



Windows API



Reporte de proyecto final de curso:

SOLUCIÓN DE 2 ECUACIONES LINEALES CON EL MÉTODO GRÁFICO

Profesor: Dr. Felipe Rolando Menchaca García

Alumno: Atlitec Sarabia Diego Alejandro

Materia: Programación Orientada a Objetos (POO)

Grupo: 2CM12



ÍNDICE.

- | | | | |
|-----------|------------------------|------------|--|
| 03 | Introducción | 29 | Estructura del menú |
| 05 | Objetivo | 34 | Gráfica de una línea recta |
| 06 | Biblioteca graphics.h | 38 | Método gráfico para resolver un sistema de ecuaciones lineales 2x2 |
| 08 | Funciones | 40 | Código |
| 11 | Plano cartesiano | 61 | Pruebas |
| 18 | Crear plano cartesiano | 120 | Conclusiones |



Introducción.

El presente proyecto muestra un programa hecho en C++ utilizando la API de Windows. Este proyecto fue propuesto por nuestro profesor de la materia "Programación Orientada a Objetos (POO)" y consiste en desarrollar un solucionador de un sistema de ecuaciones 2x2 empleando el método gráfico. Utilizando los conocimientos adquiridos a lo largo del curso, se implementaron diversas funcionalidades, como la representación gráfica del plano cartesiano, puntos y rectas, así como la intersección de estas últimas para encontrar soluciones gráficas a los sistemas de ecuaciones lineales. Este reporte detalla la estructura del programa, los métodos utilizados y las pruebas correspondientes.



Windows API

¿Qué es la API de Windows?

API son las siglas de Interfaz de programación de aplicaciones y en Windows, representan un conjunto de funcionalidades que proporcionan servicios de bajo nivel. Existen varias versiones y fueron y siguen siendo utilizadas principalmente por programadores C/C++ y hasta cierto punto por quienes desarrollan en .NET (pues este último incorpora funciones de más alto nivel para la mayoría de cosas).

Su uso se ha descontinuado con el paso del tiempo por la misma llegada de .NET, ya que resultan muy complejas de implementar.

Tiene funciones para:

- Tratamiento de archivos.
- Identificación de dispositivos.
- Creación de interfaces.
- Entre otros.

Objetivo.



Aplicar los conocimientos adquiridos en la materia de Programación Orientada a Objetos para crear un programa solucionador de sistemas de ecuaciones 2x2 utilizando el método gráfico, fomentando así el aprendizaje continuo y la mejora de habilidades en C++ y en el uso de la API de Windows, además de aprender a utilizar la biblioteca graphics en este lenguaje de programación.



¿Qué es graphics.h?



El pilar fundamental del programa es la biblioteca **graphics**, cuya utilización es el núcleo de esta implementación. Hoy en día, es poco usual encontrar información acerca de librerías tan antiguas como ésta, y mucho más cuando se enfoca en algo tan particular como la creación de gráficas de funciones matemáticas.

La librería **graphics** es aquella perteneciente a Borland que contiene múltiples funciones para generar ventanas y gráficas al estilo ms-dos.



Requisitos para usar graphics.h



- Tener instalado el IDE DEVC++.
- Poseer conocimientos básicos del lenguaje c/c++.
- Poseer conocimientos básicos de álgebra para el entendimiento correcto de las funciones.



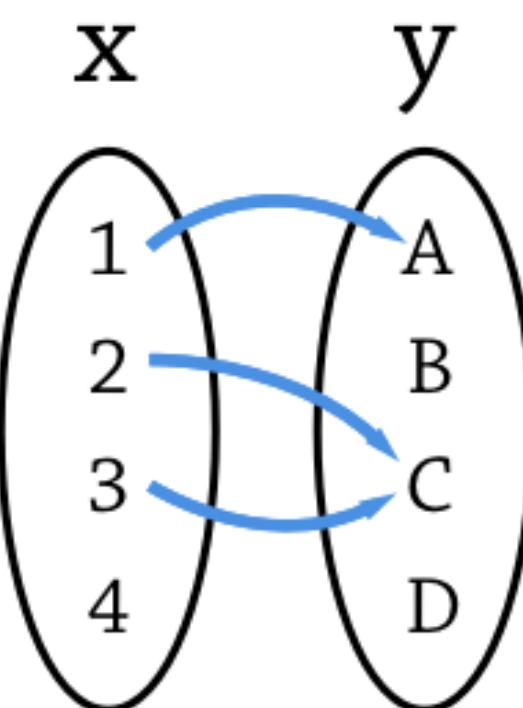
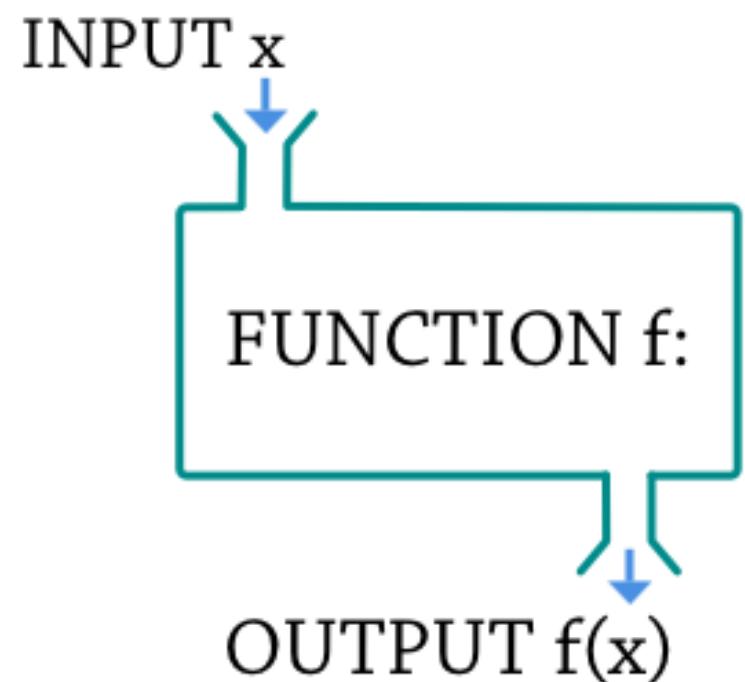
¿Qué es una función en C++?



En C++, un programa se organiza en funciones, que son bloques de código diseñados para realizar tareas específicas. La función `main()` es el punto de entrada y salida del programa, donde comienza y termina la ejecución. Las bibliotecas estándar de C++ proporcionan funciones útiles como `sqrt()` para calcular raíces cuadradas y `toupper()` para convertir caracteres a mayúsculas, contenidas en `<cmath>` y `<iomanip>` respectivamente. Los desarrolladores pueden crear funciones personalizadas, como `mi_sqrt()`, para ejecutar operaciones definidas por el usuario. Estas funciones se asemejan a las matemáticas, donde se establece una relación entre entradas y salidas; por ejemplo, una función que a cada número entero le añade cinco.



