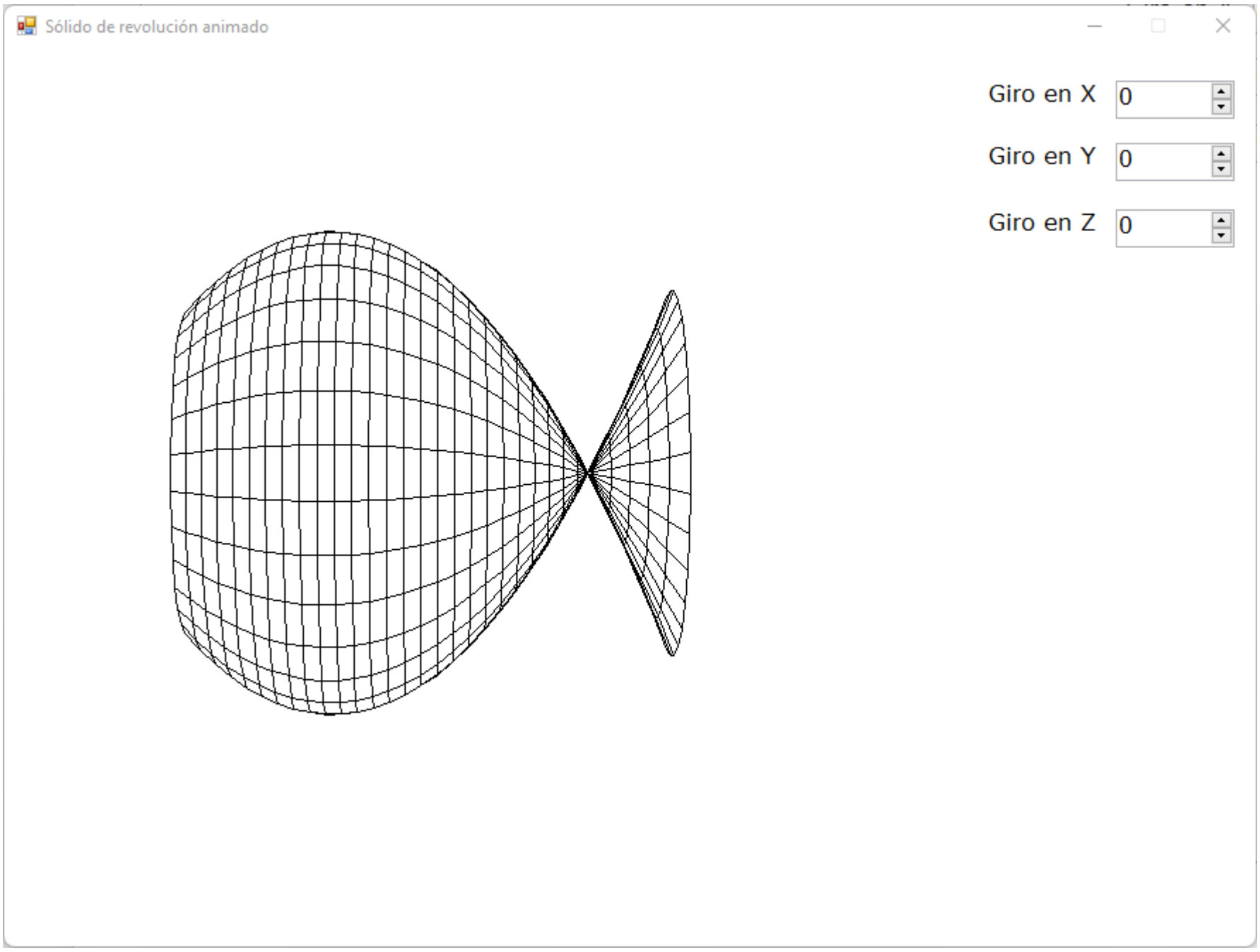


Ilustración 99:: Un "frame" de un sólido de revolución animado

Manejo de imágenes

Dada una imagen obtenida, por ejemplo, de tomar una fotografía con una cámara digital, se muestra como cargarla y mostrarla dentro de una aplicación de escritorio en C# junto con algunas manipulaciones (escala de grises, rotación y zoom).

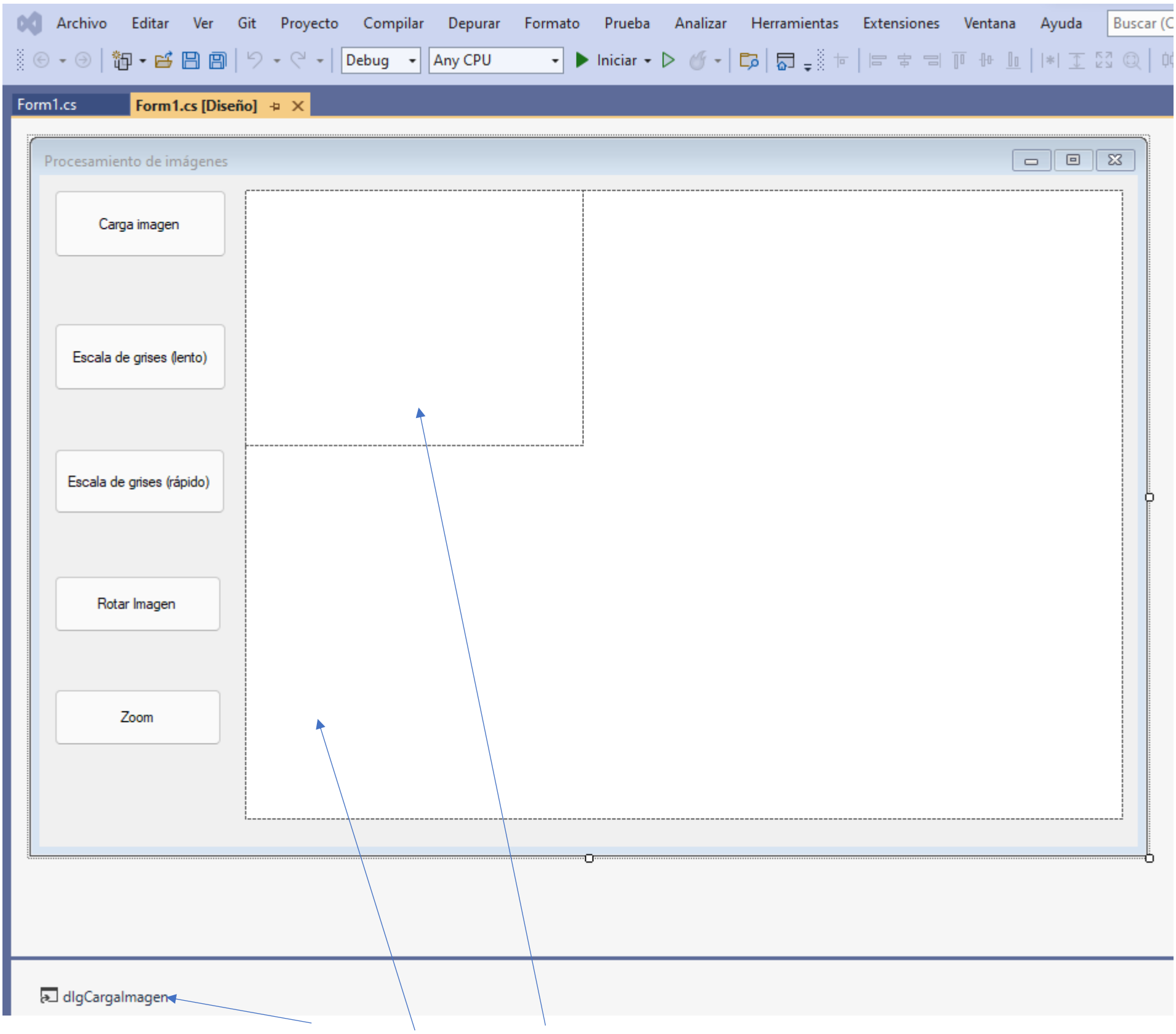


Ilustración 100: Uso de un cuadro de diálogo para abrir archivos, un panel y un control Picturebox

En el ejemplo, se usará un cuadro de diálogo (OpenFileDialog) para abrir archivos de imagen (*.jpg, *.png, *.bmp), un control panel (con autoscroll en true) y un picturebox dentro del control panel.

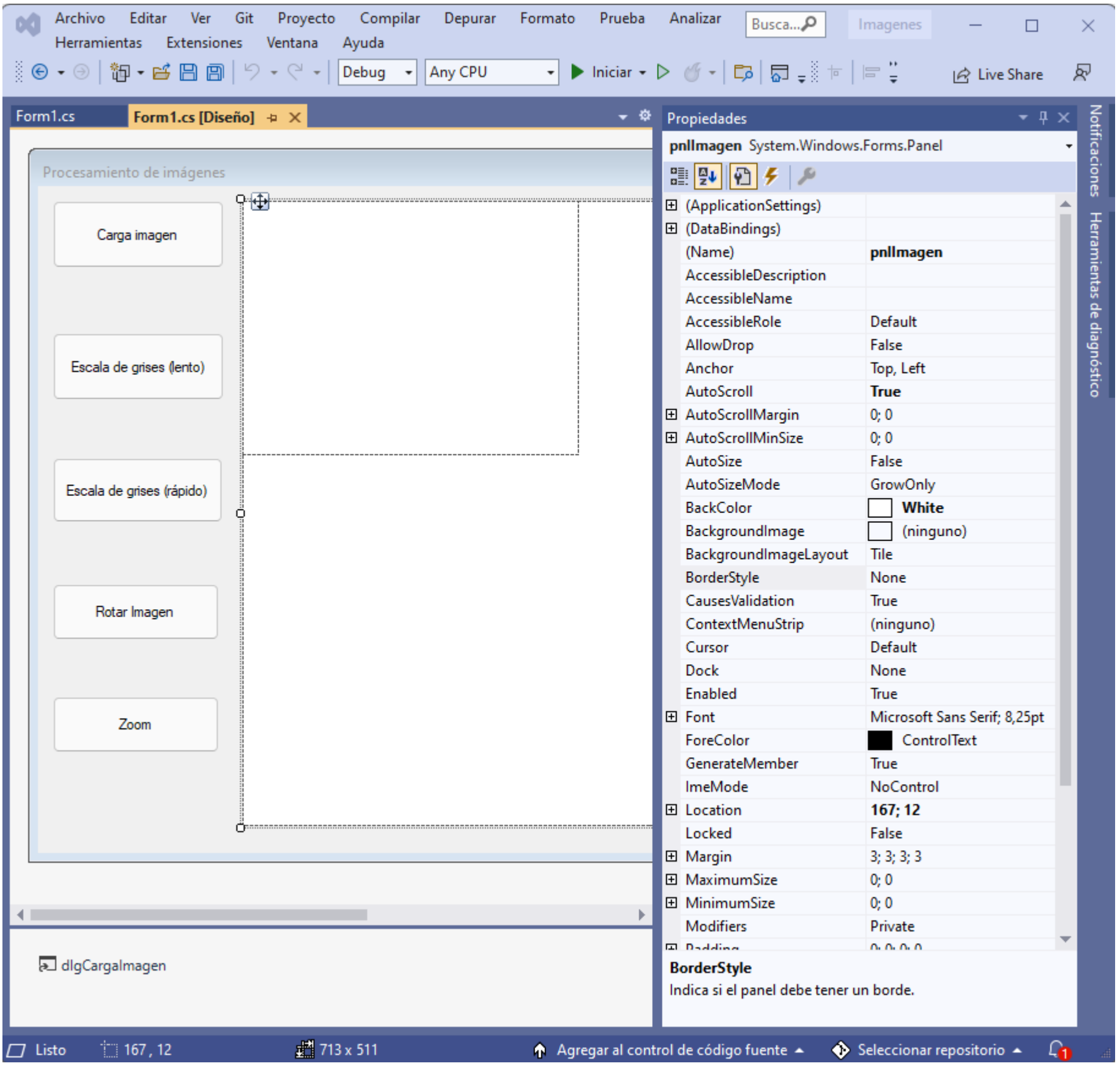


Ilustración 101: Propiedades del panel

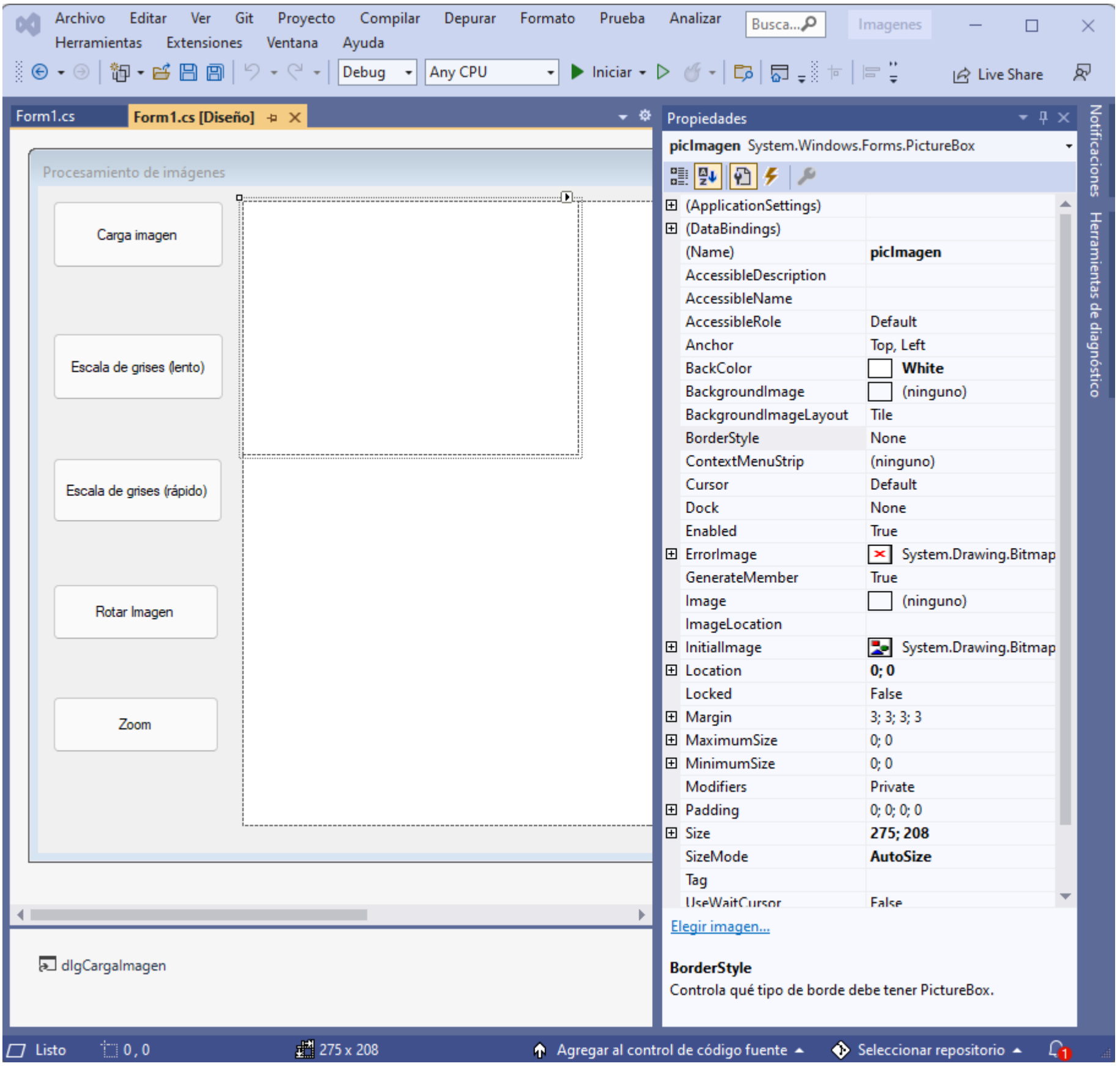


Ilustración 102: Propiedades del PictureBox

Este es el código fuente:

Imagen/Imágenes.zip/Form1.cs

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
using System.Windows.Forms;

namespace Imagenes {
    public partial class Form1 : Form {
        public Form1() {
            InitializeComponent();

            //Que tipo de archivos mostrará el cuadro de diálogo
            dlgCargaImagen.Filter = "Archivos de imagen (*.BMP; *.JPG; *.GIF)| *.BMP; *.JPG; *.GIF";
        }

        //Carga la imagen
        private void btnCarga_Click(object sender, EventArgs e) {
            //Llama a la ventana de diálogo
            if (dlgCargaImagen.ShowDialog() == DialogResult.OK) {

