Universidad de las Fuerzas Armadas ESPE

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

COMPUTACIÓN GRÁFICA

Tema: Área y perímetro de figuras geométricas.

Estudiante:

Luis Eduardo Sagnay Pilamunga

Docente:

Ing. Dario Javier Morales Caiza

07 de mayo del 2025

Informe de Desarrollo del Programa de Figuras Geométricas en C# .NET usando Windows Forms

1. Introducción

El desarrollo de este programa tiene el propósito de reforzar estos conceptos desde el enfoque computacional, aplicando los principios teoricos de la geometria, en donde, a través del lenguaje de programación C# y utilizando la tecnología de Windows Forms (.NET Framework), se construyó una aplicación capaz de calcular y representar gráficamente diversas figuras geométricas.

Este proyecto integra los principios de la programación orientada a objetos, proporcionando una solución práctica que permite al usuario calcular el área y perímetro de diferentes figuras planas. las cuales son cuadrados, rectángulos, triángulos, círculos, elipses, semicírculos, trapecios, trapezoides, rombos, romboides, deltoides y estrellas regulares. Además de visualizar su forma mediante gráficos generados dinámicamente.

2. Objetivo

Desarrollar una aplicación de escritorio utilizando C# y Windows Forms (.NET Framework), que permita al usuario calcular el área y perímetro de diversas figuras geométricas, además de mostrar su representación gráfica de dicha figura.

3. Funcionalidades implementadas

3.1. Interfaz de Usuario

- **Campos de entrada** (TextBox) para capturar dimensiones necesarias como base, altura, lados, radio, diagonales, etc.
- **Botones** para ejecutar los cálculos.
- **Etiquetas o campos de salida** (TextBox) para mostrar el **área** y el **perímetro** calculados.
- **Canvas (PictureBox)** para graficar la figura de forma dinámica.

3.2. Validación de Entradas

- Se evita que el usuario ingrese valores negativos o vacíos.

- Se verifican condiciones geométricas necesarias (por ejemplo, que las diagonales tengan sentido, que los lados formen una figura posible, etc.).
- Se muestra un mensaje de error en caso de datos inválidos, y se evita continuar con los cálculos o gráficos.

4. Estructura del Programa

4.1. Clases

El proyecto fue diseñado utilizando principios de programación orientada a objetos (OOP). Cada figura fue implementada como una clase específica que hereda de una clase base abstracta llamada “Shape”.

4.1.1. Propiedades protegidas:

Son variables miembro declaradas como protected, lo que significa que solo la propia clase y sus clases derivadas pueden acceder a estas directamente (no desde fuera de la jerarquía). Dentro del programa se declaró a:

- mArea, mPerimeter para almacenar resultados.
- SF como factor de escala para los gráficos.
- Graphics mGraph y Pen mPen para el dibujo.
- bool isValid para saber si los datos son válidos.

4.1.2. Clases derivadas para figuras geométricas:

Cada clase hereda de Shape e implementa sus propios cálculos y gráficos. A continuación, se describen:

- **Square**: Calcula área y perímetro de un cuadrado a partir de un solo lado.
- **Rectangle**: Trabaja con base y altura.
- **Triangle**: Se implementó usando base y altura para el área, y los tres lados para el perímetro.
- **Circle**: Requiere el radio y usa Math.PI para los cálculos.
- **Ellipse**: Se basa en los semiejes mayor y menor; se usó una aproximación para el perímetro.

- **Semicircle**: Se calcula como la mitad de un círculo, sumando el diámetro al perímetro curvo.

  Trapezoid : Se implementa con dos bases y dos lados no paralelos.
- **Trapezoide**: Similar al anterior pero sin lados paralelos.
- **Rhombus** (rombo): Calculado a partir de sus diagonales o lados.
- **Rhomboid** (romboide): Calcula área con base y altura, y perímetro con los lados.
- **Deltoid**: Usa las diagonales y lados desiguales para el cálculo.
- **Star**: Dibuja una estrella regular de 5 picos con geometría angular específica.

4.3. Formularios

Cada figura geométrica trabajada en el proyecto posee un formulario individual, diseñado específicamente para facilitar la interacción del usuario con los elementos necesarios para calcular y visualizar dicha figura. Estos formularios están construidos con controles de Windows Forms y tienen una disposición coherente, intuitiva y funcional.