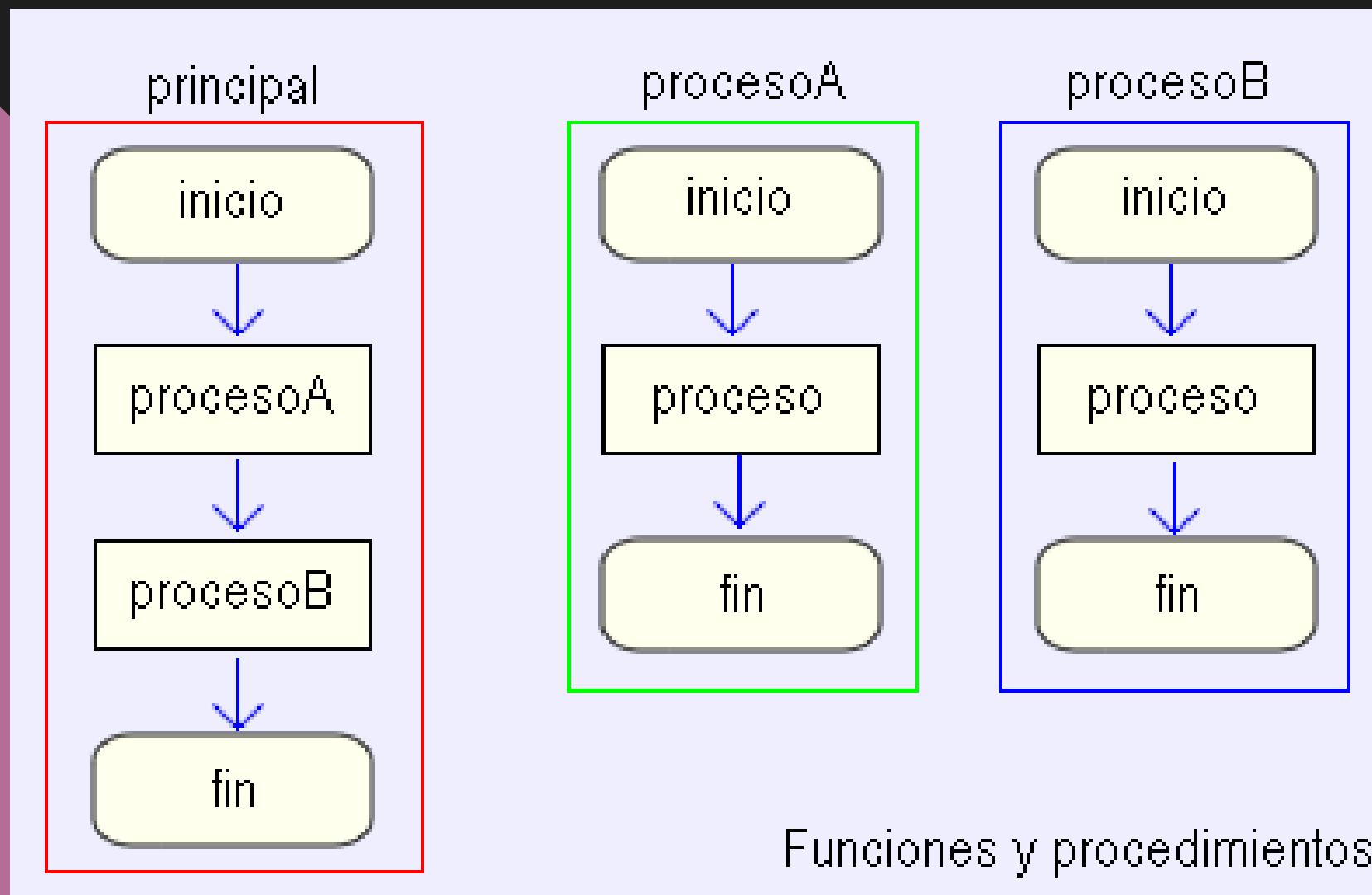
Declaración y Definición de Funciones en C++



La declaración de una función en C++ informa al compilador sobre su firma, incluyendo el tipo de retorno, el nombre y los tipos de parámetros. Por ejemplo, `unsigned int f(unsigned int x);` declara que `f` acepta un `unsigned int` y devuelve otro. La definición de la función proporciona el cuerpo de la función, que es el conjunto de instrucciones que se ejecutan al llamarla, como en `unsigned int f(unsigned int x) { return x + 5; }`, que define `f` para devolver el argumento `x` incrementado en cinco. La invocación de una función se realiza por su nombre y pasando argumentos concretos, como `f(4)`, que llama a `f` con el valor `4`.