                //Si el usuario ha seleccionado correctamente un archivo de imagen
                picImagen.Image = new Bitmap(dlgCargaImagen.FileName);

                //Activa los botones
```

```
            btnGrises.Enabled = true;
            btnGris2.Enabled = true;
            btnRotar.Enabled = true;
            btnZoom.Enabled = true;
        }
    }

    //Técnica clásica de manipulación de imágenes. Sin embargo es muy lenta.
    private void btnGrises_Click(object sender, EventArgs e) {
        Bitmap Original = new Bitmap(picImagen.Image);
        Bitmap Gris = new Bitmap(Original.Width, Original.Height);

        for (int posX = 0; posX < Original.Width; posX++) {
            for (int posY = 0; posY < Original.Height; posY++) {
                Color antes = Original.GetPixel(posX, posY);
                int EscalaGris = (int)((antes.R * 0.3) + (antes.G * 0.59) + (antes.B * 0.11));
                Color despues = Color.FromArgb(antes.A, EscalaGris, EscalaGris, EscalaGris);
                Gris.SetPixel(posX, posY, despues);
            }
        }

        picImagen.Image = Gris;
    }

    private void btnGris2_Click(object sender, EventArgs e) {
        picImagen.Image = MakeGrayscale3(new Bitmap(picImagen.Image));
    }
        }

        //Técnica rápida de manipulación de imágenes
        //Tomado de:
https://web.archive.org/web/20130111215043/http://www.switchonthecode.com/tutorials/csharp-tutorial-
convert-a-color-image-to-grayscale
        public static Bitmap MakeGrayscale3(Bitmap original) {
            //create a blank bitmap the same size as original
            Bitmap newBitmap = new Bitmap(original.Width, original.Height);

            //get a graphics object from the new image
            using (Graphics g = Graphics.FromImage(newBitmap)) {

                //create the grayscale ColorMatrix
                ColorMatrix colorMatrix = new ColorMatrix(
                   new float[][]
                   {
                     new float[] {.3f, .3f, .3f, 0, 0},
                     new float[] {.59f, .59f, .59f, 0, 0},
                     new float[] {.11f, .11f, .11f, 0, 0},
                     new float[] {0, 0, 0, 1, 0},
                     new float[] {0, 0, 0, 0, 1}
                   });

                //create some image attributes
                using (ImageAttributes attributes = new ImageAttributes()) {

                    //set the color matrix attribute
                    attributes.SetColorMatrix(colorMatrix);

                    //draw the original image on the new image
                    //using the grayscale color matrix
                    g.DrawImage(original, new Rectangle(0, 0, original.Width, original.Height),
                                0, 0, original.Width, original.Height, GraphicsUnit.Pixel, attributes);
                }
            }
            return newBitmap;
    }

    //Girar una imagen
    private void btnRotar_Click(object sender, EventArgs e) {
        picImagen.Image = RotateImage(new Bitmap(picImagen.Image), 45);
    }

    //Tomado de: https://stackoverflow.com/questions/2163829/how-do-i-rotate-a-picture-in-winforms
    public static Image RotateImage(Image img, float rotationAngle) {
        //create an empty Bitmap image
        Bitmap bmp = new Bitmap(img.Width, img.Height);

        //turn the Bitmap into a Graphics object
        Graphics gfx = Graphics.FromImage(bmp);

        //now we set the rotation point to the center of our image
```

```
            gfx.TranslateTransform((float)bmp.Width / 2, (float)bmp.Height / 2);

            //now rotate the image
            gfx.RotateTransform(rotationAngle);

            gfx.TranslateTransform(-(float)bmp.Width / 2, -(float)bmp.Height / 2);

            //set the InterpolationMode to HighQualityBicubic so to ensure a high
            //quality image once it is transformed to the specified size
            gfx.InterpolationMode = InterpolationMode.HighQualityBicubic;

            //now draw our new image onto the graphics object
            gfx.DrawImage(img, new Point(0, 0));

            //dispose of our Graphics object
            gfx.Dispose();

            //return the image
            return bmp;
        }

        //Disminuir el tamaño de una imagen
        private void btnZoom_Click(object sender, EventArgs e) {
            picImagen.Image = HaceZoom(new Bitmap(picImagen.Image), 0.5);
        }

        //Tomado de: https://stackoverflow.com/questions/10915958/how-to-zoom-an-image-inout-in-c
        public static Image HaceZoom(Image originalBitmap, double zoomFactor) {
            Size newSize = new Size((int)(originalBitmap.Width * zoomFactor), (int)(originalBitmap.Height
* zoomFactor));
            Bitmap bmp = new Bitmap(originalBitmap, newSize);
            return bmp;
        }


        /* Lecturas:
         * https://stackoverflow.com/questions/24701703/c-sharp-faster-alternatives-to-setpixel-and-
getpixel-for-bitmaps-for-windows-f
         * http://csharpexamples.com/fast-image-processing-c/
         * https://www.codeproject.com/Tips/240428/Work-with-Bitmaps-Faster-in-Csharp-3
         * https://softwarebydefault.com/
         * https://www.instructables.com/c-Edge-Detection/
         * https://www.youtube.com/watch?v=EPKyazYi4MI&list=PLM-p96nOrGcabqC2GvLLIL_rxx3q89JI1&index=1
         * http://coding-experiments.blogspot.com/search/label/edge%20detection
         */
    }
}
```

Ilustración 103: Inicio del programa

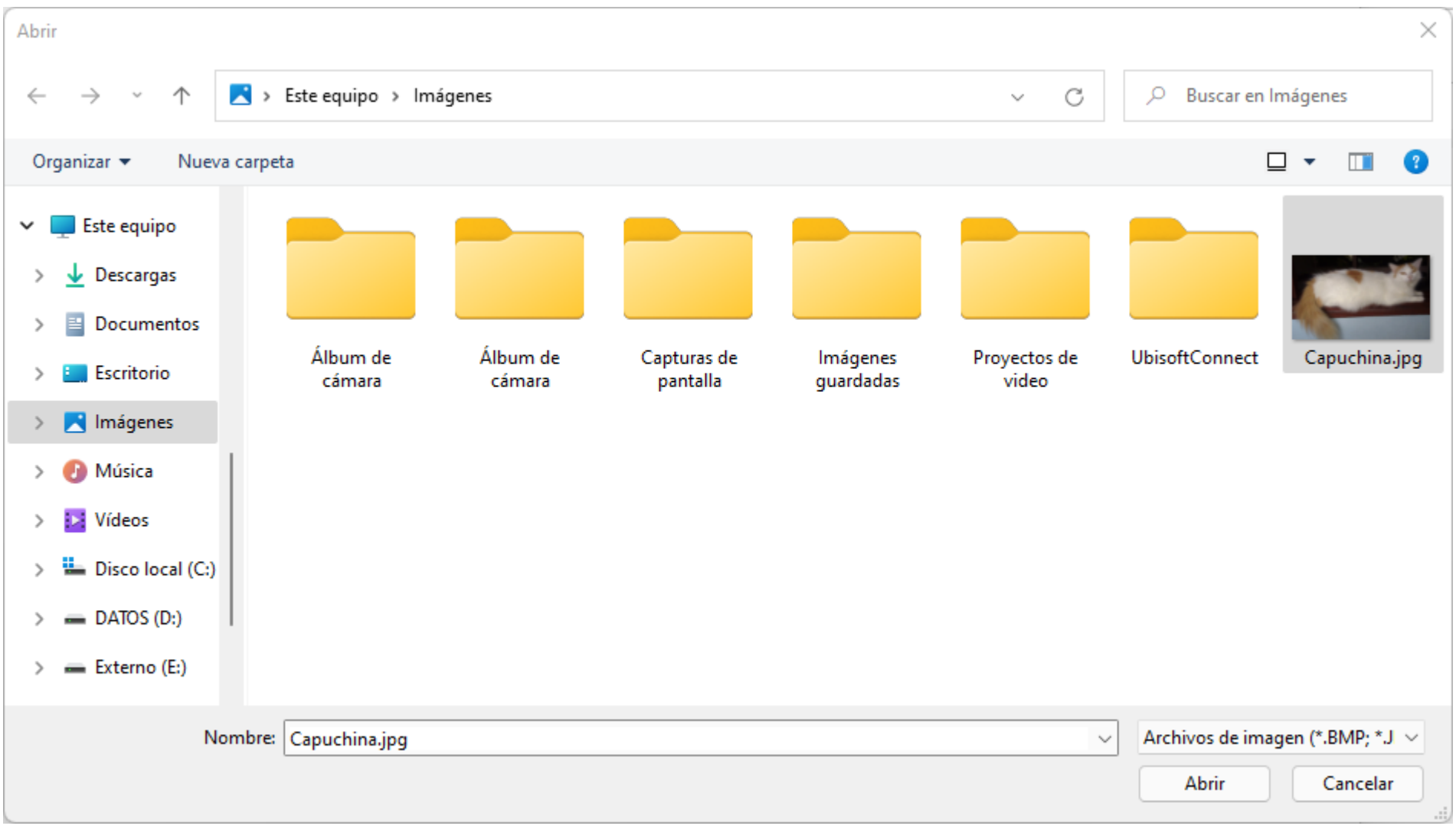


Ilustración 104: Caja de diálogo para abrir archivo de imagen

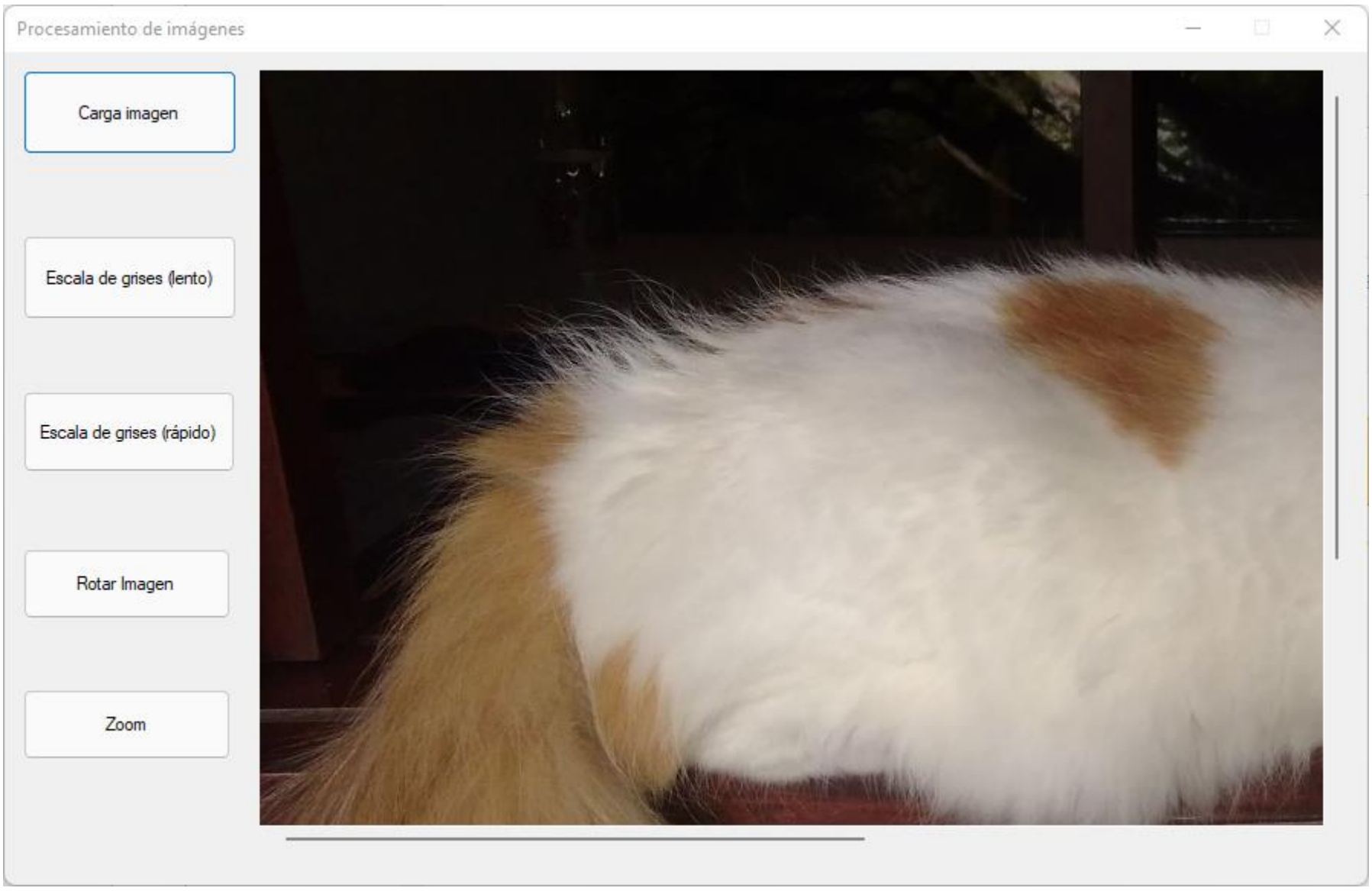


Ilustración 105: Imagen cargada

Ilustración 106: Haciendo uso del scroll vertical y horizontal

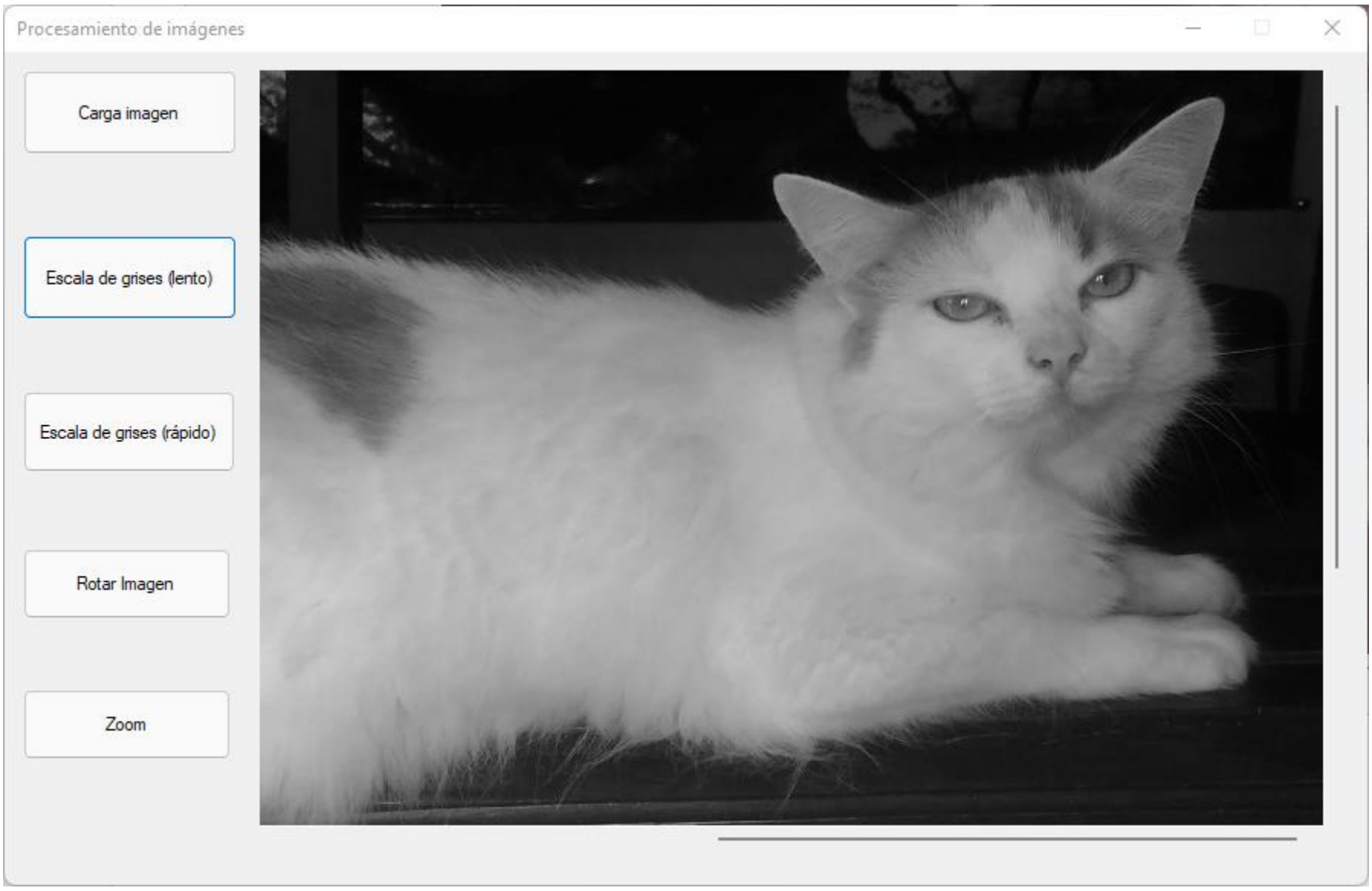


Ilustración 107: Filtro para convertir la imagen en una escala de grises

Ilustración 108: Girar una imagen

Nota: Al girar una imagen, se pierden zonas de la imagen original y el motivo es que se mantiene el tamaño del rectángulo original. Luego al girar se deben hacer cálculos previos de como modificar ese rectángulo para que reciba sin pérdida la imagen girada.

Ilustración 109: Disminuir el tamaño de una imagen

Videojuegos

Con C# es fácil hacer un videojuego, porque tenemos los controles Timer, la captura de eventos del teclado y el ratón, y un buen refresco de las ventanas. No es para juegos con alta carga gráfica, pero si suficiente para entender los conceptos.

Depredador – Presa

10. Videojuegos/01. Control.zip/Form1.cs