4.3.1. Entradas necesarias (TextBox)

Cada formulario contiene uno o más cuadros de texto (TextBox) que permiten al usuario ingresar las dimensiones requeridas para los cálculos. Estas entradas están asociadas directamente con los parámetros utilizados en los métodos de cada figura. Además, se incluyen validaciones que deshabilitan el cálculo si los valores son negativos, vacíos, o superan límites definidos (por ejemplo, mayores a 17 unidades).

4.3.2. Etiquetas (Label) descriptivas

Acompañando a cada TextBox, se colocan etiquetas claras que indican qué dato debe ingresar el usuario. Esto garantiza una experiencia de usuario más comprensible y reduce errores por confusión de campos.

4.3.3. Botones

4.3.3.1. Botón Calcular

Ejecuta la lógica completa de validación, cálculo de área y perímetro, y la representación gráfica mediante los métodos de la clase correspondiente (ReadData(), CalculateArea(), CalculatePerimeter(), PlotShape()).

4.3.3.2. Botón Limpiar (Resetear)

Limpia los campos de entrada, borra los resultados y actualiza el PictureBox, devolviendo el formulario a su estado inicial. Utiliza el método InitializeData() que cada clase figura hereda de Shape.

4.3.3.3. Botón Regresar

Cierra el formulario actual y devuelve al usuario al menú principal (FrmMain). Internamente, usa el método CloseForm() de la clase Shape.

4.3.4. PictureBox para dibujo gráfico

Cada formulario cuenta con un control PictureBox, donde se dibuja la figura correspondiente. La figura es representada utilizando los métodos de la clase Graphics (DrawRectangle, DrawPolygon, DrawEllipse, etc.) con coordenadas calculadas en tiempo de ejecución para asegurar que se dibuje de forma proporcional y centrada.

4.3.5. Campos de salida deshabilitados (TextBox no editables)

Los resultados del área y el perímetro se muestran en campos que están deshabilitados (txtOutput.ReadOnly = true), impidiendo que el usuario los modifique. Estos campos se actualizan automáticamente con el resultado de los cálculos al presionar el botón de cálculo.

4.3.6. Instancia de la clase correspondiente a la figura

Dentro de cada formulario, se crea una instancia privada de la clase correspondiente (por ejemplo: Square cuadrado = new Square();). Esta instancia es la encargada de manejar toda la lógica de validación, cálculo y dibujo. Al usar polimorfismo, todos los formularios interactúan con sus figuras de forma estandarizada a través de los métodos definidos en la clase base Shape.

4.4. Métodos

4.4.1. Métodos Abstractos

En la arquitectura del proyecto, se emplean métodos abstractos definidos en la clase base Shape. Un método abstracto declara su firma (nombre, parámetros y tipo de retorno) pero no

contiene implementación, lo que obliga a todas las clases derivadas a proporcionar su propia versión mediante el modificador override. Esto garantiza que cada figura implemente su lógica específica para lectura, cálculo y graficado, respetando el principio de polimorfismo.

- **ReadData(params TextBox[] inputs):** Se encarga de leer y validar las dimensiones introducidas por el usuario desde los campos de entrada (TextBox). Cada clase concreta adapta este método a los parámetros particulares de la figura (lados, radios, diagonales, etc.). Incluye validaciones como no aceptar valores vacíos, negativos o fuera de un rango lógico (por ejemplo, mayores a 17).

- **CalculateArea():** Implementado por cada figura para calcular su área con base en su fórmula correspondiente.

- **CalculatePerimeter():** Implementado para calcular el perímetro según la definición geométrica de la figura.

- **PlotShape(PictureBox canvas):** Método clave para graficar la figura dentro de un PictureBox. Cada figura define cómo calcular los puntos o coordenadas necesarios y dibuja sobre el lienzo usando objetos Graphics y Pen. Esto permite representar visualmente la figura con precisión, escalada y centrada en el formulario.

4.4.2. Métodos Virtuales

Los métodos virtuales se definen en la clase base Shape con una implementación por defecto. Las clases hijas pueden usarlos tal cual o sobrescribirlos si requieren un comportamiento distinto. Estos métodos encapsulan tareas comunes que se repiten en todas las figuras, promoviendo la reutilización de código.

- **InitializeData(TextBox txtPerimeter, TextBox txtArea, PictureBox picCanvas):** Restaura el estado del formulario a su configuración inicial. Limpia los resultados previos, borra el área de dibujo del PictureBox y reinicia los valores internos del objeto figura. Este método se llama cuando el usuario presiona el botón “Limpiar”.