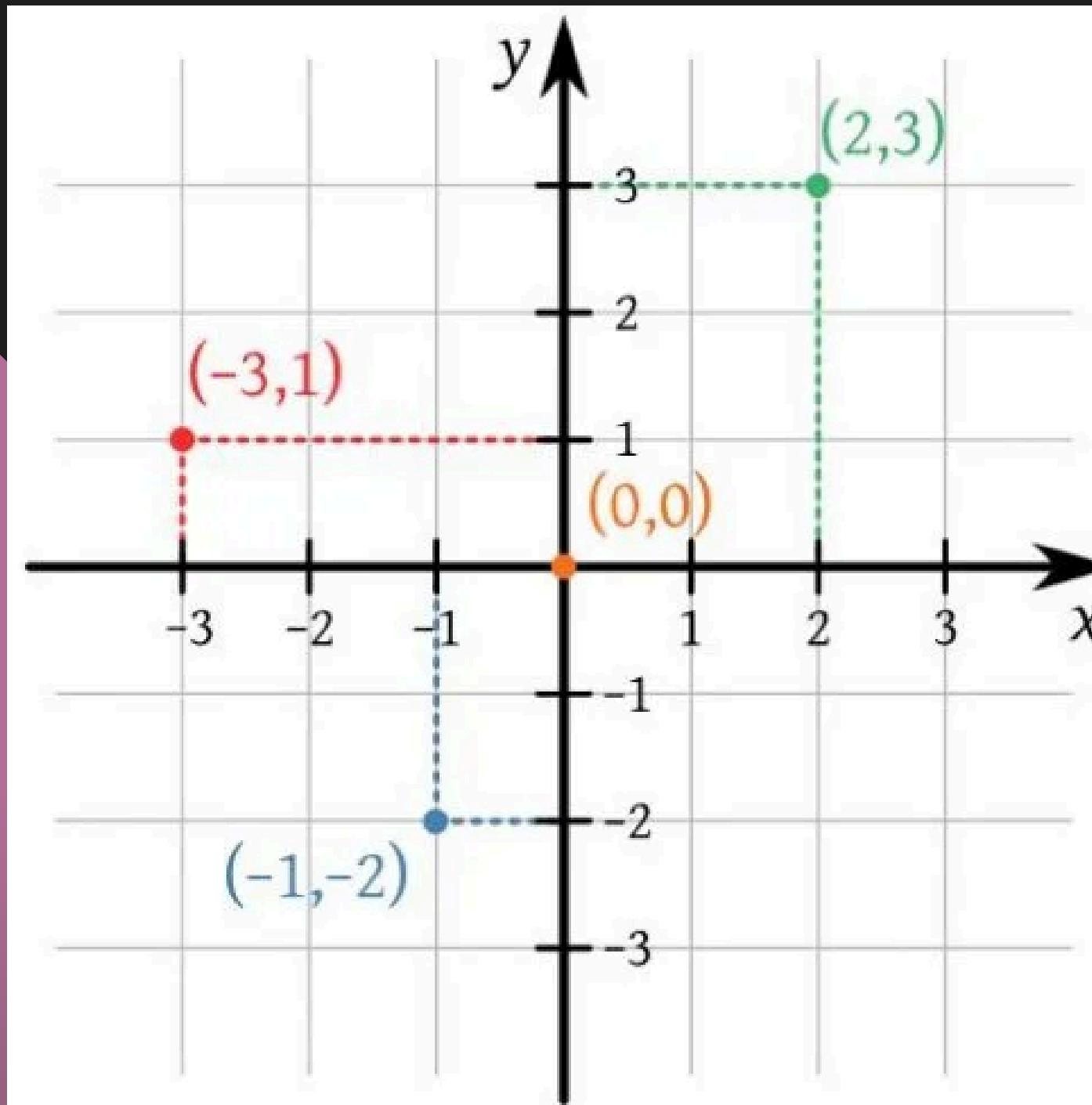
Funciones a usar en este programa



- **initwindow(int width, int height, ...):** Inicia la ventana de A x B tamaño.
- **closegraph():** Cierra la ventana inciada previamente.
- **putpixel(int x, int y, int color):** Pinta un pixel en punto(x,y)
- **line(int x1, int y1, int x2, int y2):** Pinta una línea.
- **delay(int msec):** Espera un tiempo para continuar con la ejecución del programa.
- **outtextxy(int x, int y, char *textstring):** Pinta un string en la ventana en el punto(x,y).
- **cleardevice():** Limpia el contenido de la ventana.
- **settextstyle(int font, int direction, int charszie):** Da un stilo al texto de la ventana.
- **setcolor(int color):** Define un nuevo color a usar.



Plano cartesiano



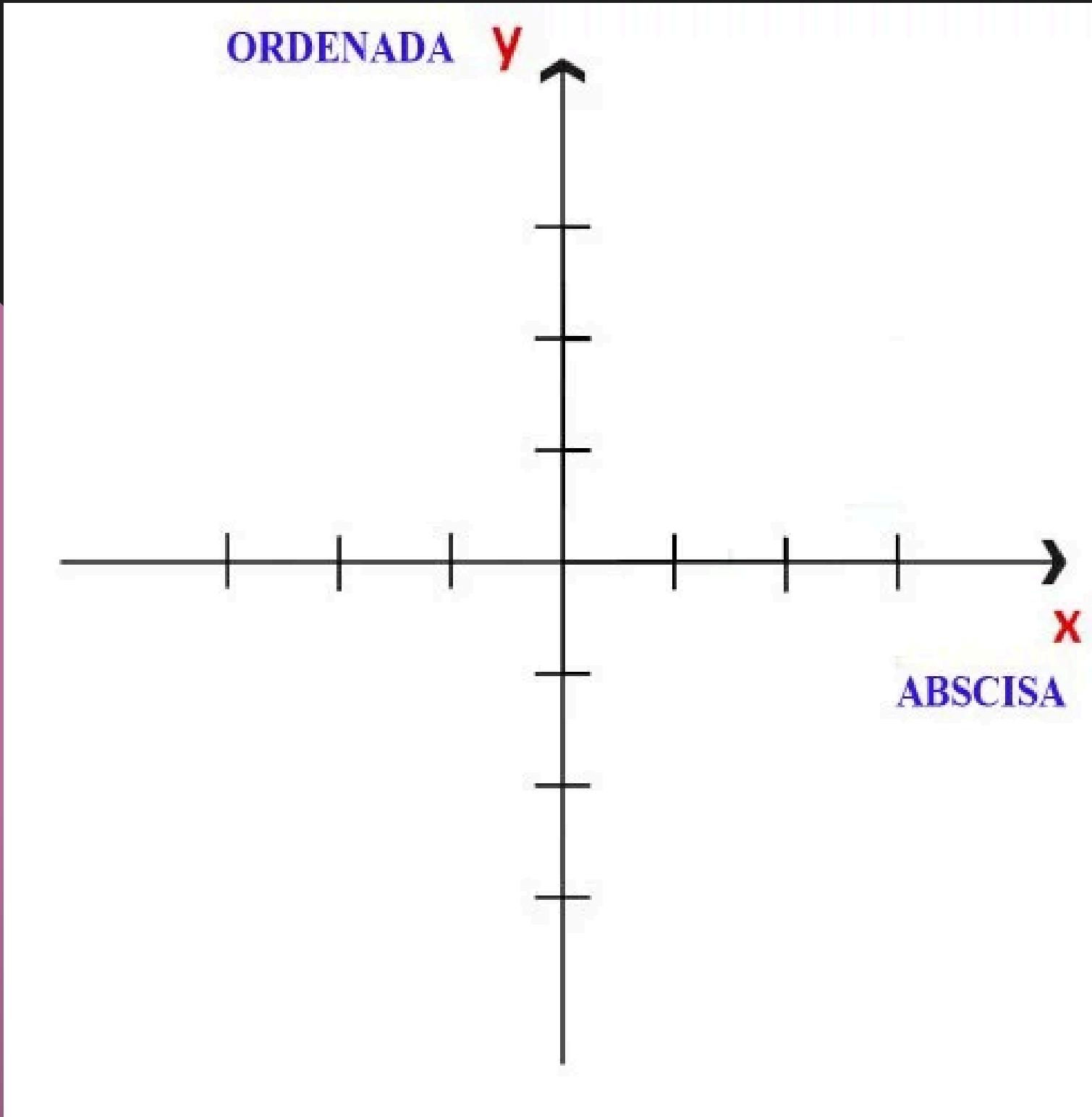
Se conoce como **plano cartesiano**, **coordenadas cartesianas** o **sistema cartesiano**, a dos rectas numéricas perpendiculares, una horizontal y otra vertical, que se cortan en un punto llamado **origen o punto cero**.

La finalidad del **plano cartesiano** es describir la posición o ubicación de un punto en el **plano**, la cual está representada por el **sistema de coordenadas**.