```
//Videojuego: El depredador (un cuadro rojo) controlado por el jugador con las teclas de flecha, busca la presa (un cuadro azul) evadiendo obstáculos (cuadros negros)
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Constantes para calificar cada celda del tablero
        private const int ESCAMINO = 0; //Celdas que representan caminos por donde puede transitar el depredador
        private const int ESOBSTACULO = 1; //Celdas que representan obstáculos por donde NO puede transitar el depredador
        private const int ESDEPREDADOR = 2; //Celda donde se dibuja la posición del depredador
        private const int ESPRESA = 3; //Celda donde se dibuja la posición de la presa
        private const int FINALIZA = 4; //Cuando el depredador cae sobre la presa

        int[,] Tablero; //Dónde ocurre realmente la acción, las coordenadas serán siempre [fila, columna]
        int DepredadorFila, DepredadorColumna; //Coordenadas del depredador
        int DepredaFila, DepredaColumna; //Desplazamiento del depredador
        int PresaFila, PresaColumna; //Coordenadas de la presa
        Random azar; //Único generador de números aleatorios.

        public Form1() {
            InitializeComponent();
            azar = new Random();
            IniciarParametros(); //Inicia el tablero, los obstáculos, el depredador y la presa
        }

        private void mnuReiniciar_Click(object sender, EventArgs e) {
            IniciarParametros();
            Refresh();
        }

        public void IniciarParametros() {
            //Inicializa el tablero.
            Tablero = new int[30, 30]; //Tablero[filas, columnas]

            //Habrán obstáculos en el 20% del tablero
            int Obstaculos = Tablero.GetLength(0) * Tablero.GetLength(1) * 20 / 100;
            for (int cont = 1; cont <= Obstaculos; cont++) {
                int obstaculoFila, obstaculoColumna;
                do {
                    obstaculoFila = azar.Next(0, Tablero.GetLength(0));
                    obstaculoColumna = azar.Next(0, Tablero.GetLength(1));
                } while (Tablero[obstaculoFila, obstaculoColumna] != ESCAMINO);
                Tablero[obstaculoFila, obstaculoColumna] = ESOBSTACULO;
            }

            //Inicializa la posición del depredador que no sea encima de un obstáculo
            do {
                DepredadorFila = azar.Next(0, Tablero.GetLength(0));
                DepredadorColumna = azar.Next(0, Tablero.GetLength(1));
            } while (Tablero[DepredadorFila, DepredadorColumna] == ESOBSTACULO);
            Tablero[DepredadorFila, DepredadorColumna] = ESDEPREDADOR;

            //Inicializa la posición de la presa que no sea encima de un obstáculo o del depredador
            do {
                PresaFila = azar.Next(0, Tablero.GetLength(0));
                PresaColumna = azar.Next(0, Tablero.GetLength(1));
            } while (Tablero[PresaFila, PresaColumna] == ESOBSTACULO || Tablero[PresaFila, PresaColumna] == ESDEPREDADOR);
            Tablero[PresaFila, PresaColumna] = ESPRESA;
        }

        private void Form1_KeyDown(object sender, KeyEventArgs e) {
            DepredaFila = 0;
            DepredaColumna = 0;
```

```
            //Dependiendo de que tecla de flecha presiona el jugador
            if (e.KeyCode == Keys.Up) DepredaColumna = -1;
            if (e.KeyCode == Keys.Down) DepredaColumna = 1;
            if (e.KeyCode == Keys.Left) DepredaFila = -1;
            if (e.KeyCode == Keys.Right) DepredaFila = 1;

            Logica();
            Refresh();
        }

        public void Logica() {
            //Chequea que no se salga del tablero
            if (DepredadorFila + DepredaFila < 0 || DepredadorColumna + DepredaColumna < 0 ||
DepredadorFila + DepredaFila >= Tablero.GetLength(0) || DepredadorColumna + DepredaColumna >=
Tablero.GetLength(1)) return;

            //Valida si puede desplazarse a esa nueva casilla
            if (Tablero[DepredadorFila + DepredaFila, DepredadorColumna + DepredaColumna] == ESCAMINO) {
                Tablero[DepredadorFila, DepredadorColumna] = ESCAMINO; //Borra la antigua posición
                DepredadorFila += DepredaFila; //Desplaza en Fila
                DepredadorColumna += DepredaColumna; //Desplaza en Columna
                Tablero[DepredadorFila, DepredadorColumna] = ESDEPREDADOR; //Dibuja la nueva posición
            }
            //Valida si terminó el juego cuando el depredador cae en la casilla de la presa
            else if (Tablero[DepredadorFila + DepredaFila, DepredadorColumna + DepredaColumna] == ESPRESA)
{
                Tablero[DepredadorFila, DepredadorColumna] = ESCAMINO; //Borra la antigua posición
                Tablero[DepredadorFila + DepredaFila, DepredadorColumna + DepredaColumna] = FINALIZA;
//Dibuja la captura
                Refresh();
                MessageBox.Show("El depredador alcanzó a la presa", "Depredador - Presa",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                IniciarParametros();
                Refresh();
            }
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Tamaño de cada celda
            int tamanoFila = 500 / Tablero.GetLength(0);
            int tamanoColumna = 500 / Tablero.GetLength(1);
            int desplaza = 50;

            //Dibuja el arreglo bidimensional
            for (int Fila = 0; Fila < Tablero.GetLength(0); Fila++) {
                for (int Columna = 0; Columna < Tablero.GetLength(1); Columna++) {
                    int UbicaFila = Fila * tamanoFila + desplaza;
                    int UbicaColumna = Columna * tamanoColumna + desplaza;
                    switch (Tablero[Fila, Columna]) {
                        case ESCAMINO: e.Graphics.DrawRectangle(Pens.Blue, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                        case ESOBSTACULO: e.Graphics.FillRectangle(Brushes.Black, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                        case ESDEPREDADOR: e.Graphics.FillRectangle(Brushes.Red, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                        case ESPRESA: e.Graphics.FillRectangle(Brushes.Blue, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                        case FINALIZA: e.Graphics.FillRectangle(Brushes.Brown, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                    }
                }
            }
        }
    }
}
```

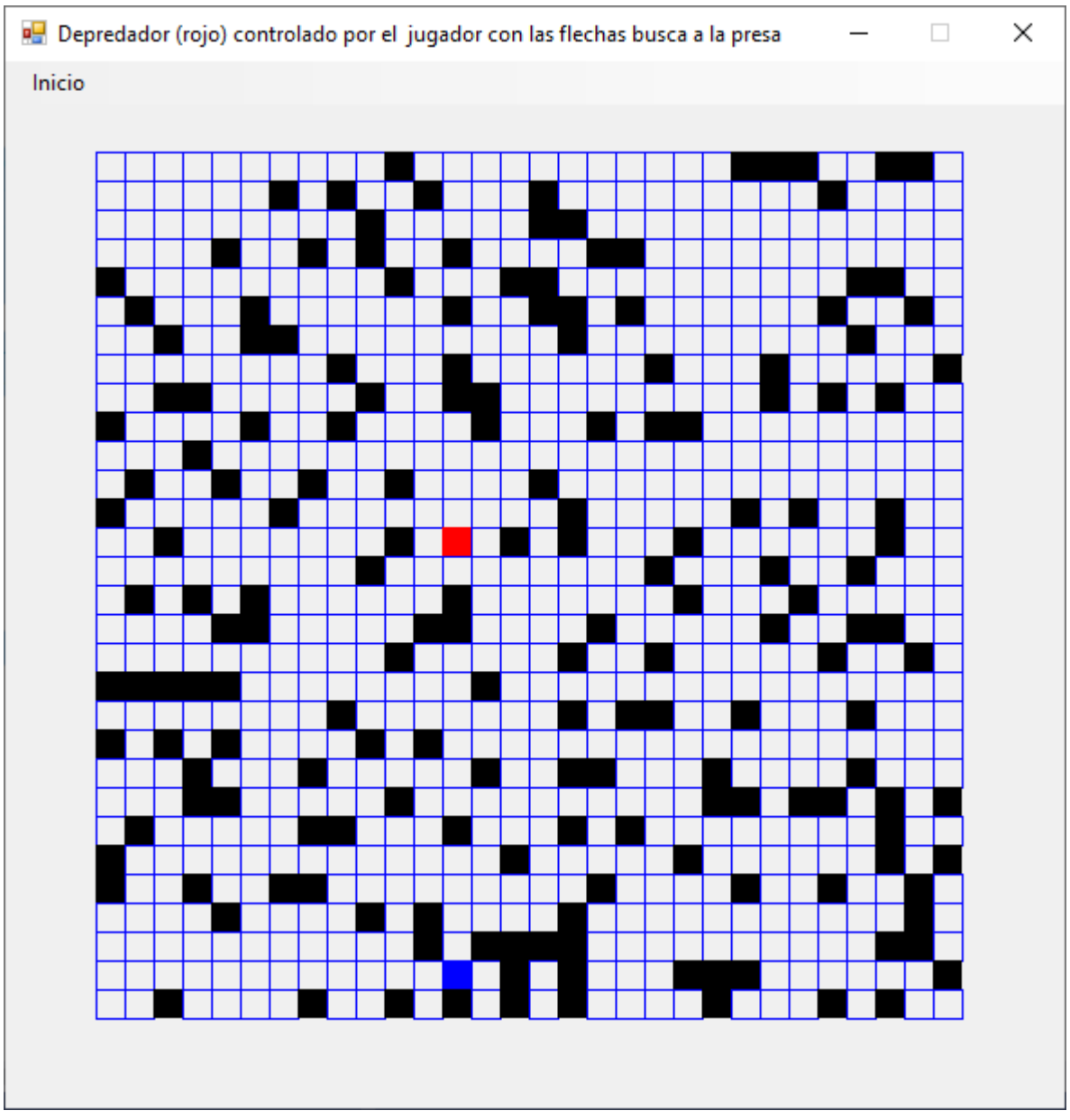


Ilustración 110: Depredador en rojo, presa en azul

10. Videojuegos/02. ControlAnima.zip/Form1.cs

```
//Videojuego: El depredador (un cuadro rojo) controlado por el jugador con las teclas de flecha, busca la presa (un cuadro azul) evadiendo obstáculos (cuadros negros).
//El depredador es controlado por el control Timer (se mueve solo dependiendo de la dirección seleccionada por el jugador)
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Constantes para calificar cada celda del tablero
        private const int ESCAMINO = 0; //Celdas que representan caminos por donde puede transitar el depredador
        private const int ESOBSTACULO = 1; //Celdas que representan obstáculos por donde NO puede transitar el depredador
        private const int ESDEPREDADOR = 2; //Celda donde se dibuja la posición del depredador
        private const int ESPRESA = 3; //Celda donde se dibuja la posición de la presa
        private const int FINALIZA = 4; //Cuando el depredador cae sobre la presa

        int[,] Tablero; //Dónde ocurre realmente la acción, las coordenadas serán siempre [fila, columna]
        int DepredadorFila, DepredadorColumna; //Coordenadas del depredador
        int DepredaFila, DepredaColumna; //Desplazamiento del depredador
        int PresaFila, PresaColumna; //Coordenadas de la presa
        Random azar; //Único generador de números aleatorios.

        public Form1() {
            InitializeComponent();
            azar = new Random();
            IniciarParametros();
        }

        private void mnuReiniciar_Click(object sender, EventArgs e) {
            IniciarParametros();
            Refresh();
        }

        public void IniciarParametros() {
            //Inicializa el tablero.
            Tablero = new int[30, 30];

            //Habrán obstáculos en el 20% del tablero
            int Obstaculos = Tablero.GetLength(0) * Tablero.GetLength(1) * 20 / 100;
            for (int cont = 1; cont <= Obstaculos; cont++) {
                int obstaculoFila, obstaculoColumna;
                do {
                    obstaculoFila = azar.Next(0, Tablero.GetLength(0));
                    obstaculoColumna = azar.Next(0, Tablero.GetLength(1));
                } while (Tablero[obstaculoFila, obstaculoColumna] != ESCAMINO);
                Tablero[obstaculoFila, obstaculoColumna] = ESOBSTACULO;
            }

            //Inicializa la posición del depredador que no sea encima de un obstáculo
            do {
                DepredadorFila = azar.Next(0, Tablero.GetLength(0));
                DepredadorColumna = azar.Next(0, Tablero.GetLength(1));
            } while (Tablero[DepredadorFila, DepredadorColumna] == ESOBSTACULO);
            Tablero[DepredadorFila, DepredadorColumna] = ESDEPREDADOR;

            //Inicializa la posición de la presa que no sea encima de un obstáculo o del depredador
            do {
                PresaFila = azar.Next(0, Tablero.GetLength(0));
                PresaColumna = azar.Next(0, Tablero.GetLength(1));
            } while (Tablero[PresaFila, PresaColumna] == ESOBSTACULO || Tablero[PresaFila, PresaColumna] == ESDEPREDADOR);
            Tablero[PresaFila, PresaColumna] = ESPRESA;
        }

        private void timerAnimar_Tick(object sender, EventArgs e) {
            Logica();
            Refresh();
        }

        private void Form1_KeyDown(object sender, KeyEventArgs e) {
            DepredaFila = 0;
            DepredaColumna = 0;
```

```
            //Dependiendo de que tecla de flecha presiona el jugador
            if (e.KeyCode == Keys.Up) DepredaColumna = -1;
            if (e.KeyCode == Keys.Down) DepredaColumna = 1;
            if (e.KeyCode == Keys.Left) DepredaFila = -1;
            if (e.KeyCode == Keys.Right) DepredaFila = 1;
        }

        public void Logica() {
            //Chequea que no se salga del tablero
            if (DepredadorFila + DepredaFila < 0 || DepredadorColumna + DepredaColumna < 0 ||
DepredadorFila + DepredaFila >= Tablero.GetLength(0) || DepredadorColumna + DepredaColumna >=
Tablero.GetLength(1)) return;

            //Valida si puede desplazarse a esa nueva casilla
            if (Tablero[DepredadorFila + DepredaFila, DepredadorColumna + DepredaColumna] == ESCAMINO) {
                Tablero[DepredadorFila, DepredadorColumna] = ESCAMINO; //Borra la antigua posición
                DepredadorFila += DepredaFila; //Desplaza en Fila
                DepredadorColumna += DepredaColumna; //Desplaza en Columna
                Tablero[DepredadorFila, DepredadorColumna] = ESDEPREDADOR; //Dibuja la nueva posición
            }
            //Valida si terminó el juego cuando el depredador cae en la casilla de la presa
            else if (Tablero[DepredadorFila + DepredaFila, DepredadorColumna + DepredaColumna] == ESPRESA)
{
                Tablero[DepredadorFila, DepredadorColumna] = ESCAMINO; //Borra la antigua posición
                Tablero[DepredadorFila + DepredaFila, DepredadorColumna + DepredaColumna] = FINALIZA;
//Dibuja la captura
                Refresh();
                MessageBox.Show("El depredador alcanzó a la presa", "Depredador - Presa",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                IniciarParametros();
                Refresh();
            }
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Tamaño de cada celda
            int tamanoFila = 500 / Tablero.GetLength(0);
            int tamanoColumna = 500 / Tablero.GetLength(1);
            int desplaza = 50;