- **PrintData(TextBox txtPerimeter, TextBox txtArea):** Se encarga de mostrar el área y el perímetro calculados en los campos de salida del formulario. Utiliza los atributos mArea y mPerimeter de la clase base.

Cada figura concreta como Square, Triangle, Circle, Deltoid, Rectangle, Rhombus, etc., implementa estos métodos de manera específica, respetando su lógica matemática y gráfica. Esta estructura orientada a objetos asegura una implementación coherente, robusta y escalable del sistema, donde agregar nuevas figuras requiere un mínimo de esfuerzo y sin alterar la lógica general del programa.

5. Gráficos de las figuras

Cada clase implementa el método: “public override void PlotShape(PictureBox picCanvas)”

Dentro de este método, se utilizan herramientas del espacio de nombres System.Drawing para representar visualmente las figuras en el PictureBox. En donde fue necesario utilizar:

5.1. Point y PointF

Estos tipos de datos representan coordenadas en un sistema bidimensional (X, Y).

Point: Utiliza valores enteros para las coordenadas.

PointF: Utiliza valores de punto flotante (float), lo que permite una mayor precisión, ideal cuando se hacen cálculos con decimales (por ejemplo, al usar funciones trigonométricas).

Se utilizan para definir los vértices de figuras como triángulos, deltoides, estrellas, entre otras figuras.

5.2. DrawPolygon(Pen, PointF[])

Este método de Graphics dibuja un polígono cerrado conectando en orden una serie de puntos dados (último punto se conecta con el primero automáticamente).

Se utilizan para figuras con lados definidos como triángulo, rombo, romboide, estrella, deltoide. Además para estas figuras fue necesario realizar cálculos y fórmulas adicionales para poder dibujar correctamente.

5.3. DrawRectangle(Pen, float x, float y, float width, float height)

Dibuja un rectángulo en pantalla a partir de una posición de inicio (X, Y) y un tamaño (ancho y alto).

Este método se utilizó para representar cuadrados y rectángulos.

5.4. DrawEllipse(Pen, float x, float y, float width, float height)

Dibuja una elipse dentro del rectángulo definido por los parámetros. Si el ancho y alto son iguales, se dibuja un círculo.

Se utilizó para dibujar las figuras de círculo y elipse.

5.5. DrawArc(Pen, float x, float y, float width, float height, float startAngle, float sweepAngle)

Dibuja un arco de una elipse, desde un ángulo inicial (startAngle) y en un ángulo de barrido (sweepAngle).

Se utilizó este método para poder dibujar al semicírculo, junto con la clase DrawLine.