El **plano cartesiano** también sirve para analizar matemáticamente figuras geométricas como la **parábola**, la **hipérbola**, la **línea recta**, la **circunferencia** y la **elipse**, las cuales forman parte de la **geometría analítica**.



Elementos del plano cartesiano



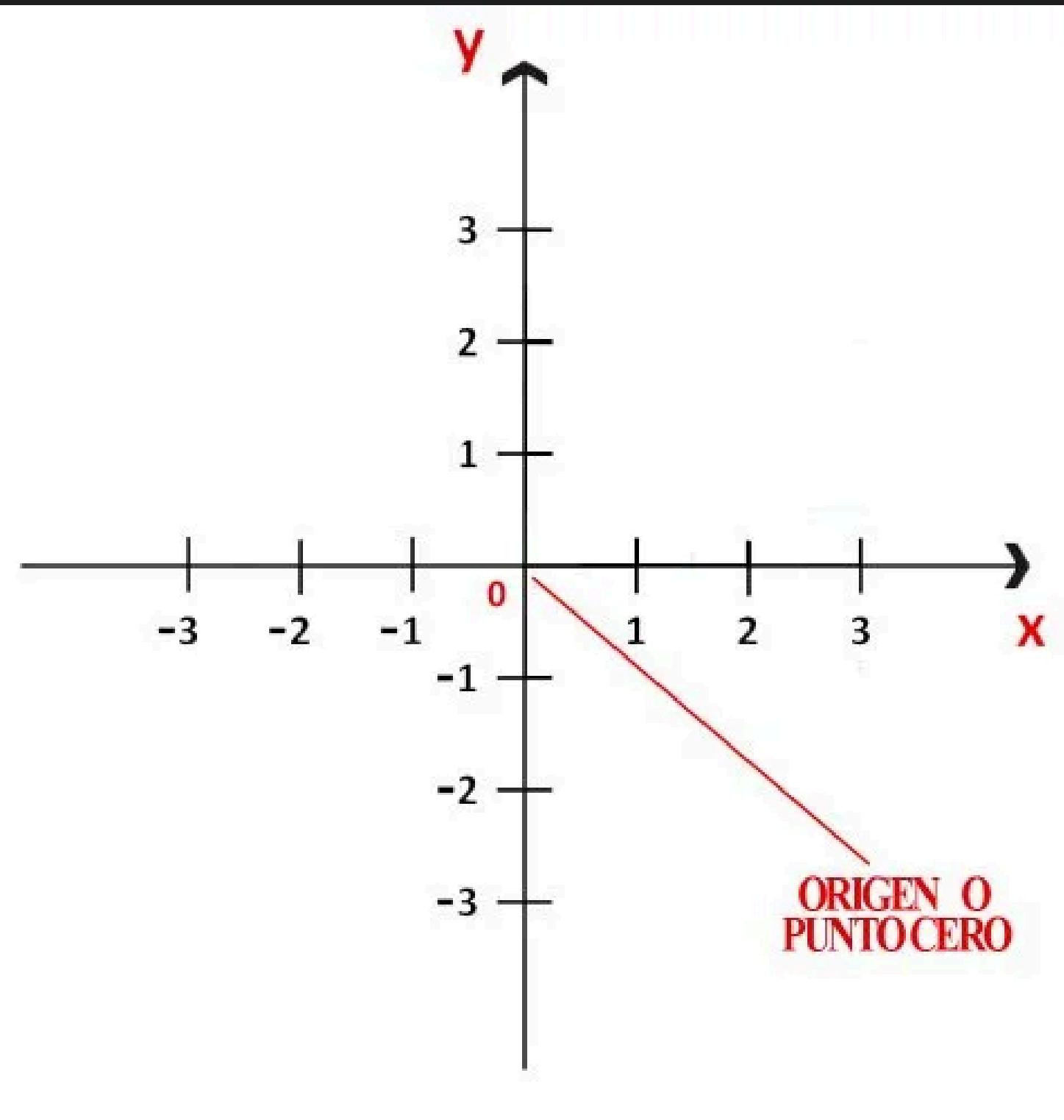
Los elementos y características que conforman el **plano cartesiano** son los **ejes coordenados**, el **origen**, los **cuadrantes** y las **coordenadas**.

Se llaman **ejes coordenados** a las dos rectas perpendiculares que se conectan en un punto del **plano**. Estas rectas reciben el **nombre de abscisa y ordenada**.

- **Abscisa:** el eje de las **abscisas** está dispuesto de manera **horizontal** y se identifica con la letra "**x**".
- **Ordenada:** el eje de las **ordenadas** está **orientado verticalmente** y se representa con la letra "**y**".