            //Dibuja el arreglo bidimensional
            for (int Fila = 0; Fila < Tablero.GetLength(0); Fila++) {
                for (int Columna = 0; Columna < Tablero.GetLength(1); Columna++) {
                    int UbicaFila = Fila * tamanoFila + desplaza;
                    int UbicaColumna = Columna * tamanoColumna + desplaza;
                    switch (Tablero[Fila, Columna]) {
                        case ESCAMINO: e.Graphics.DrawRectangle(Pens.Blue, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                        case ESOBSTACULO: e.Graphics.FillRectangle(Brushes.Black, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                        case ESDEPREDADOR: e.Graphics.FillRectangle(Brushes.Red, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                        case ESPRESA: e.Graphics.FillRectangle(Brushes.Blue, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                        case FINALIZA: e.Graphics.FillRectangle(Brushes.Brown, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                    }
                }
            }
        }
    }
}
```

Evade – Piedras

10. Videojuegos/03. Evade.zip/Form1.cs

```
//Videojuego: El jugador debe evitar que le caigan las piedras encima
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Constantes para calificar cada celda del tablero
        private const int VACIO = 0;
        private const int PIEDRA = 1;
        private const int JUGADOR = 2;

        int[,] Tablero; //Dónde ocurre realmente la acción
        int PosJugador, MovimientoJugador; //Columna donde está del jugador
        Random azar; //Único generador de números aleatorios.

        public Form1() {
            InitializeComponent();
            azar = new Random();
            IniciarParametros();
        }

        private void mnuReiniciar_Click(object sender, EventArgs e) {
            IniciarParametros();
            Refresh();
        }

        public void IniciarParametros() {
            //Inicializa el tablero Fila,Columna
            Tablero = new int[50, 30];
            MovimientoJugador = 0;
        }

        private void timerAnimar_Tick(object sender, EventArgs e) {
            MuevePiedras();
            Refresh();
        }

        private void Form1_KeyDown(object sender, KeyEventArgs e) {
            //Dependiendo de que tecla de flecha presiona el jugador
            if (e.KeyCode == Keys.Left) MovimientoJugador = -1; //Jugador se mueve a la izquierda
            if (e.KeyCode == Keys.Right) MovimientoJugador = 1; //Jugador se mueve a la derecha
            MueveJugador();
            Refresh();
        }

        public void MueveJugador() {
            //Movimiento del jugador
            if (PosJugador + MovimientoJugador < 0 || PosJugador + MovimientoJugador >=
Tablero.GetLength(1)) return;

            Tablero[Tablero.GetLength(0) - 1, PosJugador] = VACIO;
            PosJugador += MovimientoJugador;
            Tablero[Tablero.GetLength(0) - 1, PosJugador] = JUGADOR;
        }

        public void MuevePiedras() {

            //Movimiento de las piedras
            for (int fila = Tablero.GetLength(0)-1; fila > 0; fila--) {
                for (int columna = 0; columna < Tablero.GetLength(1); columna++) {
                    Tablero[fila, columna] = Tablero[fila - 1, columna];
                }
            }

            //Verifica si una piedra golpeó al jugador
            if (Tablero[Tablero.GetLength(0) - 1, PosJugador] == PIEDRA) {
                timerAnimar.Stop();
                MessageBox.Show("Ha sido alcanzado por una piedra", "Fin del juego", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                IniciarParametros();
                timerAnimar.Start();
                return;
            }
```

```
            else
                Tablero[Tablero.GetLength(0) - 1, PosJugador] = JUGADOR;

            //Pone piedras al azar en la primera fila
            int colPiedra, TotalPiedras = 3;
            for (colPiedra = 0; colPiedra < Tablero.GetLength(1); colPiedra++)  Tablero[0, colPiedra] =
VACIO;
            for (int piedra = 1; piedra <= TotalPiedras; piedra++) {
                do {
                    colPiedra = azar.Next(0, Tablero.GetLength(1));
                } while (Tablero[0, colPiedra] == PIEDRA);
                Tablero[0, colPiedra] = PIEDRA;
            }
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Tamaño de cada celda
            int tamFila = 500 / Tablero.GetLength(0);
            int tamColumna = 500 / Tablero.GetLength(1);
            int desplaza = 50;

            //Dibuja el arreglo bidimensional
            for (int Fila = 0; Fila < Tablero.GetLength(0); Fila++) {
                for (int Columna = 0; Columna < Tablero.GetLength(1); Columna++) {
                    int UbicaFila = Fila * tamFila + desplaza;
                    int UbicaColumna = Columna * tamColumna + desplaza;
                    switch (Tablero[Fila, Columna]) {
                        case VACIO: e.Graphics.DrawRectangle(Pens.Blue, UbicaColumna, UbicaFila,
tamColumna, tamFila); break;
                        case PIEDRA: e.Graphics.FillRectangle(Brushes.Black, UbicaColumna, UbicaFila,
tamColumna, tamFila); break;
                        case JUGADOR: e.Graphics.FillRectangle(Brushes.Red, UbicaColumna, UbicaFila,
tamColumna, tamFila); break;
                    }
                }
            }
        }
    }
}
```

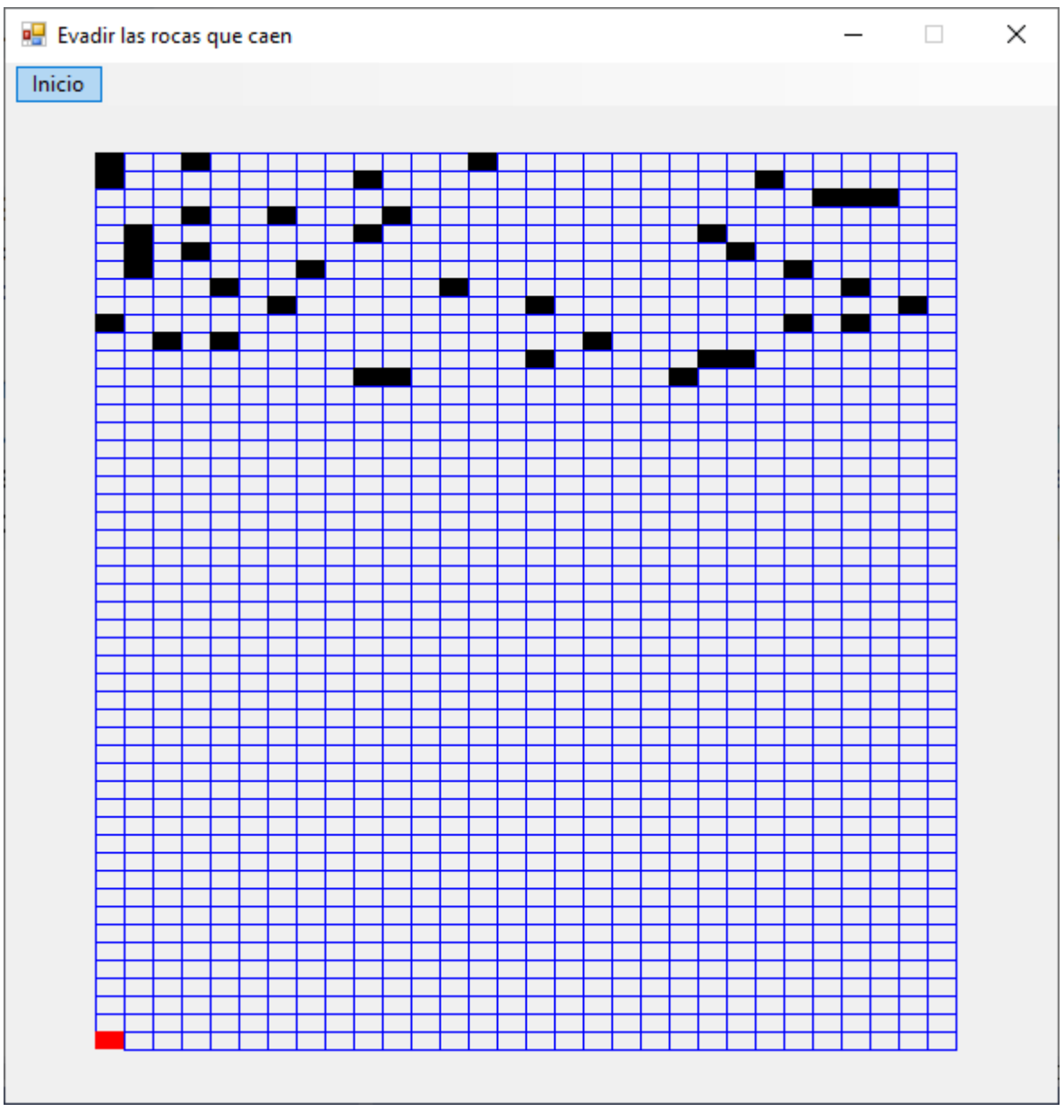


Ilustración 111: Evadir las rocas que caen

Evitar paredes generadas

A medida que se desplaza el jugador va dejando una pared, el jugador debe evitar chocar con esa pared.

10. Videojuegos/04. DejaRastro.zip/Form1.cs

```
//Videojuego: El jugador debe evitar chocar con la pared que va dejando atrás
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Constantes para calificar cada celda del tablero
        private const int VACIO = 0;
        private const int PARED = 1;

        int[,] Tablero; //Dónde ocurre realmente la acción

        //Jugador: dónde está, cómo se mueve
        int PosFila, PosColumna, MueveFila, MueveColumna;

        public Form1() {
            InitializeComponent();
            IniciarParametros();
        }

        private void mnuReiniciar_Click(object sender, EventArgs e) {
            IniciarParametros();
            Refresh();
        }

        public void IniciarParametros() {
            //Inicializa el tablero: Fila,Columna
            Tablero = new int[50, 50];

            //Pone como pared el perímetro del tablero
            for (int fila = 0; fila < Tablero.GetLength(0); fila++) {
                Tablero[fila, 0] = PARED;
                Tablero[fila, Tablero.GetLength(1) - 1] = PARED;
            }

            for (int columna = 0; columna < Tablero.GetLength(1); columna++) {
                Tablero[0, columna] = PARED;
                Tablero[Tablero.GetLength(0) - 1, columna] = PARED;
            }

            //Ubica al jugador
            PosFila = 20;
            PosColumna = 20;
            MueveFila = 0;
            MueveColumna = 1;
        }

        private void timerAnimar_Tick(object sender, EventArgs e) {
            Logica();
            Refresh();
        }

        private void Form1_KeyDown(object sender, KeyEventArgs e) {
            //Dependiendo de que tecla de flecha presiona el jugador
            if (e.KeyCode == Keys.Left && MueveColumna == 0) {
                MueveColumna = -1;
                MueveFila = 0;
            }

            if (e.KeyCode == Keys.Right && MueveColumna == 0) {
                MueveColumna = 1;
                MueveFila = 0;
            }

            if (e.KeyCode == Keys.Up && MueveFila == 0) {
                MueveFila = -1;
                MueveColumna = 0;
            }

            if (e.KeyCode == Keys.Down && MueveFila == 0) {
                MueveFila = 1;
                MueveColumna = 0;
            }
```

```
        }

        public void Logica() {
            PosFila += MueveFila;
            PosColumna += MueveColumna;

            //Verifica si colisionó con una pared
            if (Tablero[PosFila, PosColumna] == PARED) {
                timerAnimar.Stop();
                MessageBox.Show("Ha colisionado con la pared", "Fin del juego", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                IniciarParametros();
                timerAnimar.Start();
                return;
            }

            Tablero[PosFila, PosColumna] = PARED;
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Tamaño de cada celda
            int tamFila = 500 / Tablero.GetLength(0);
            int tamColumna = 500 / Tablero.GetLength(1);
            int desplaza = 50;

            //Dibuja el arreglo bidimensional
            for (int Fila = 0; Fila < Tablero.GetLength(0); Fila++) {
                for (int Columna = 0; Columna < Tablero.GetLength(1); Columna++) {
                    int UbicaFila = Fila * tamFila + desplaza;
                    int UbicaColumna = Columna * tamColumna + desplaza;
                    switch (Tablero[Fila, Columna]) {
                        case VACIO: e.Graphics.DrawRectangle(Pens.Blue, UbicaColumna, UbicaFila,
tamColumna, tamFila); break;
                        case PARED: e.Graphics.FillRectangle(Brushes.Black, UbicaColumna, UbicaFila,
tamColumna, tamFila); break;
                    }
                }
            }
        }
    }
}
```

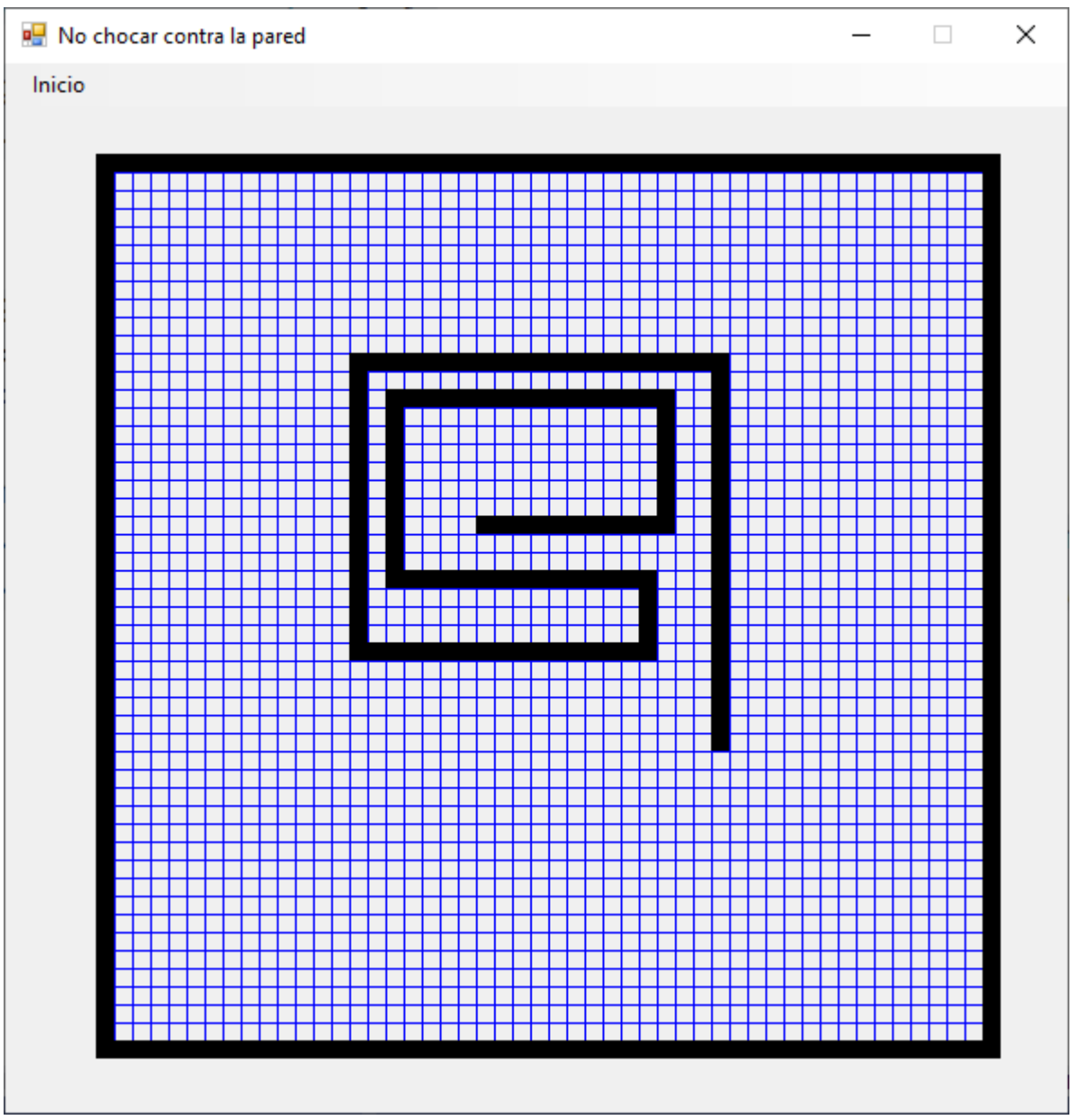


Ilustración 112: Evitar chocar con las paredes

Detecta Colisiones

10. Videojuegos/05. Colision.zip/Form1.cs

```
//Videojuego: Detecta colisiones entre dos cuadrados
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Constantes para calificar cada celda del tablero
        private const int ESVACIO = 0;
        private const int ESOBSTACULO = 1;
        private const int ESJUGADOR = 2;

        int[,] Tablero; //Dónde ocurre realmente la acción
        int JugadorFila, JugadorColumna, JugadorTamano; //Coordenadas del jugador
        int MueveFila, MueveColumna; //Desplazamiento del jugador
        int ObstaculoFila, ObstaculoColumna, ObstaculoTamano; //Coordenadas del obstáculo

        public Form1() {
            InitializeComponent();
            IniciarParametros();
        }

        private void mnuReiniciar_Click(object sender, EventArgs e) {
            IniciarParametros();
            Refresh();
        }

        public void IniciarParametros() {
            //Inicializa el tablero.
            Tablero = new int[30, 30];