6. Ejecución

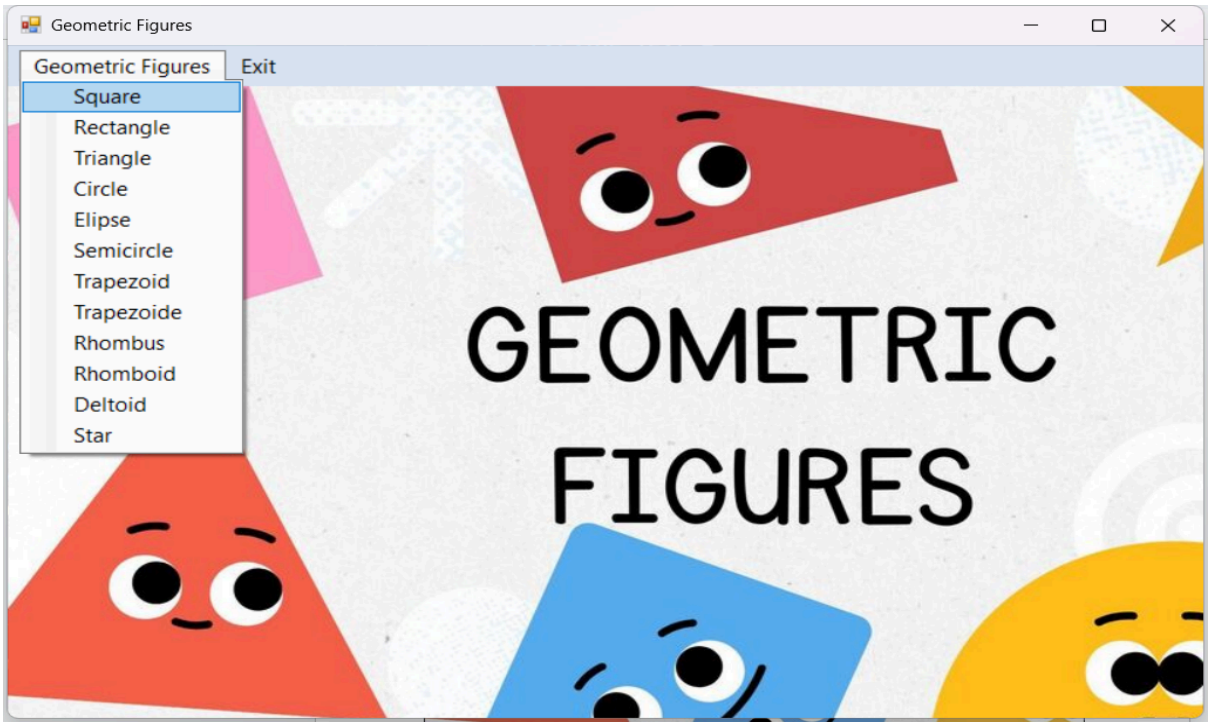

Figura 1. *Menú Principal*

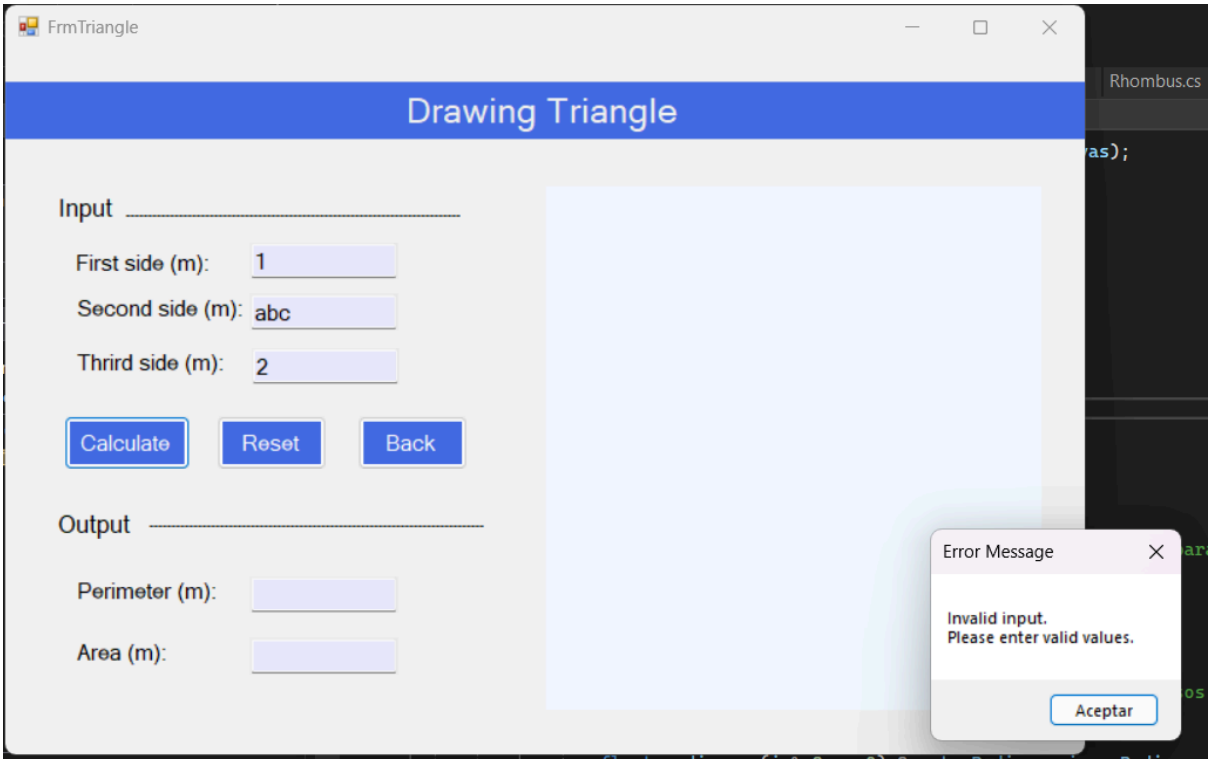

Figura 2. *Validación de entrada de datos no numéricos*

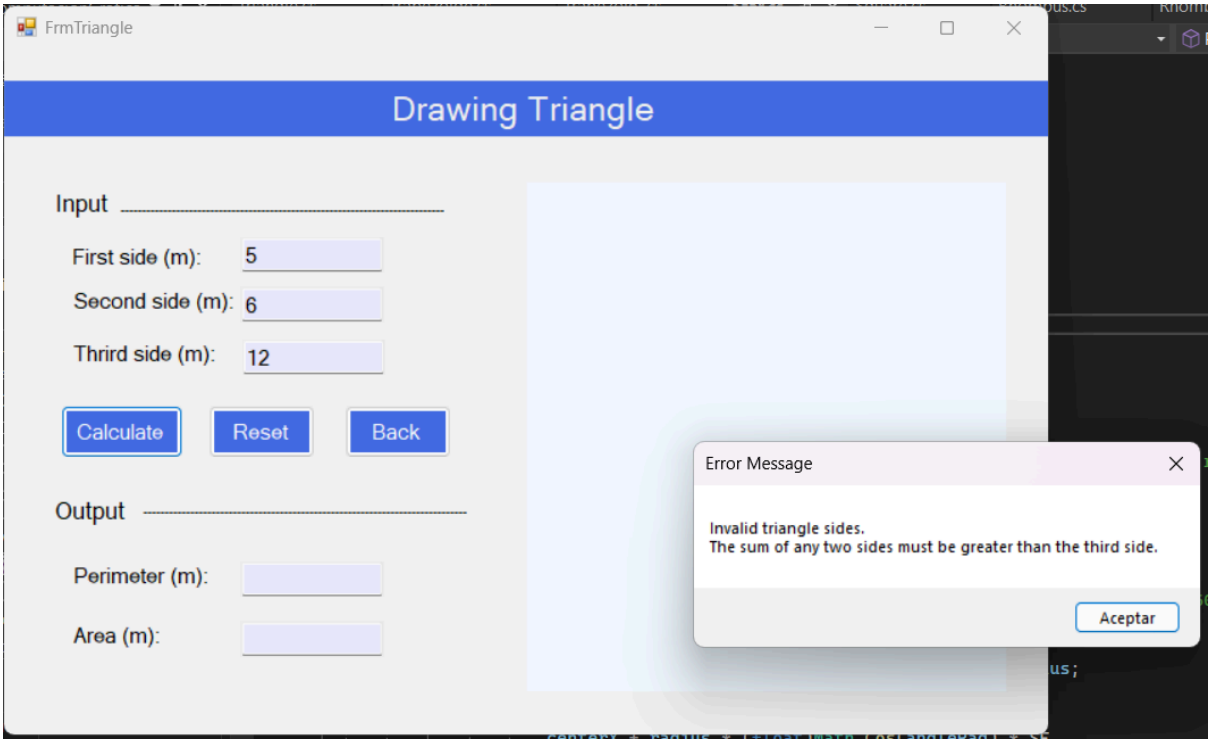

Figura 3. *Validación de consistencia de entradas numéricas*

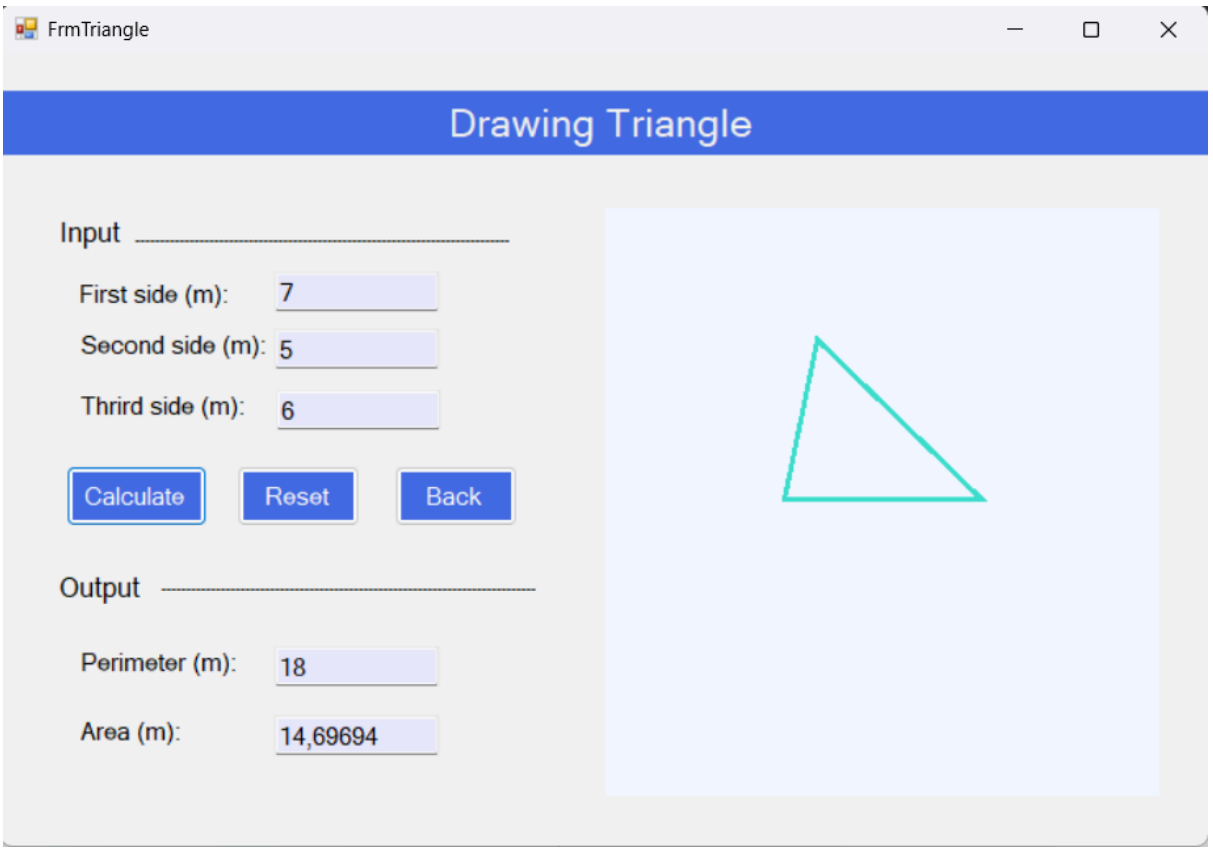

Figura 4. *Entradas, cálculos y gráfico correcto.*

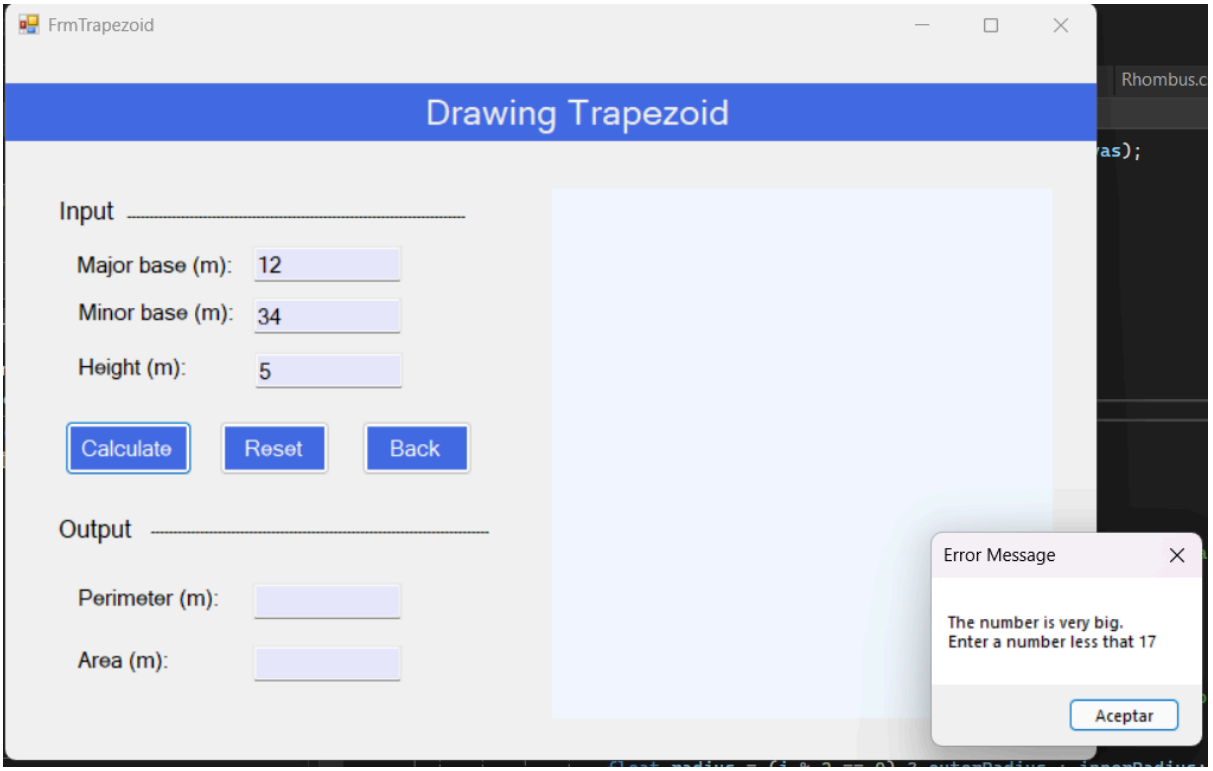

Figura 5. *Validación de entradas muy grandes*

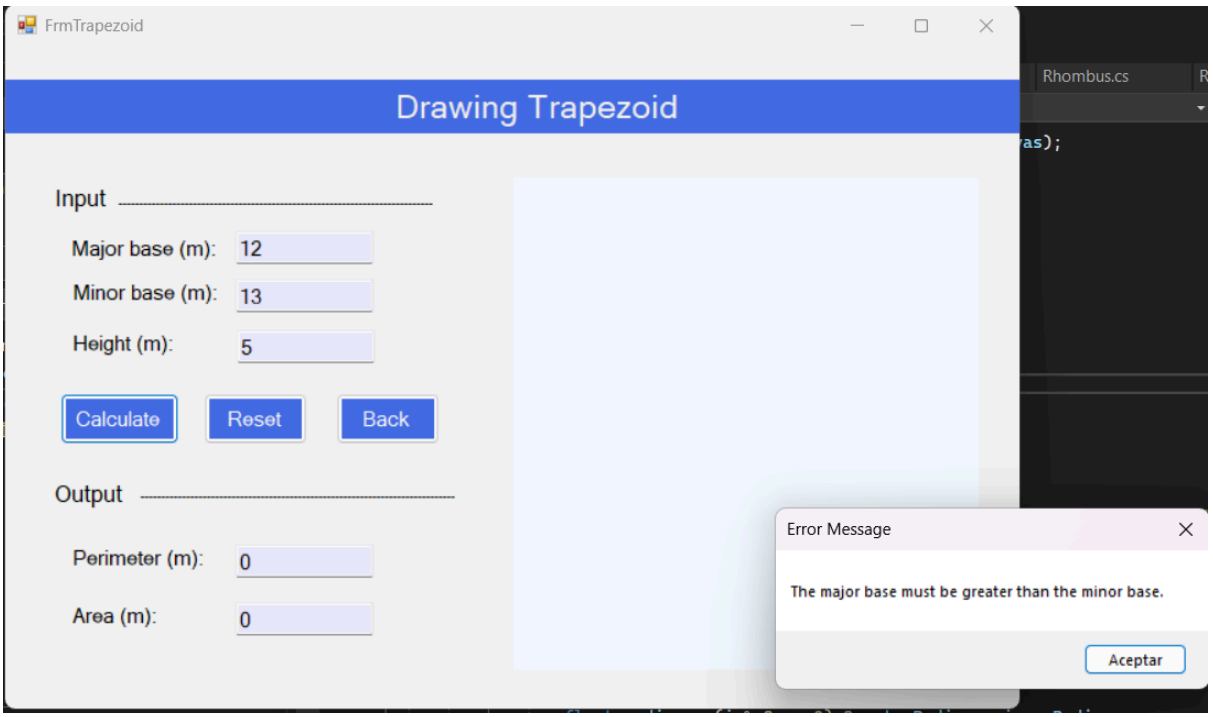