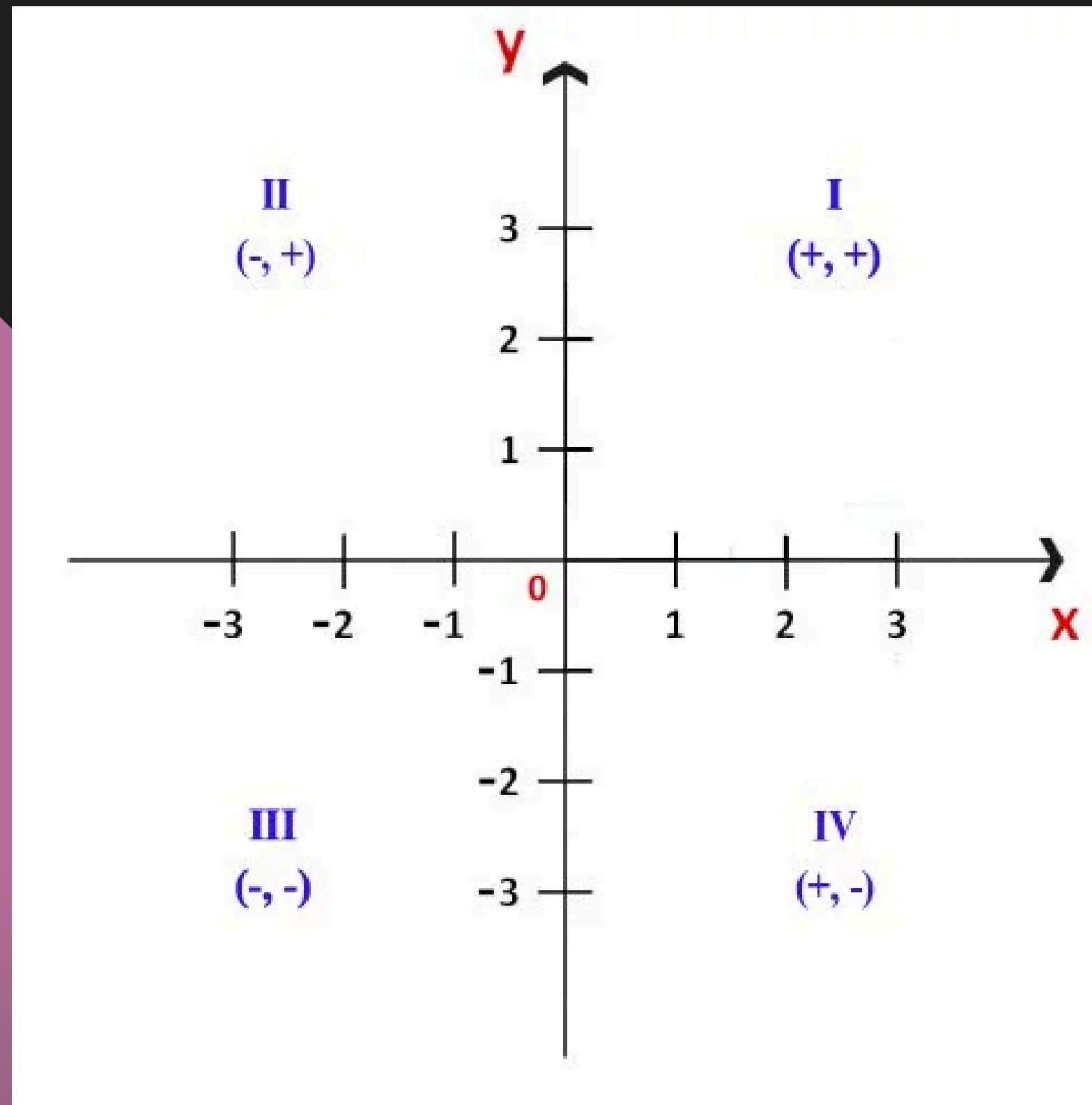
Origen o punto 0



Se llama **origen** al punto en el que se intersecan los ejes "x" e "y", punto al cual se le asigna el valor de **cero** (0). Por ese motivo, también se conoce como **punto cero** (**punto 0**). Cada eje representa una escala numérica que será **positiva** o **negativa** de acuerdo a su dirección respecto del origen. Así, respecto del origen o punto 0, el **segmento derecho** del eje "x" es **positivo**, mientras que el **izquierdo** es **negativo**. Consecuentemente, el **segmento ascendente** del eje "y" es **positivo**, mientras que el **segmento descendente** es **negativo**.



Cuadrantes del plano cartesiano



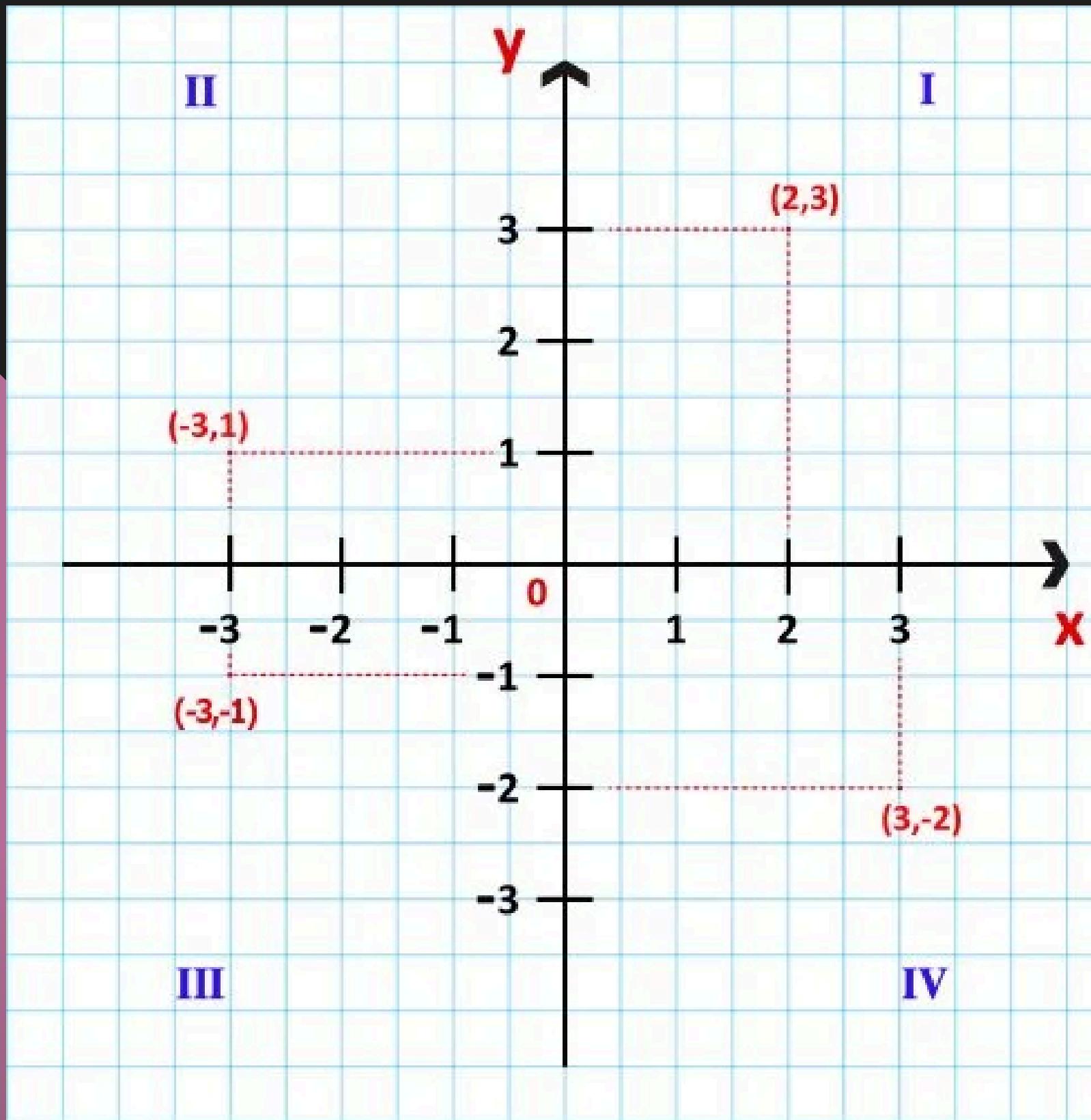
Se llaman cuadrantes a las cuatro áreas que se forman por la unión de las dos rectas perpendiculares. Los puntos del plano se describen dentro de estos cuadrantes.

Los cuadrantes se enumeran tradicionalmente con números romanos: I, II, III y IV.

- Cuadrante I: la abscisa y la ordenada son positivas.
- Cuadrante II: la abscisa es negativa y la ordenada positiva.
- Cuadrante III: tanto la abscisa como la ordenada son negativas.
- Cuadrante IV: la abscisa es positiva y la ordenada negativa.