            //Inicializa la posición y tamaño del jugador
            JugadorFila = 0;
            JugadorColumna = 0;
            JugadorTamano = 5;

            //Desplazamiento del jugador
            MueveFila = 0;
            MueveColumna = 1;

            //Inicializa la posición y tamaño del obstáculo
            ObstaculoFila = 15;
            ObstaculoColumna = 15;
            ObstaculoTamano = 10;
        }

        private void timerAnimar_Tick(object sender, EventArgs e) {
            Logica();
            Refresh();
        }

        private void Form1_KeyDown(object sender, KeyEventArgs e) {
            MueveFila = 0;
            MueveColumna = 0;

            if (e.KeyCode == Keys.Right) MueveFila = 1;
            if (e.KeyCode == Keys.Left)  MueveFila = -1;
            if (e.KeyCode == Keys.Up) MueveColumna = -1;
            if (e.KeyCode == Keys.Down) MueveColumna = 1;
        }

        public void Logica() {
            if (JugadorFila + MueveFila < 0 || JugadorColumna + MueveColumna < 0 || JugadorFila +
MueveFila + JugadorTamano > Tablero.GetLength(0) || JugadorColumna + MueveColumna + JugadorTamano >
Tablero.GetLength(1)) return;

            //Borra la antigua posición del jugador
            for (int cont = 0; cont < JugadorTamano; cont++) {
                Tablero[JugadorFila + cont, JugadorColumna] = ESVACIO;
                Tablero[JugadorFila + cont, JugadorColumna + JugadorTamano - 1] = ESVACIO;
                Tablero[JugadorFila, JugadorColumna + cont] = ESVACIO;
                Tablero[JugadorFila + JugadorTamano - 1, JugadorColumna + cont] = ESVACIO;
            }

            //Desplaza al jugador
            JugadorFila += MueveFila;
```

```
            JugadorColumna += MueveColumna;

            //Grafica la nueva posición del jugador
            for (int cont = 0; cont < JugadorTamano; cont++) {
                Tablero[JugadorFila + cont, JugadorColumna] = ESJUGADOR;
                Tablero[JugadorFila + cont, JugadorColumna + JugadorTamano - 1] = ESJUGADOR;
                Tablero[JugadorFila, JugadorColumna + cont] = ESJUGADOR;
                Tablero[JugadorFila + JugadorTamano - 1, JugadorColumna + cont] = ESJUGADOR;
            }

            //Grafica la posición del obstáculo
            for (int cont = 0; cont < ObstaculoTamano; cont++) {
                Tablero[ObstaculoFila + cont, ObstaculoColumna] = ESOBSTACULO;
                Tablero[ObstaculoFila + cont, ObstaculoColumna + ObstaculoTamano - 1] = ESOBSTACULO;
                Tablero[ObstaculoFila, ObstaculoColumna + cont] = ESOBSTACULO;
                Tablero[ObstaculoFila + ObstaculoTamano - 1, ObstaculoColumna + cont] = ESOBSTACULO;
            }

            //Chequea si hay colisión
            bool HayColision = false;

            //Punto superior izquierdo
            if (JugadorFila >= ObstaculoFila && JugadorColumna >= ObstaculoColumna && JugadorFila <
ObstaculoFila + ObstaculoTamano && JugadorColumna < ObstaculoColumna + ObstaculoTamano)
                HayColision = true;

            //Punto superior derecho
            else if (JugadorFila + JugadorTamano > ObstaculoFila && JugadorColumna >= ObstaculoColumna &&
JugadorFila + JugadorTamano < ObstaculoFila + ObstaculoTamano && JugadorColumna < ObstaculoColumna +
ObstaculoTamano)
                HayColision = true;

            //Punto inferior izquierdo
            else if (JugadorFila >= ObstaculoFila && JugadorColumna + JugadorTamano > ObstaculoColumna &&
JugadorFila < ObstaculoFila + ObstaculoTamano && JugadorColumna + JugadorTamano < ObstaculoColumna +
ObstaculoTamano)
                HayColision = true;

            //Punto inferior derecho
            else if (JugadorFila + JugadorTamano > ObstaculoFila && JugadorColumna + JugadorTamano >
ObstaculoColumna && JugadorFila + JugadorTamano < ObstaculoFila + ObstaculoTamano && JugadorColumna +
JugadorTamano < ObstaculoColumna + ObstaculoTamano)
                HayColision = true;

            if (HayColision) {
                lblColision.Text = "COLISIÓN";
                lblColision.BackColor = Color.Red;
            }
            else {
                lblColision.Text = "No hay colisión";
                lblColision.BackColor = Color.White;
            }

        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Tamaño de cada celda
            int tamanoFila = 500 / Tablero.GetLength(0);
            int tamanoColumna = 500 / Tablero.GetLength(1);
            int desplaza = 50;

            //Dibuja el arreglo bidimensional
            for (int Fila = 0; Fila < Tablero.GetLength(0); Fila++) {
                for (int Columna = 0; Columna < Tablero.GetLength(1); Columna++) {
                    int UbicaFila = Fila * tamanoFila + desplaza;
                    int UbicaColumna = Columna * tamanoColumna + desplaza;
                    switch (Tablero[Fila, Columna]) {
                        case ESVACIO: e.Graphics.DrawRectangle(Pens.Blue, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                        case ESOBSTACULO: e.Graphics.FillRectangle(Brushes.Black, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                        case ESJUGADOR: e.Graphics.FillRectangle(Brushes.Red, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                    }
                }
            }
        }
    }
}
```

Ilustración 113: Figuras no colisionan

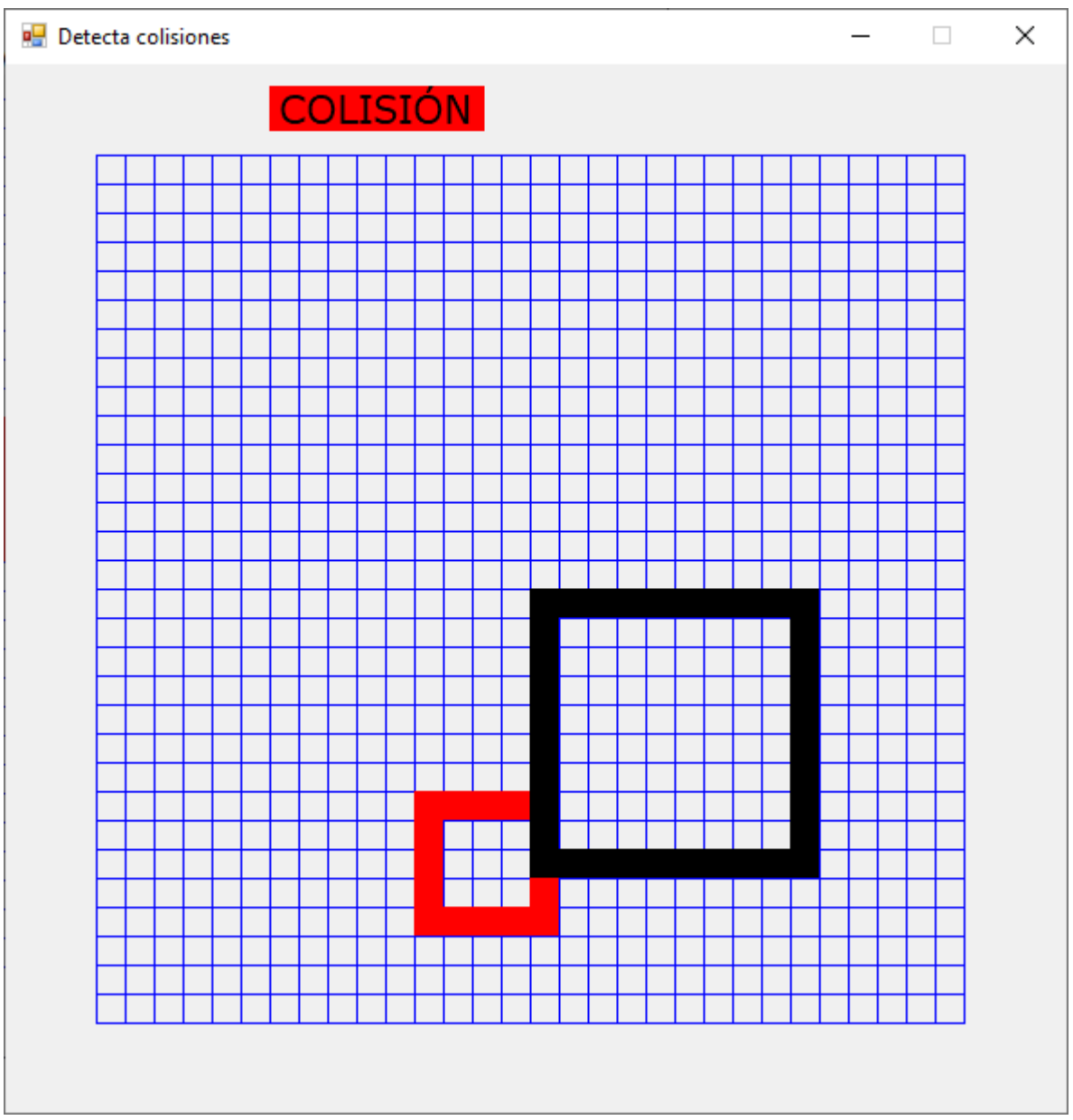


Ilustración 114: Figuras colisionan

Rebote y agregar paredes

10. Videojuegos/06. Rebotando.zip/Form1.cs

```
//Rebote de una pelota. El usuario puede agregar o quitar paredes
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Constantes para calificar cada celda del tablero
        private const int INACTIVA = 0;
        private const int ACTIVA = 1;
        private const int PELOTA = 2;

        //Posición y desplazamiento de la pelota
        private int PelotaFila, PelotaColumna, incFila, incColumna;

        int[,] Tablero; //Dónde ocurre realmente la acción

        //Tamaño de cada celda
        int tamanoFila, tamanoColumna, desplaza;

        public Form1() {
            InitializeComponent();
            IniciarParametros();
        }

    public void IniciarParametros() {
        Tablero = new int[30, 30]; //Inicializa el tablero.

        //Dibuja el perímetro
        for (int fila = 0; fila < Tablero.GetLength(0); fila++) {
            Tablero[fila, 0] = ACTIVA;
            Tablero[fila, Tablero.GetLength(1) - 1] = ACTIVA;
        }

        for (int columna = 0; columna < Tablero.GetLength(0); columna++) {
            Tablero[0, columna] = ACTIVA;
            Tablero[Tablero.GetLength(0) - 1, columna] = ACTIVA;
        }

        //Tamaño de cada celda
        tamanoFila = 500 / Tablero.GetLength(0);
        tamanoColumna = 500 / Tablero.GetLength(1);
        desplaza = 50;

        //Pelota
        PelotaFila = 12;
        PelotaColumna = 15;
        incFila = 1;
        incColumna = 1;

        TimerAnimar.Start();
    }


        private void TimerAnimar_Tick(object sender, System.EventArgs e) {
            Logica();
            Refresh(); //Visual de la animación
        }

        //Lógica del rebote
        private void Logica() {
            if (PelotaFila + incFila < 0 || PelotaFila + incFila >= Tablero.GetLength(0) || PelotaColumna
+ incColumna < 0 || PelotaColumna + incColumna >= Tablero.GetLength(1)) return;

            //Si golpea alguna pared
            bool cambio = false;
            if (Tablero[PelotaFila + 1, PelotaColumna] == ACTIVA) { incFila *= -1; cambio = true; }
            if (Tablero[PelotaFila - 1, PelotaColumna] == ACTIVA) { incFila *= -1; cambio = true; }
            if (Tablero[PelotaFila, PelotaColumna + 1] == ACTIVA) { incColumna *= -1; cambio = true; }
            if (Tablero[PelotaFila, PelotaColumna - 1] == ACTIVA) { incColumna *= -1; cambio = true; }

            //Si golpea una punta
            if (!cambio) {
                if (Tablero[PelotaFila + incFila, PelotaColumna + incColumna] == ACTIVA) {
                    incFila *= -1;
                    incColumna *= -1;
                }
```

```
            }

            //Dibuja la nueva posición siempre y cuando esté permitida
            if (Tablero[PelotaFila+incFila, PelotaColumna+incColumna] == INACTIVA) {
                Tablero[PelotaFila, PelotaColumna] = INACTIVA;
                PelotaFila += incFila;
                PelotaColumna += incColumna;
                Tablero[PelotaFila, PelotaColumna] = PELOTA;
            }
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            //Dibuja el arreglo bidimensional
            for (int Fila = 0; Fila < Tablero.GetLength(0); Fila++) {
                for (int Columna = 0; Columna < Tablero.GetLength(1); Columna++) {
                    int UbicaFila = Fila * tamanoFila + desplaza;
                    int UbicaColumna = Columna * tamanoColumna + desplaza;
                    switch (Tablero[Fila, Columna]) {
                        case INACTIVA: e.Graphics.DrawRectangle(Pens.Blue, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                        case ACTIVA: e.Graphics.FillRectangle(Brushes.Black, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                        case PELOTA: e.Graphics.FillRectangle(Brushes.Red, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                    }
                }
            }
        }

        //Cuando da clic sobre una celda, la activa o la desactiva
        private void Form1_MouseClick(object sender, MouseEventArgs e) {
            int Fila = (e.X - desplaza) / tamanoFila;
            int Columna = (e.Y - desplaza) / tamanoColumna;