Figura 6. *Validación de consistencia de entradas numéricas*

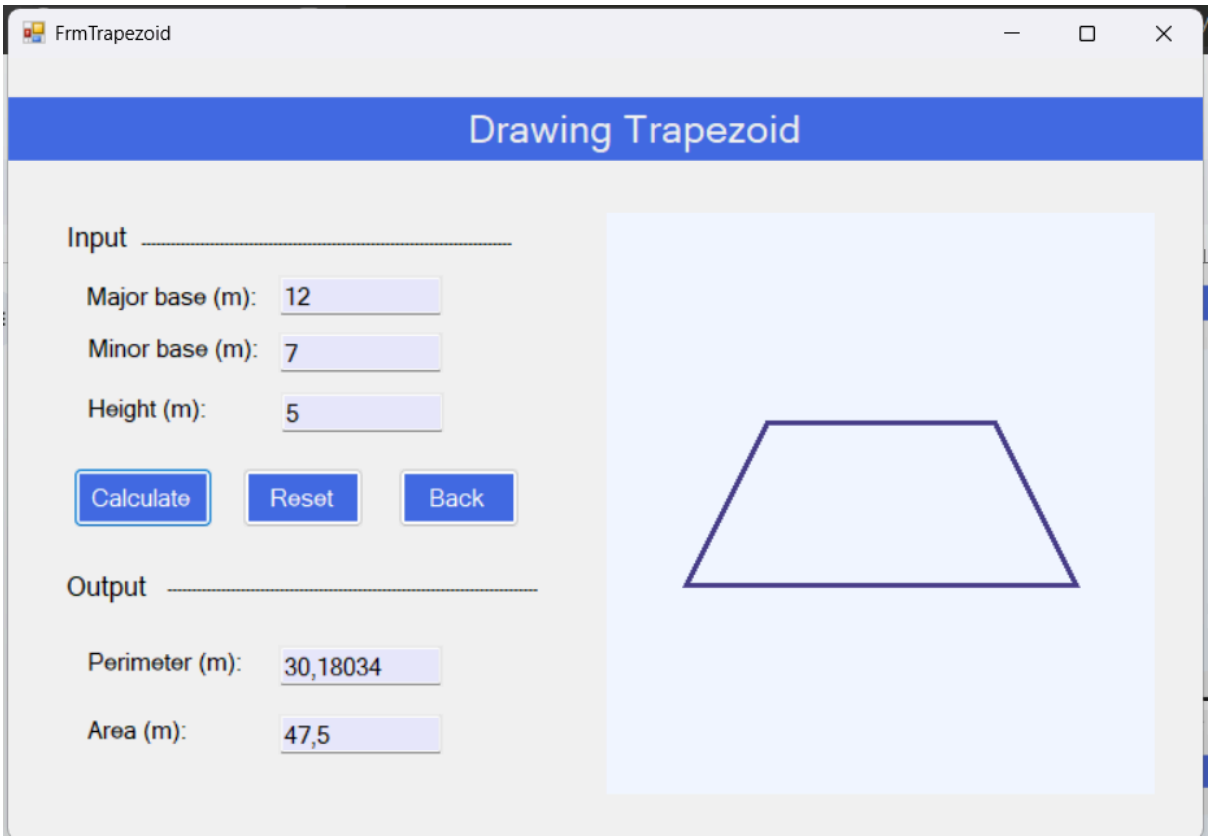

Figura 7. *Entradas, cálculos y gráfico correcto.*

7. Conclusiones

El desarrollo de este programa permite aplicar los conceptos fundamentales de Computación Gráfica en un entorno práctico utilizando C# y Windows Forms. A lo largo de este programa se integraron conocimientos tanto de programación orientada a objetos como de geometría computacional, destacando la construcción y manipulación de objetos