Coordenadas del plano cartesiano



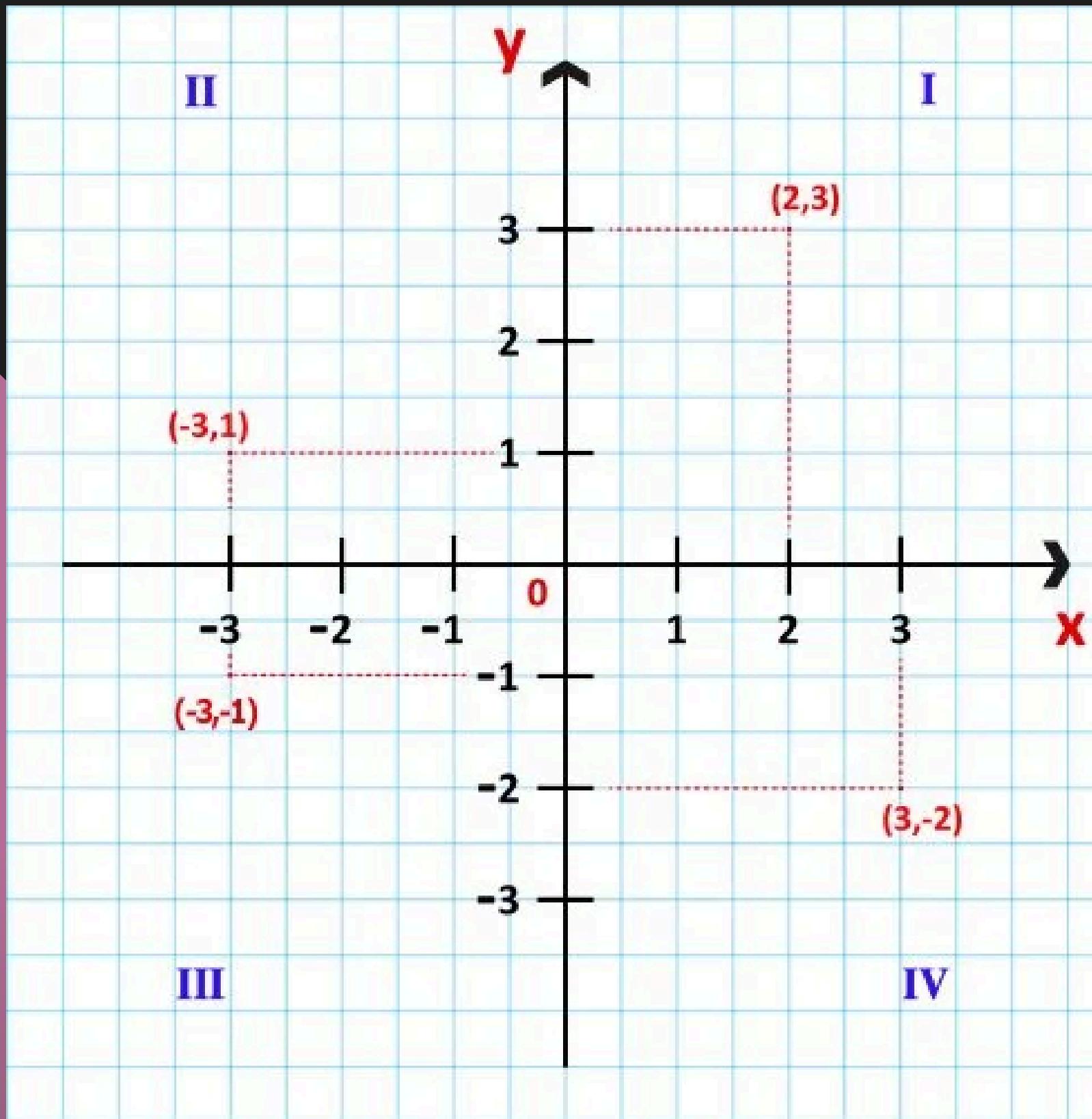
Las **coordenadas** son los números que nos dan la ubicación del punto en el plano. Las coordenadas se forman asignando un determinado valor al eje "x" y otro valor al eje "y". Esto se representa de la siguiente manera:

$P(x, y)$, donde:

- P = punto en el plano;
- x = eje de la abscisa (horizontal);
- y = eje de la ordenada (vertical).



Coordenadas del plano cartesiano



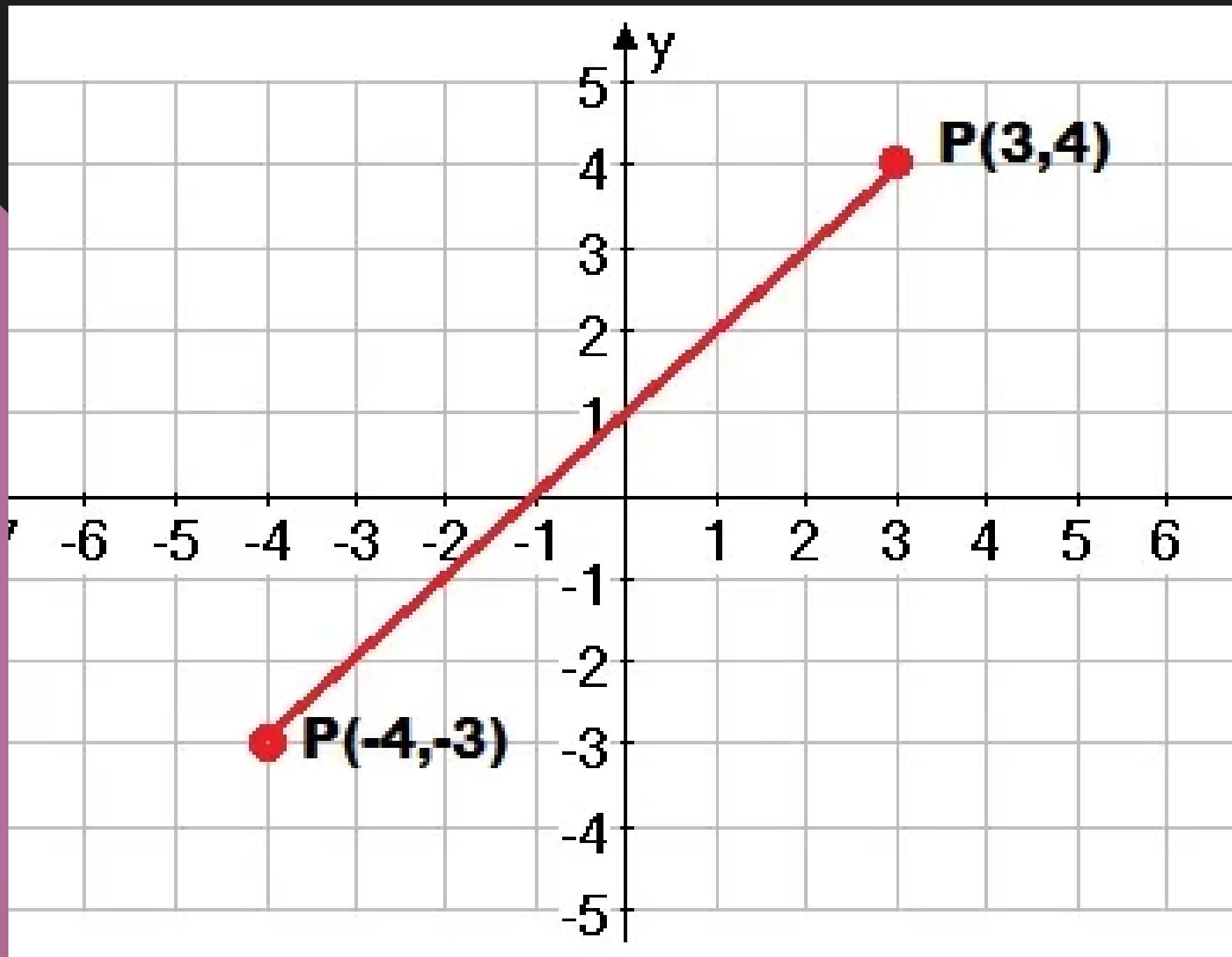
En este ejemplo, las **coordenadas** de los puntos en cada cuadrante son:

- cuadrante I, P (2, 3);
- cuadrante II, P (-3, 1);
- cuadrante III, P (-3, -1) y
- cuadrante IV, P (3, -2).

Si lo que queremos es saber la ubicación de un punto a partir de unas **coordenadas** previamente asignadas, entonces trazamos una línea perpendicular desde el número indicado de la abscisa, y otra desde el número de la ordenada. La intersección o cruce de ambas proyecciones nos da la ubicación espacial del punto.



Coordenadas del plano cartesiano



En este ejemplo, $P(3,4)$ nos da la ubicación precisa del punto en el cuadrante I del plano. El 3 pertenece al eje de las abscisas y el 4 (segmento derecho) al eje de las ordenadas (segmento ascendente).

$P(-3,-4)$ nos da la ubicación específica del punto en el cuadrante III del plano. El -3 pertenece al eje de las abscisas (segmento izquierdo) y el -4 al eje de las ordenadas (segmento descendente).



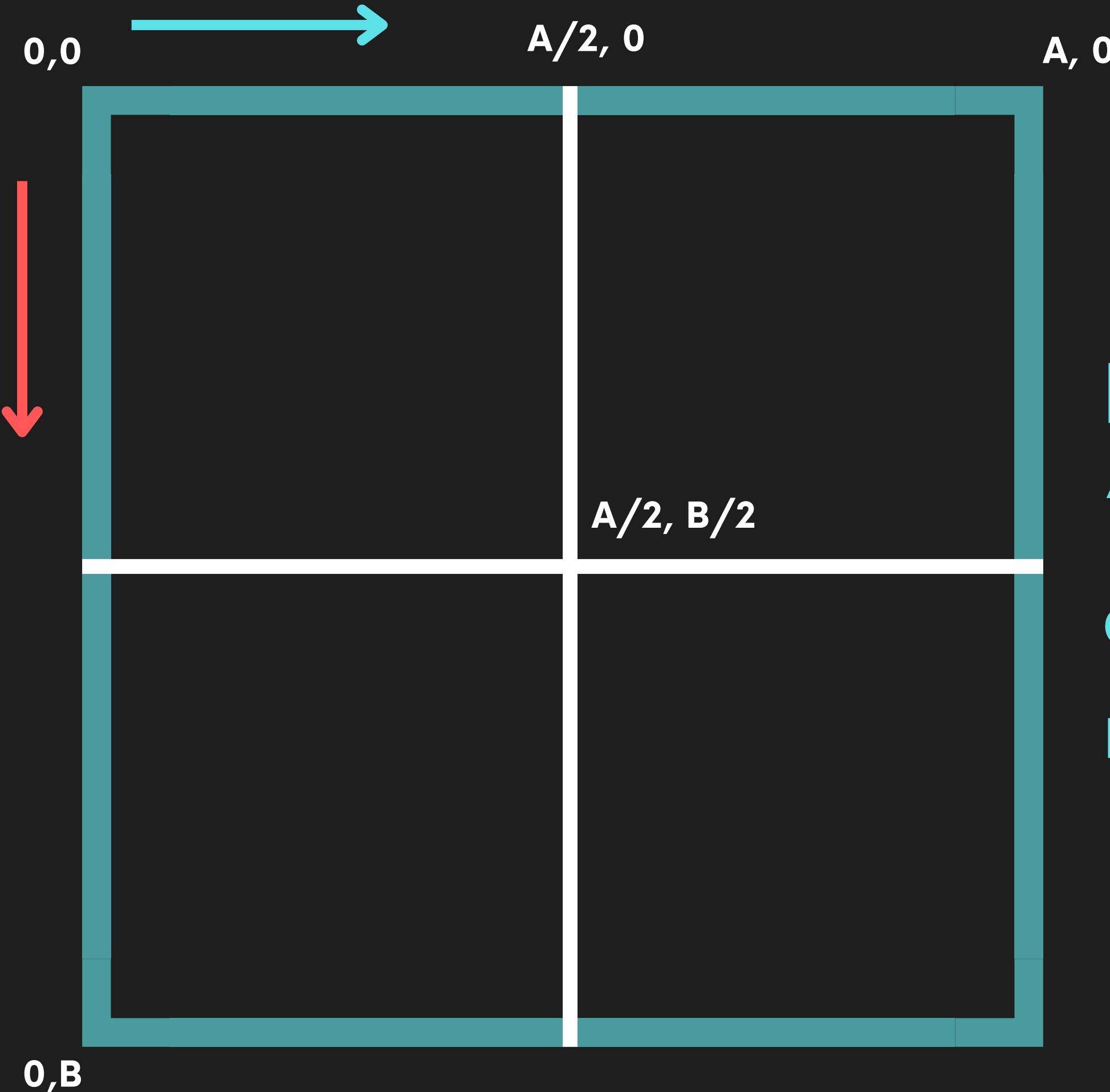
Tomar en cuenta



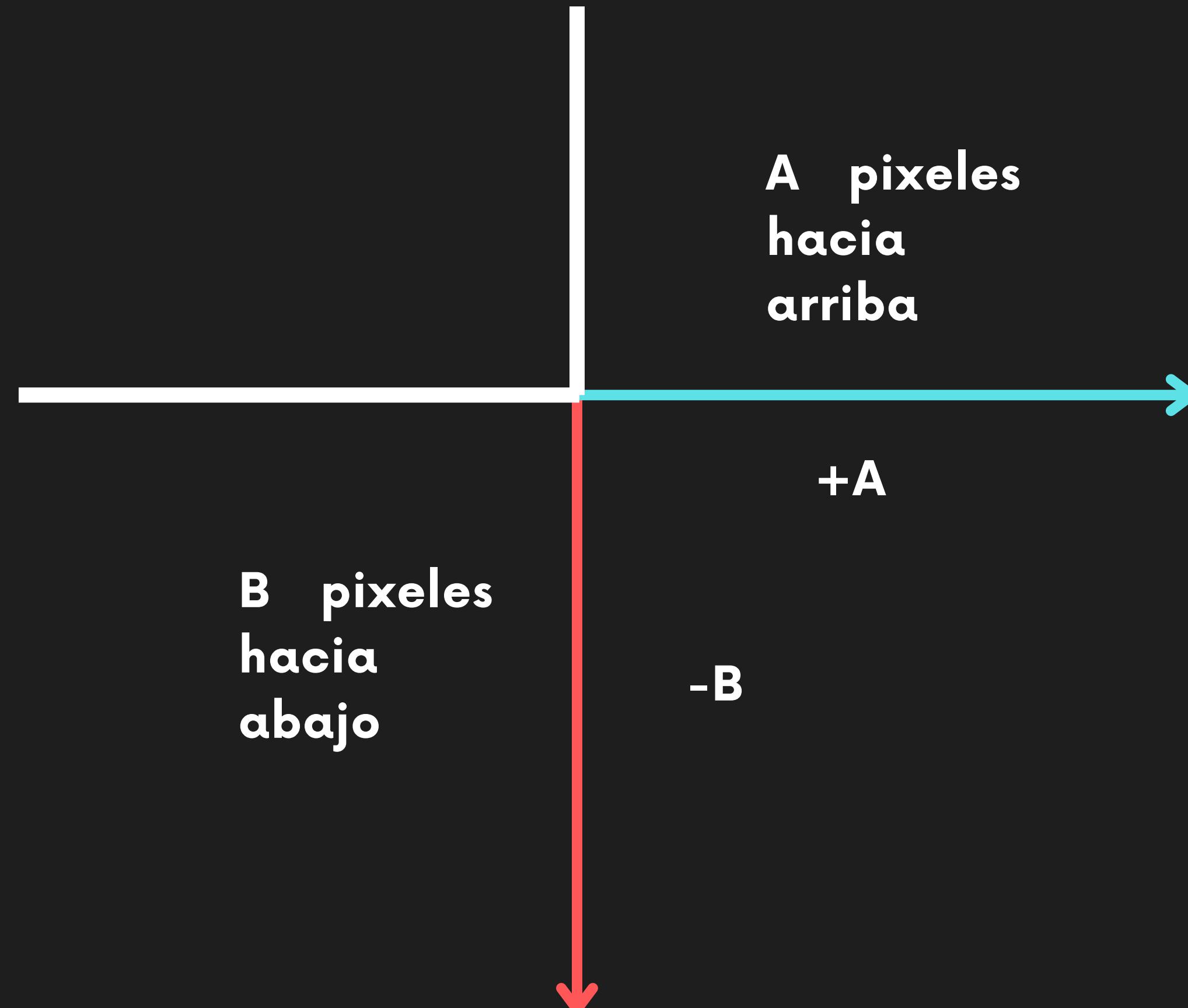
- Eje **(0,0)** de la ventana es la **esquina superior izquierda**.
- Cambiaremos esto al **centro** de la **ventana**.
- Cada pixel representa un punto, por lo que deberemos ampliar dicho rango para poder observar mejor las gráficas.