            if (Fila >= 0 && Columna >= 0 && Fila < Tablero.GetLength(0) && Columna <
Tablero.GetLength(1)) {
                switch (Tablero[Fila, Columna]) {
                    case INACTIVA: Tablero[Fila, Columna] = ACTIVA; break;
                    case ACTIVA: Tablero[Fila, Columna] = INACTIVA; break;
                }
            }
        }
    }
}
```

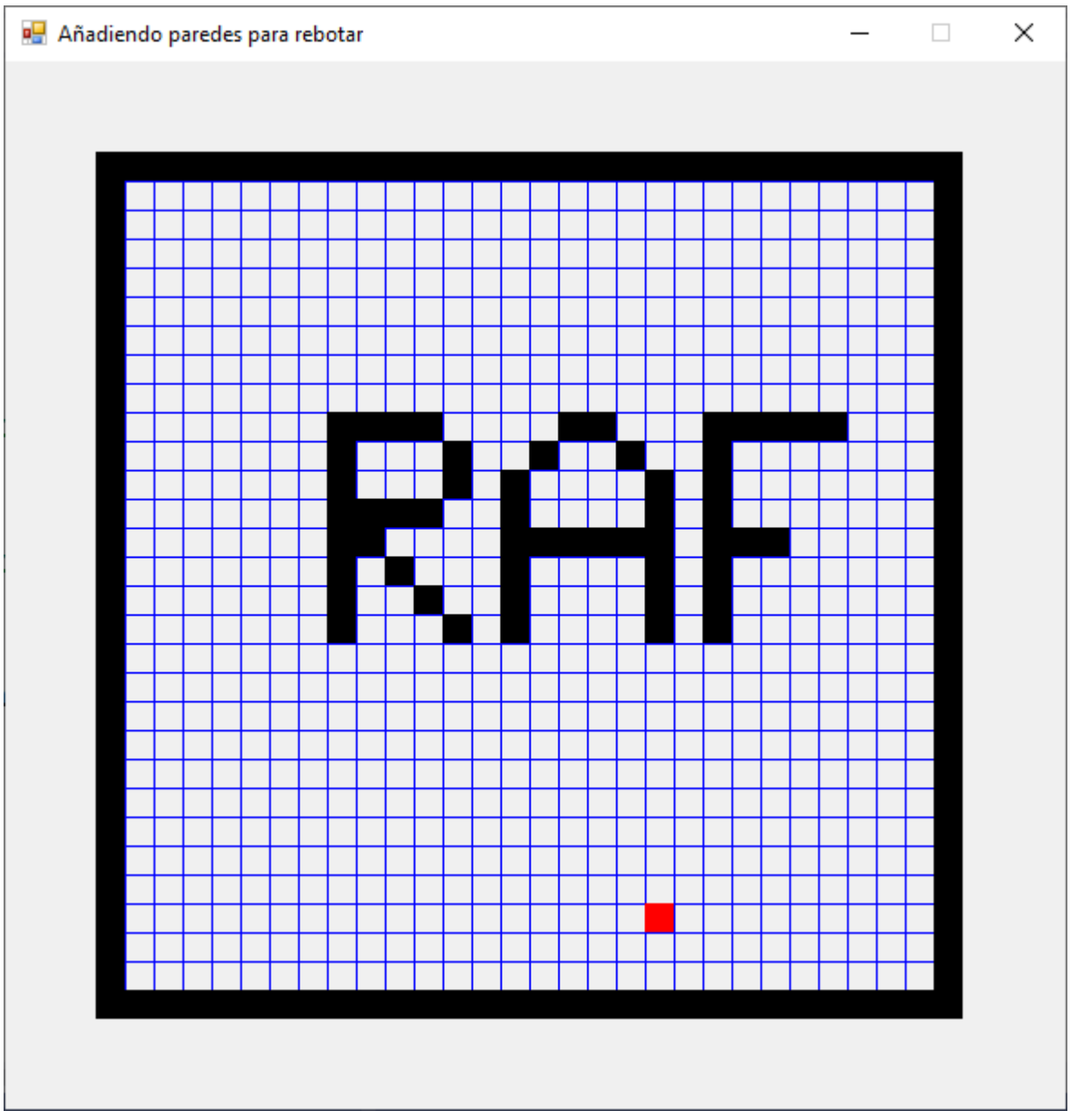


Ilustración 115: Rebote y agregar paredes con el mouse

Simulaciones

Gráfico vectorial que nace de una cadena

Se da una cadena (string) que contiene cuatro letras: N, S, E, O (Norte, Sur, Este, Oeste) y un número de un sólo digito. El programa al leer cada letra hace una línea vertical hacia arriba (Norte), vertical hacia abajo (Sur), horizontal a la izquierda (Oeste), horizontal a la derecha (Este). Líneas de la misma longitud multiplicada por el número de un solo dígito. Lo interesante es que si varía el tamaño de la línea el dibujo crece o decrece en tamaño, pero no pierde definición. Eso es un gráfico vectorial que nace de una cadena. Haciendo una analogía con la biología, se simula el ADN con esa cadena (el genotipo) y el dibujo resultante es el fenotipo.

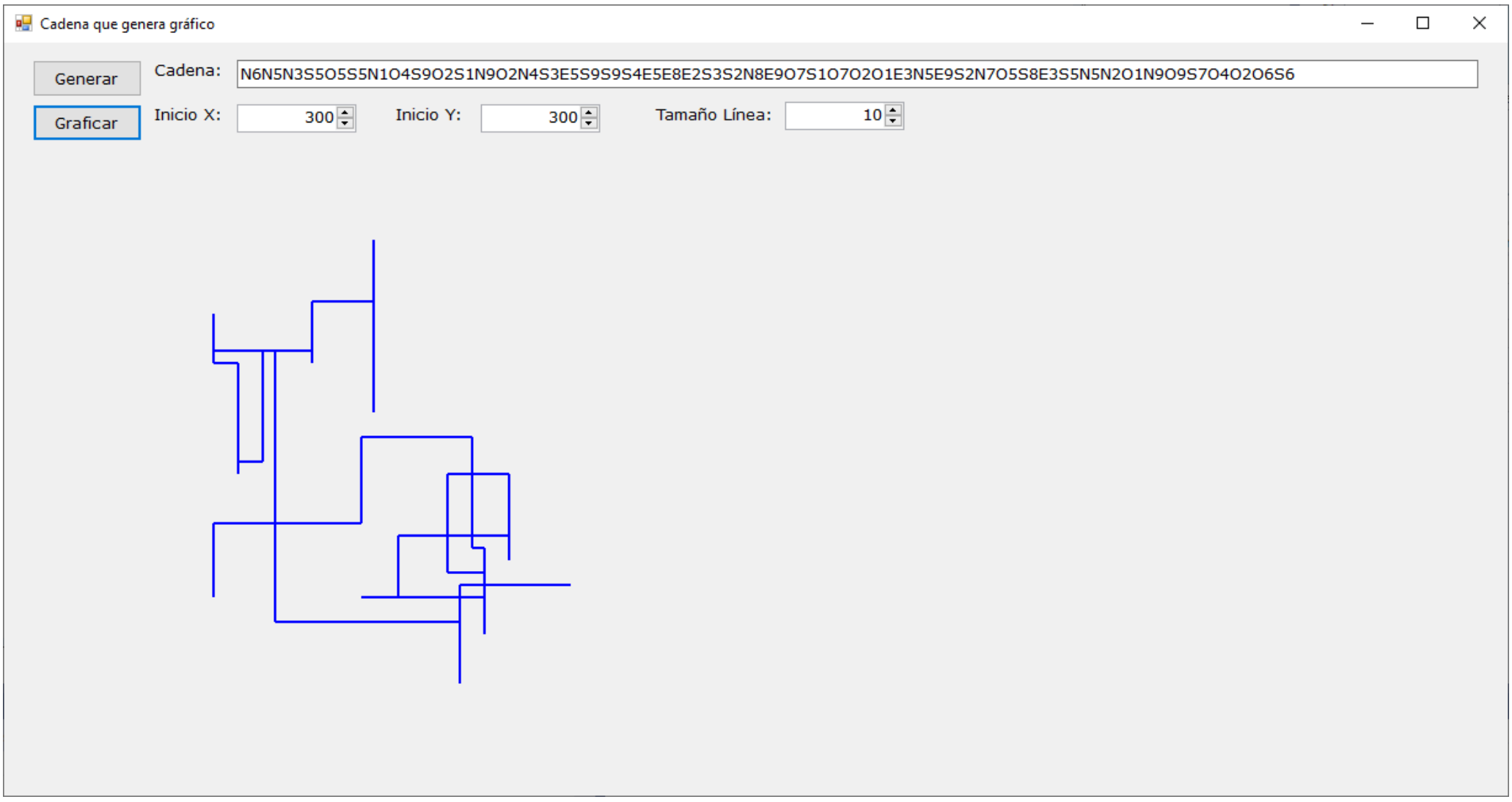


Ilustración 116: Cadena original y gráfico resultante

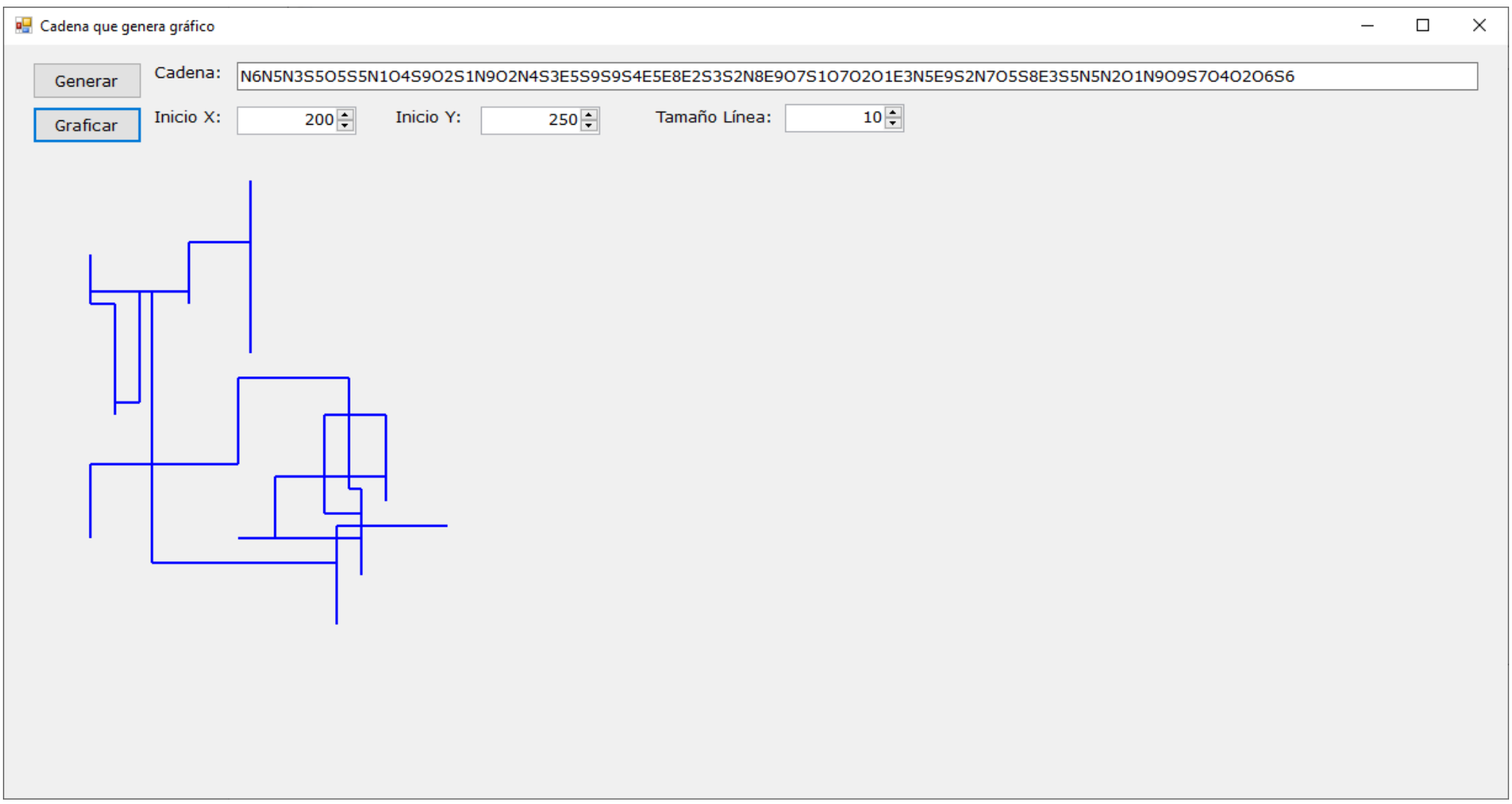


Ilustración 117: El gráfico cambia de punto de inicio

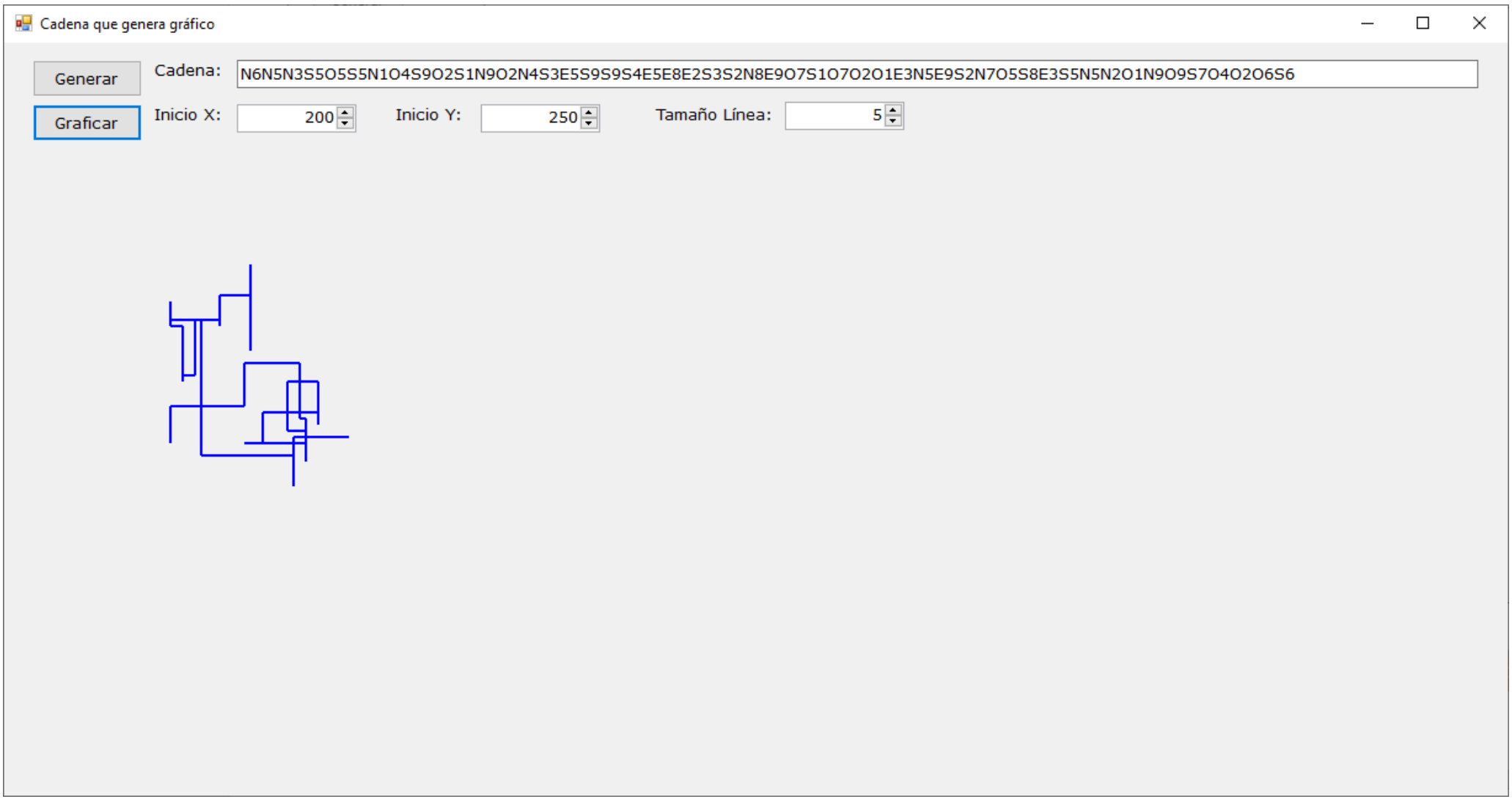


Ilustración 118: El gráfico cambia de tamaño

11. Simulación/01. Vectorial.zip/Form1.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace GraficoCSharp {
    public partial class Form1 : Form {
        int tamano, iniX, iniY;
        string camino;

        public Form1() {
            InitializeComponent();
            camino = "";
        }

        private void Form1_Paint(object sender, PaintEventArgs e) {
            if (camino.Length == 0) return;

            Graphics grafico = e.Graphics;
            Pen lapiz = new Pen(Color.Blue, 2);

            //Punto de Inicio y fin de los segmentos
            int finX = iniX, finY = iniY;

            //Hace el gráfico vectorial
            for (int cont = 0; cont < camino.Length; cont += 2) { //El camino guía como se dibuja
                int avance = camino[cont + 1] - '0';
                switch (camino[cont]) {
                    case 'N': finY = iniY - avance * tamano; break;
                    case 'S': finY = iniY + avance * tamano; break;
                    case 'E': finX = iniX + avance * tamano; break;
                    case 'O': finX = iniX - avance * tamano; break;
                }
                grafico.DrawLine(lapiz, iniX, iniY, finX, finY);
                iniX = finX;
                iniY = finY;
            }
        }

        /* Genera una cadena de texto al azar con la siguiente estructura:
         * letra+numero+letra+numero+.....+letra+numero
         * Donde letra puede ser A, B, D, I  (Arriba, aBajo, Derecha, Izquierda)
         * Y número desde 1 hasta 9
```

```
         * Esa cadena será interpretada en Paint para hacer el gráfico vectorial */
        private void btnGenerar_Click(object sender, System.EventArgs e) {
            Random azar = new Random();
            int avanzar;
            string cadena = "";
            for (int cont = 1; cont <= 50; cont++) { //50 pares de letra+numero se generarán
                switch (azar.Next() % 4) {
                    case 0: cadena += "N"; break;
                    case 1: cadena += "S"; break;
                    case 2: cadena += "E"; break;
                    case 3: cadena += "O"; break;
                }
                avanzar = azar.Next() % 9 + 1;
                cadena += avanzar.ToString();
            }
            txtCadena.Text = cadena;
        }

        //Hace que el evento Paint() se dispare
        private void btnGraficar_Click(object sender, EventArgs e) {
            tamano = Convert.ToInt32(numTamano.Value); //Tamaño segmento linea
            iniX = Convert.ToInt32(numIniX.Value); //Punto X inicial del gráfico de tortuga
            iniY = Convert.ToInt32(numIniY.Value); //Punto Y inicial del gráfico de tortuga
            camino = txtCadena.Text;

            Refresh();
        }
    }
}
```

Algoritmo genético que determina la cadena de un dibujo

Es el proceso inverso del ejemplo anterior y es dar con la cadena que genera un gráfico dado por el usuario que lo ha dibujado. Para ello se recurre a una rama de la Inteligencia Artificial conocida como algoritmos evolutivos, más precisamente los algoritmos genéticos.