gráficos mediante coordenadas, uso de Graphics, y métodos como DrawPolygon, DrawEllipse, entre otros.

Además, se consolidó el entendimiento de cómo representar figuras geométricas mediante el cálculo de sus vértices, aplicando fórmulas geométricas y trigonométricas para generar una visualización precisa de estas figuras en 2D. Cada figura requirió un análisis particular sobre cómo calcular sus dimensiones y cómo transformarlas adecuadamente a un sistema de coordenadas gráfico usando escalas (SF), centrado en pantalla y proporción. Así como la validación de datos de entrada, asegurando que los valores ingresados por el usuario sean coherentes y permitan la correcta ejecución de los métodos de dibujo y cálculo.

8. Recomendaciones

1. Modularizar aún más el código, empaquetar y separar código y cálculos matemáticos para mejorar el mantenimiento y la reutilización de funciones.
2. Incluir una función de escala automática basada en el tamaño de la figura y del PictureBox, para evitar limitar los valores a un rango fijo para que se muestre de manera correcta la figura.
3. Agregar una opción para guardar las figuras como imágenes, lo que puede ser útil en contextos educativos o de documentación técnica.
4. Extender el programa para que pueda realizar transformaciones gráficas básicas como traslaciones, rotaciones o escalados, con el fin de afianzar los conceptos avanzados de la asignatura de Computación Gráfica.

9. Anexos

Link de Github:
https://github.com/luis-sagx/ComputacionGrafica/tree/main/GeometricFigures