El método no es 100% preciso pero sirve para ilustrar el uso de este tipo de algoritmos.

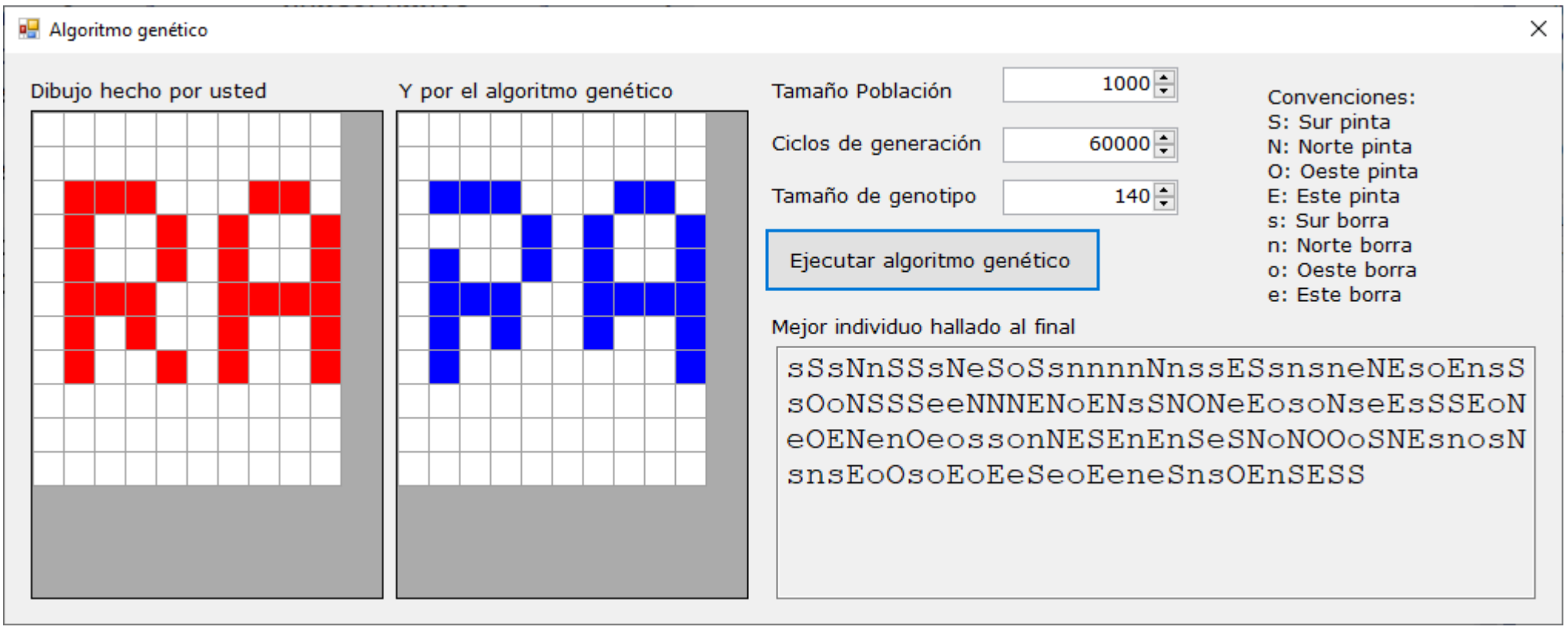


Ilustración 119: A la izquierda, gráfico hecho por el usuario. A la derecha el gráfico obtenido por la cadena encontrada

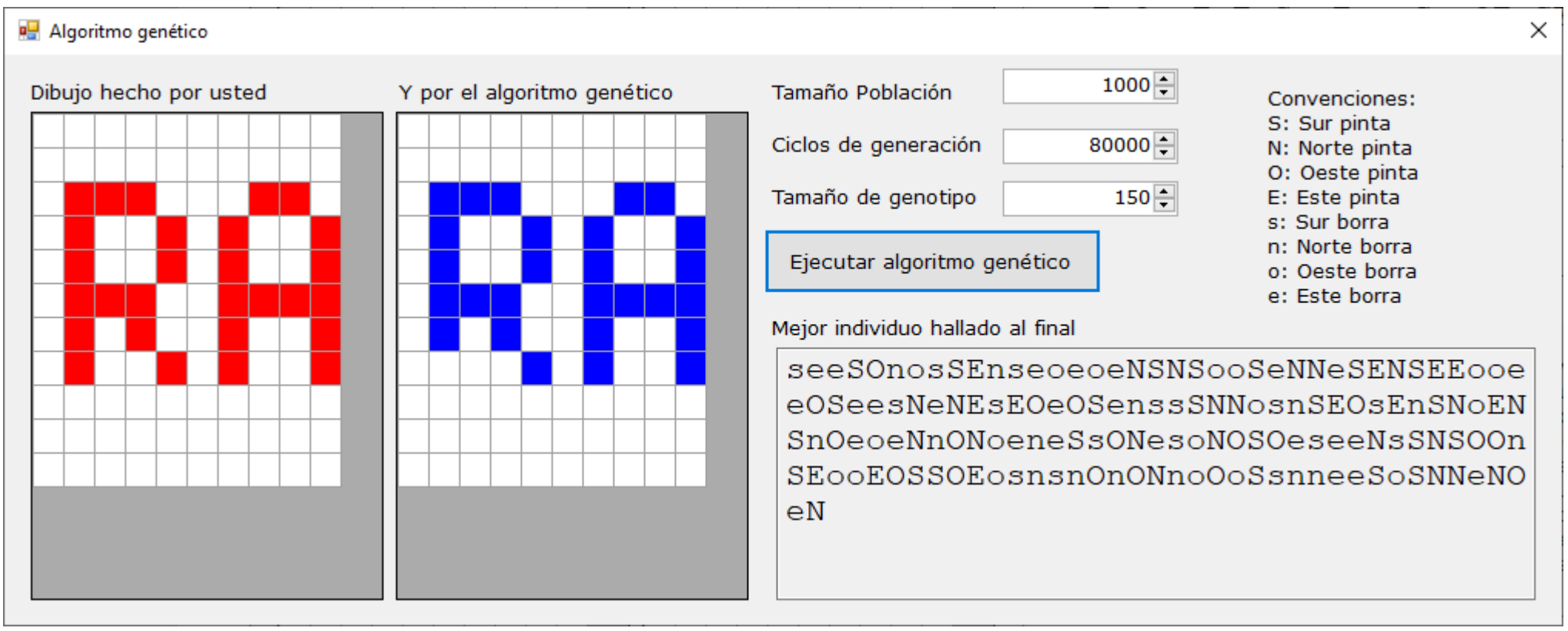


Ilustración 120:A la izquierda, gráfico hecho por el usuario. A la derecha el gráfico obtenido por la cadena encontrada

11. Simulación/02. AGDibujo.zip/Form1.cs

```
/* Dar con la cadena que genera un gráfico dado. Para ello se recurre a una rama de la Inteligencia
Artificial
 * conocida como algoritmos evolutivos, más precisamente los algoritmos genéticos.
 * */
using System;
using System.Drawing;
using System.Windows.Forms;

namespace AGDibujo {
    public partial class Form1 : Form {
        private int NUMFILAS = 10, NUMCOLUMNAS = 10;
        public Form1() {
            InitializeComponent();
            lblManual.Text = "Convenciones: " + Environment.NewLine +
                "S: Sur pinta" + Environment.NewLine +
                "N: Norte pinta" + Environment.NewLine +
                "O: Oeste pinta" + Environment.NewLine +
                "E: Este pinta" + Environment.NewLine +
                "s: Sur borra" + Environment.NewLine +
                "n: Norte borra" + Environment.NewLine +
```

```
            "o: Oeste borra" + Environment.NewLine +
            "e: Este borra";
    }

    //Llena las filas de los grid
    private void Form1_Load(object sender, EventArgs e) {
        //Llena de filas el GridView
        for (int filas = 0; filas < NUMFILAS; filas++) {
            dgDibujo.Rows.Add((DataGridViewRow)dgDibujo.Rows[0].Clone());
            dgMejor.Rows.Add((DataGridViewRow)dgMejor.Rows[0].Clone());
        }
    }

    //Cuando el usuario de clic en alguna celda, esta cambia de color
    private void dgDibujo_CellClick(object sender, DataGridViewCellEventArgs e) {
        if (dgDibujo.Rows[e.RowIndex].Cells[e.ColumnIndex].Style.BackColor == Color.Red)
            dgDibujo.Rows[e.RowIndex].Cells[e.ColumnIndex].Style.BackColor = Color.White;
        else
            dgDibujo.Rows[e.RowIndex].Cells[e.ColumnIndex].Style.BackColor = Color.Red;

        dgDibujo.CurrentCell = null; //No mostrar el cursor de la celda actual
    }

    //Activa el proceso de algoritmos genéticos
    private void btnGenetico_Click(object sender, EventArgs e) {
        ProcesoGenetico();
    }

    }

    //Algoritmo genético
    private void ProcesoGenetico() {
        //1. Pasa a un arreglo bidimensional el contenido del grid

        int[,] Tablero = new int[NUMFILAS, NUMCOLUMNAS];
        for (int fila = 0; fila < NUMFILAS; fila++)
            for (int columna = 0; columna < NUMCOLUMNAS; columna++)
                if (dgDibujo.Rows[fila].Cells[columna].Style.BackColor == Color.Red) Tablero[fila,
columna] = 1;

        //2. Genera la población de individuos
        Random azar = new Random();
        int tamGenotipo = Convert.ToInt32(numTamGenotipo.Value);
        int tamPoblacion = Convert.ToInt32(numPoblacion.Value);
        string[] individuos = new string[tamPoblacion];
        for (int genera = 0; genera < tamPoblacion; genera++)
            individuos[genera] = GeneraIndividuo(azar, tamGenotipo);

        //3. Ciclo de algoritmo genético
        int ciclos = Convert.ToInt32(numCiclos.Value);
        for (int cont = 1; cont <= ciclos; cont++) {

            //4. Toma dos individuos al azar
            int indivA = azar.Next() % tamPoblacion;
            int indivB = azar.Next() % tamPoblacion;

            //5. Evalúa cual individuo es mejor
            int adaptaA = EvaluaIndividuo(individuos[indivA], Tablero);
            int adaptaB = EvaluaIndividuo(individuos[indivB], Tablero);
            string Mejor;
            if (adaptaA < adaptaB)
                Mejor = individuos[indivA];
            else
                Mejor = individuos[indivB];

            //6. Muta alguna parte al azar. Nota: La mutación puede dañar al individuo hijo
            string hijo;
            do {
                char[] arreglo = Mejor.ToCharArray();
                int posMuta = azar.Next() % tamGenotipo;
                switch (azar.Next() % 8) {
                    case 0: arreglo[posMuta] = 'O'; break;
                    case 1: arreglo[posMuta] = 'E'; break;
                    case 2: arreglo[posMuta] = 'N'; break;
                    case 3: arreglo[posMuta] = 'S'; break;
                    case 4: arreglo[posMuta] = 'o'; break;
                    case 5: arreglo[posMuta] = 'e'; break;
                    case 6: arreglo[posMuta] = 'n'; break;
                    case 7: arreglo[posMuta] = 's'; break;
                }
                hijo = new string(arreglo);
```

```
            } while (EvaluaIndividuo(hijo, Tablero) == -1); //Hasta que sea una mutación viable

            //7. Sobreescribe el peor adaptado

            if (adaptaA < adaptaB)
                individuos[indivB] = hijo;
            else
                individuos[indivA] = hijo;
        }

        //8. Muestra el mejor individuo hallado
        int ElMejor = -1, MejorAproxima = int.MaxValue, acumula = 0;
        for (int cont = 0; cont < individuos.Length; cont++) {
            int evaluando = EvaluaIndividuo(individuos[cont], Tablero);
            acumula += evaluando;
            if (evaluando < MejorAproxima) {
                MejorAproxima = evaluando;
                ElMejor = cont;
            }
        }

        txtMejor.Text = individuos[ElMejor];
        LlenarGrid(txtMejor.Text, Tablero);
    }

    //Retorna una cadena de dibujo que sería el individuo
    private string GeneraIndividuo(Random azar, int tamGenotipo) {
        string individuo = "";
        int fila = 0, columna = 0, movimiento, varFila = 0, varColumna = 0;

        for (int cont = 1; cont <= tamGenotipo; cont++) { //Número de instrucciones que tendrá el
individuo
            do { //Hasta que se encuentre un movimiento viable
                movimiento = azar.Next() % 8;
                switch (movimiento) {
                    case 0:
                    case 1: varFila = 0; varColumna = -1; break; //Arriba
                    case 2:
                    case 3: varFila = 0; varColumna = 1; break; //aBajo
                    case 4:
                    case 5: varFila = -1; varColumna = 0; break; //Izquierda
                    case 6:
                    case 7: varFila = 1; varColumna = 0; break; //Derecha
                }
            } while (fila + varFila < 0 || fila + varFila > NUMFILAS - 1 || columna + varColumna < 0
|| columna + varColumna > NUMCOLUMNAS - 1);

            switch (movimiento) {  //Se añade ese movimiento viable
                case 0: individuo += "O"; break;
                case 1: individuo += "o"; break;
                case 2: individuo += "E"; break;
                case 3: individuo += "e"; break;
                case 4: individuo += "N"; break;
                case 5: individuo += "n"; break;
                case 6: individuo += "S"; break;
                case 7: individuo += "s"; break;
            }

            //Se actualiza la posición del cursor
            fila += varFila;
            columna += varColumna;
        }
        return individuo;  //Retorna el individuo generado
    }

    //Evalúa la adaptación del individuo. Entre más cercano a cero es mejor.
    private int EvaluaIndividuo(string cadena, int[,] tablero) {

        //En un arreglo bidimensional se guarda el camino dibujado por el individuo
        int[,] pinta = new int[NUMFILAS, NUMCOLUMNAS];

        //El caminar del individuo
        int fila = 0, columna = 0, valor = 0;
        for (int letra = 0; letra < cadena.Length; letra++) {
            switch (cadena[letra]) {
                case 'O':
                case 'o': columna--; break;
                case 'E':
                case 'e': columna++; break;
                case 'N':
```

```
                    case 'n': fila--; break;
                    case 'S':
                    case 's': fila++; break;
                }
                switch (cadena[letra]) {
                    case 'O':
                    case 'E':
                    case 'N':
                    case 'S': valor = 1; break;
                    case 'o':
                    case 'e':
                    case 'n':
                    case 's': valor = 0; break;

                }

                //Si una mutación hace inviable al individuo retorna -1
                if (fila < 0 || columna < 0 || fila > NUMFILAS-1 || columna > NUMCOLUMNAS - 1) return -1;

                //Actualiza el tablero como dibuja el individuo
                pinta[fila, columna] = valor;
            }

            //El valor absoluto de la resta entre lo que dibuja el individuo y el dibujo del usuario, será
la adaptación
            int adapta = 0;
            for (fila = 0; fila < NUMFILAS; fila++)
                for (columna = 0; columna < NUMCOLUMNAS; columna++)
                    adapta += Math.Abs(pinta[fila, columna] - tablero[fila, columna]);

            return adapta;
        }

        //Muestra el tablero con lo pedido por el usuario y el individuo generado
        private void LlenarGrid(string cadena, int[,] tablero) {
            int fila, columna;
            for (fila = 0; fila < NUMFILAS; fila++)
                for (columna = 0; columna < NUMCOLUMNAS; columna++)
                    if (tablero[fila, columna] == 0)
                        dgDibujo.Rows[fila].Cells[columna].Style.BackColor = Color.White;
                    else
                        dgDibujo.Rows[fila].Cells[columna].Style.BackColor = Color.Red;

            //En un arreglo bidimensional se guarda el camino dibujado por el individuo
            int[,] pinta = new int[NUMFILAS, NUMCOLUMNAS];

            //El caminar del individuo
            fila = 0;
            columna = 0;
            int valor = 0;
            for (int letra = 0; letra < cadena.Length; letra++) {
                switch (cadena[letra]) {
                    case 'O':
                    case 'o': columna--; break;
                    case 'E':
                    case 'e': columna++; break;
                    case 'N':
                    case 'n': fila--; break;
                    case 'S':
                    case 's': fila++; break;
                }
                switch (cadena[letra]) {
                    case 'O':
                    case 'E':
                    case 'N':
                    case 'S': valor = 1; break;
                    case 'o':
                    case 'e':
                    case 'n':
                    case 's': valor = 0; break;

                }
                //Actualiza el tablero como dibuja el individuo
                pinta[fila, columna] = valor;
            }

            for (fila = 0; fila < NUMFILAS; fila++)
                for (columna = 0; columna < NUMCOLUMNAS; columna++)
                    if (pinta[fila, columna] == 0)
```

```
                        dgMejor.Rows[fila].Cells[columna].Style.BackColor = Color.White;
                    else
                        dgMejor.Rows[fila].Cells[columna].Style.BackColor = Color.Blue;

            dgDibujo.CurrentCell = null; //No mostrar el cursor de la celda actual
            dgMejor.CurrentCell = null; //No mostrar el cursor de la celda actual
        }
    }
}
```

Población e infección

Simula una población de individuos en la cual uno tiene una infección y como esta se va esparciendo por la población. Los individuos se mueven al azar y si un infectado (en rojo) coincide en la misma casilla que uno sano (en negro), el individuo sano se infecta.

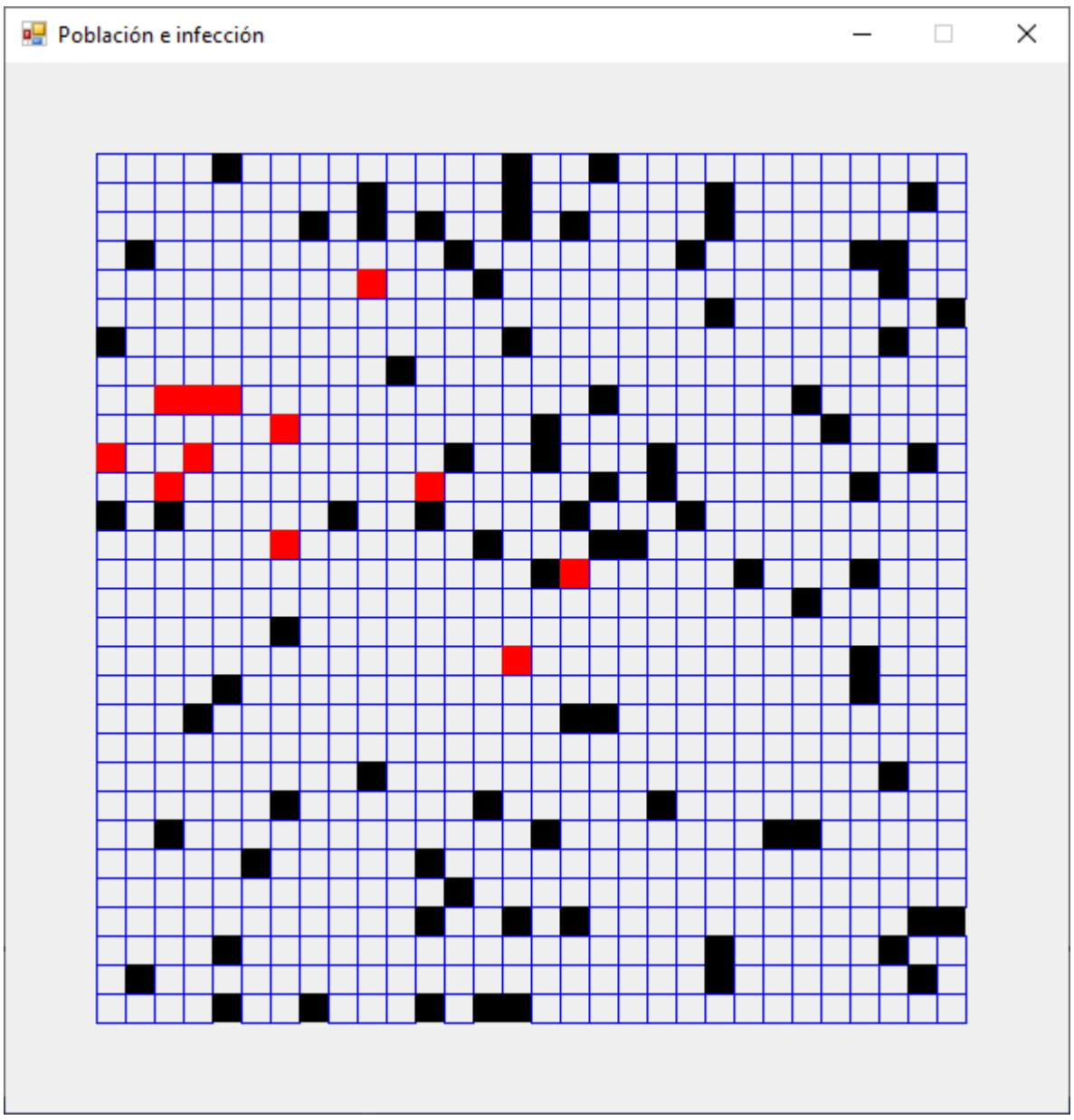


Ilustración 121: La infección empieza a extenderse

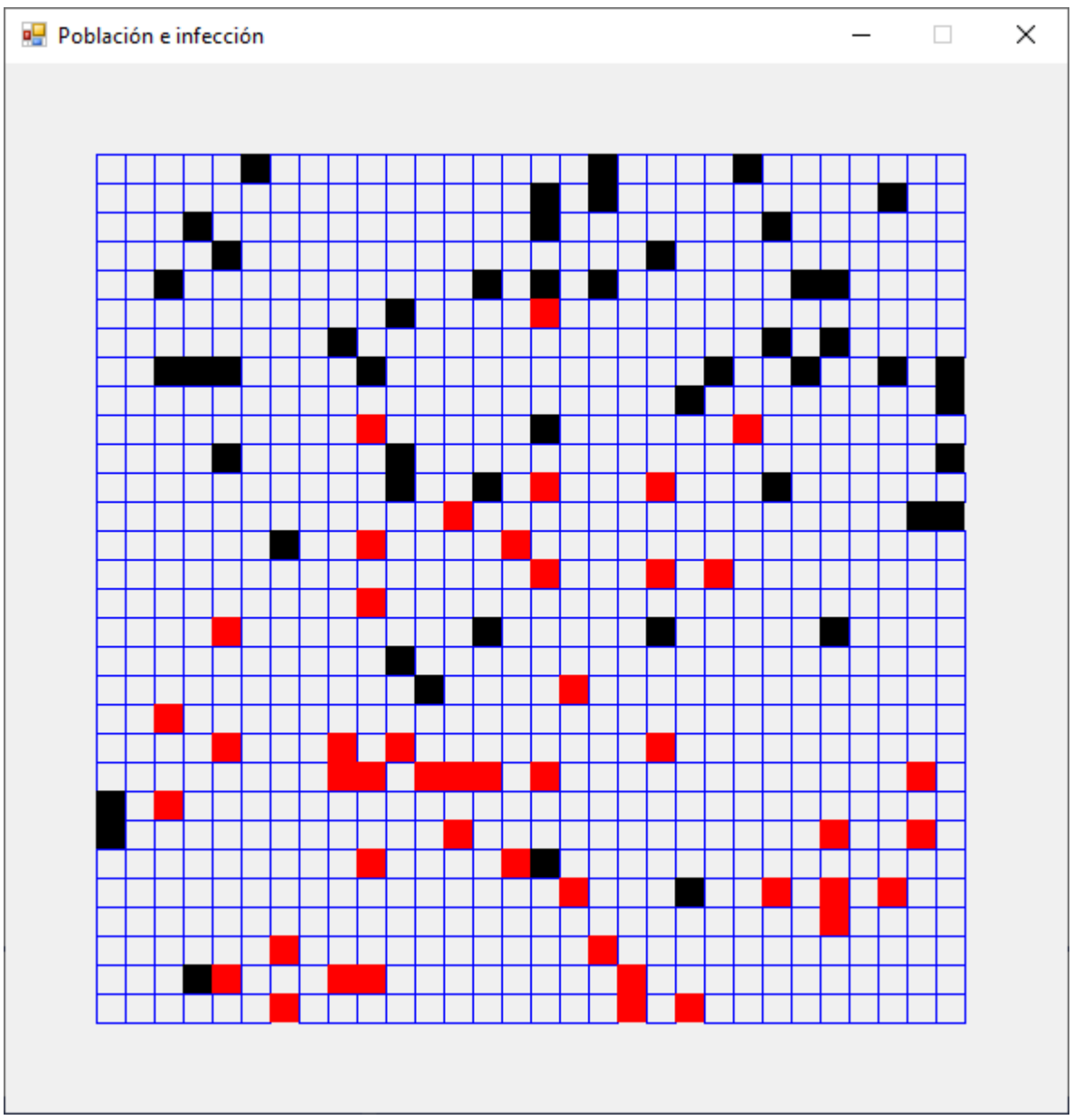


Ilustración 122: Infección bastante extendida

11. Simulación/03. Infección.zip/Individuo.cs

```
/* Cada individuo, su posición, sus movimientos y su estado
 * */

namespace Graficos {
    internal class Individuo {
        public int Fila, Columna, Estado;

        public Individuo(int Fila, int Columna, int Estado) {
            this.Fila = Fila;
            this.Columna = Columna;
            this.Estado = Estado;
        }

        public void Mover(int direccion, int NumFilas, int NumColumnas) {
            switch (direccion) {
                case 0: Fila--; Columna--; break;
                case 1: Fila--; break;
                case 2: Fila--; Columna++; break;
                case 3: Columna--; break;
                case 4: Columna++; break;
                case 5: Fila++; Columna--; break;
                case 6: Fila++; break;
                case 7: Fila++; Columna++; break;
            }

            if (Fila < 0) Fila = 0;
            if (Columna < 0) Columna = 0;
            if (Fila >= NumFilas) Fila = NumFilas-1;
            if (Columna >= NumColumnas) Columna = NumColumnas-1;
        }
    }
}
```

11. Simulación/03. Infección.zip/Form1.cs

```
//Simula como una infección se esparce entre la población
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace Graficos {
    public partial class Form1 : Form {
        //Constantes para calificar cada celda del tablero
        private const int VACIO = 0;
        private const int SANO = 1;
        private const int INFECTADO = 2;

        int[,] Tablero; //Dónde ocurre realmente la acción

        Random azar;

        //Tamaño de cada celda
        int tamanoFila, tamanoColumna, desplaza;

        //Población
        List<Individuo> Individuos;

        public Form1() {
            InitializeComponent();
            IniciarParametros();
        }

        public void IniciarParametros() {
            Tablero = new int[30, 30]; //Inicializa el tablero.
            azar = new Random();

            //Tamaño de cada celda
            tamanoFila = 500 / Tablero.GetLength(0);
            tamanoColumna = 500 / Tablero.GetLength(1);
            desplaza = 50;

            //Inicializa la población
            Individuos = new List<Individuo>();
```

```
        int NumIndividuos = 100;
        for (int cont = 1; cont <= NumIndividuos; cont++) {
            int Fila, Columna;
            do {
                Fila = azar.Next(Tablero.GetLength(0));
                Columna = azar.Next(Tablero.GetLength(1));
            } while (Tablero[Fila, Columna] != VACIO);
            Tablero[Fila, Columna] = SANO;
            Individuos.Add(new Individuo(Fila, Columna, SANO));
        }

        //Inicia con un individuo infectado
        Individuos[azar.Next(NumIndividuos)].Estado = INFECTADO;

        TimerAnimar.Start();
    }


    private void TimerAnimar_Tick(object sender, System.EventArgs e) {
        Logica();
        Refresh(); //Visual de la animación
    }

    //Lógica de la población
    private void Logica() {

        //Mueve los individuos
        for (int cont = 0; cont < Individuos.Count; cont++)
            Individuos[cont].Mover(azar.Next(8), Tablero.GetLength(0), Tablero.GetLength(1));

        //Limpia el tablero
        for (int Fila = 0; Fila < Tablero.GetLength(0); Fila++)
            for (int Columna = 0; Columna < Tablero.GetLength(1); Columna++)
                Tablero[Fila, Columna] = VACIO;

        //Refleja ese movimiento en el tablero
        for (int cont = 0; cont < Individuos.Count; cont++) {
            int Fila = Individuos[cont].Fila;
            int Columna = Individuos[cont].Columna;
            Tablero[Fila, Columna] = Individuos[cont].Estado;
        }

        //Chequea si un individuo infectado coincide con un individuo sano en la misma casilla para
infectarlo
        for (int cont = 0; cont < Individuos.Count; cont++) {
            if (Individuos[cont].Estado == INFECTADO) {
                for (int busca = 0; busca < Individuos.Count; busca++) {
                    if (Individuos[cont].Fila == Individuos[busca].Fila &&
                        Individuos[cont].Columna == Individuos[busca].Columna)
                        Individuos[busca].Estado = INFECTADO;
                }
            }
        }
    }

    private void Form1_Paint(object sender, PaintEventArgs e) {
        //Dibuja el arreglo bidimensional
        for (int Fila = 0; Fila < Tablero.GetLength(0); Fila++) {
            for (int Columna = 0; Columna < Tablero.GetLength(1); Columna++) {
                int UbicaFila = Fila * tamanoFila + desplaza;
                int UbicaColumna = Columna * tamanoColumna + desplaza;
                switch (Tablero[Fila, Columna]) {
                    case VACIO: e.Graphics.DrawRectangle(Pens.Blue, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                    case SANO: e.Graphics.FillRectangle(Brushes.Black, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                    case INFECTADO: e.Graphics.FillRectangle(Brushes.Red, UbicaFila, UbicaColumna,
tamanoFila, tamanoColumna); break;
                }
            }
        }
    }
}
```

Bibliografía

[1] Wikipedia, «GNU Lesser General Public License,» 2017. [En línea]. Available: https://es.wikipedia.org/wiki/GNU_Lesser_General_Public_License. [Último acceso: enero 2022].

[2] Microsoft, «Microsoft Windows,» enero 2022. [En línea]. Available: http://windows.microsoft.com/en-US/windows/home. [Último acceso: 26 enero 2022].

[3] Microsoft, «Visual Studio 2022,» enero 2022. [En línea]. Available: https://visualstudio.microsoft.com/es/vs/. [Último acceso: 26 enero 2022].

[4] Wikipedia, «C#,» 2022. [En línea]. Available: https://es.wikipedia.org/wiki/C_Sharp. [Último acceso: 26 enero 2022].

[5] Microsoft, «Crear y dibujar trazados,» enero 2022. [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/desktop/winforms/advanced/constructing-and-drawing-paths?view=netframeworkdesktop-4.8. [Último acceso: 26 enero 2022].

[6] Wikipedia, «Curva de Bézier,» enero 2022. [En línea]. Available: https://es.wikipedia.org/wiki/Curva_de_B%C3%A9zier. [Último acceso: 26 enero 2022].

[7] Net-informations.com, «C# Timer Control,» 2022. [En línea]. Available: http://csharp.net-informations.com/gui/timer-cs.htm. [Último acceso: 26 enero 2022].

[8] C# Corner, «Timer in C#,» 2022. [En línea]. Available: https://www.c-sharpcorner.com/article/timer-in-C-Sharp/. [Último acceso: 26 enero 2022].

[9] Microsoft, «How to: Reduce Graphics Flicker with Double Buffering for Forms and Controls,» [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/advanced/how-to-reduce-graphics-flicker-with-double-buffering-for-forms-and-controls?view=netframeworkdesktop-4.8. [Último acceso: 26 enero 2022].

[10] Microsoft, «Desktop Guide (Windows Forms .NET),» 2021. [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-6.0. [Último acceso: 26 enero 2022].

[11] Wikipedia, «Algoritmo de Bresenham,» 2022. [En línea]. Available: https://es.wikipedia.org/wiki/Algoritmo_de_Bresenham. [Último acceso: 26 enero 2022].

[12] Wikipedia, «Traslación (geometría),» 2022. [En línea]. Available: https://es.wikipedia.org/wiki/Traslaci%C3%B3n_(geometr%C3%ADa). [Último acceso: 26 enero 2022].

[13] Wikipedia, «Rotación (matemáticas),» 2022. [En línea]. Available: https://es.wikipedia.org/wiki/Rotaci%C3%B3n_(matem%C3%A1ticas). [Último acceso: 26 enero 2022].

[14] Wikipedia, «Proyección tridimensional,» 2022. [En línea]. Available: https://es.wikipedia.org/wiki/Proyecci%C3%B3n_tridimensional. [Último acceso: 26 enero 2022].

[15] I. Olmos, «Transformaciones Geométricas 3D,» 2021. [En línea]. Available: https://www.cs.buap.mx/~iolmos/graficacion/5_Transformaciones_geometricas_3D.pdf. [Último acceso: 26 enero 2022].

[16] Wikipedia, «Hidden-line removal,» 2022. [En línea]. Available: https://en.wikipedia.org/wiki/Hidden-line_removal. [Último acceso: 26 enero 2022].