- La **consola** estará activa para poder ingresar **datos** y modificar **variables**.



CREAR PLANO
CARTESIANO



Eje
“central”
en estos
momentos





CREAR PLANO
CARTESIANO

>>>

Ancho = A
Alto = B

-A/2, 0

0,0

0, -B/2

A, 0

-N

A/2 + M, B/2 - N

A/2, B/2

+M

A/2, 0

0,B/2

Transforma-
ción final



Fórmula final

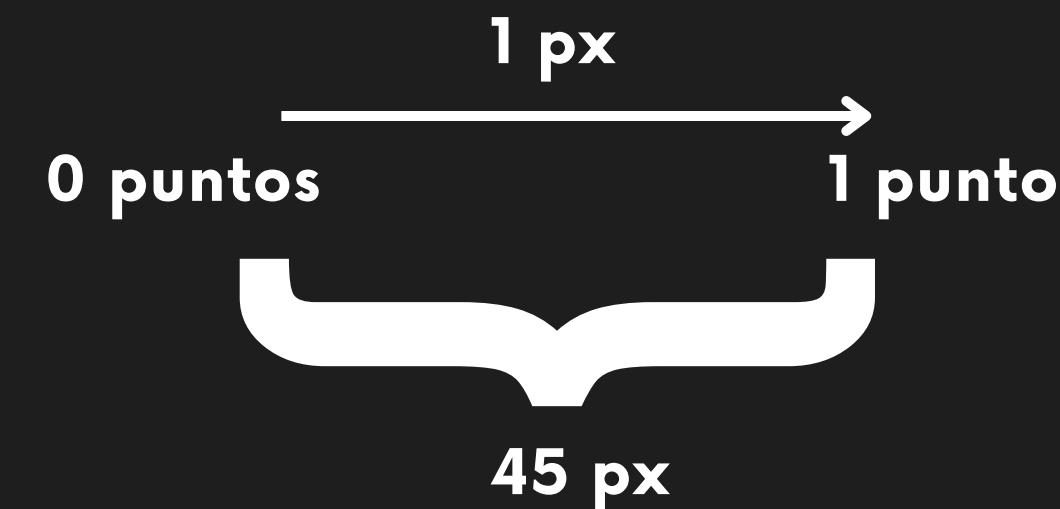
$$y = 3x^2 + 2x + 1$$
$$\frac{Y_1 - Y_2}{X_2 - X_1} = a \ . \ a \neq 0$$
$$\frac{\partial f}{\partial y} y_x = \frac{\partial f}{\partial x} - \frac{\partial f}{\partial y_x} y_{xx}$$
$$- \frac{\partial f}{\partial x}$$
$$\int f(x) dx = F(x) + C$$
$$x^2 + (y - \sqrt[3]{x^2})^2 = 1$$

Sea Ancho = A y Alto = B:
Punto en x,y = Punto $(A/2 + x, B/2 - y)$



Tenemos además, que 1px = 1punto

Deseamos agrandar un poco más la longitud de 1 punto, elegiremos la longitud para 1 punto 45 px, es decir:



1 punto = 45px



Tenemos además, que 1px = 1punto

Para lograr esta transformación, tenemos que analizar la formula de punto que habíamos creado anteriormente, donde cada x que tomemos avanzará por 1px.

Si: Punto en x,y = Punto ($A/2 + x*1$, $B/2 - y*1$) entonces:

Aplicando nuestra nueva relación:

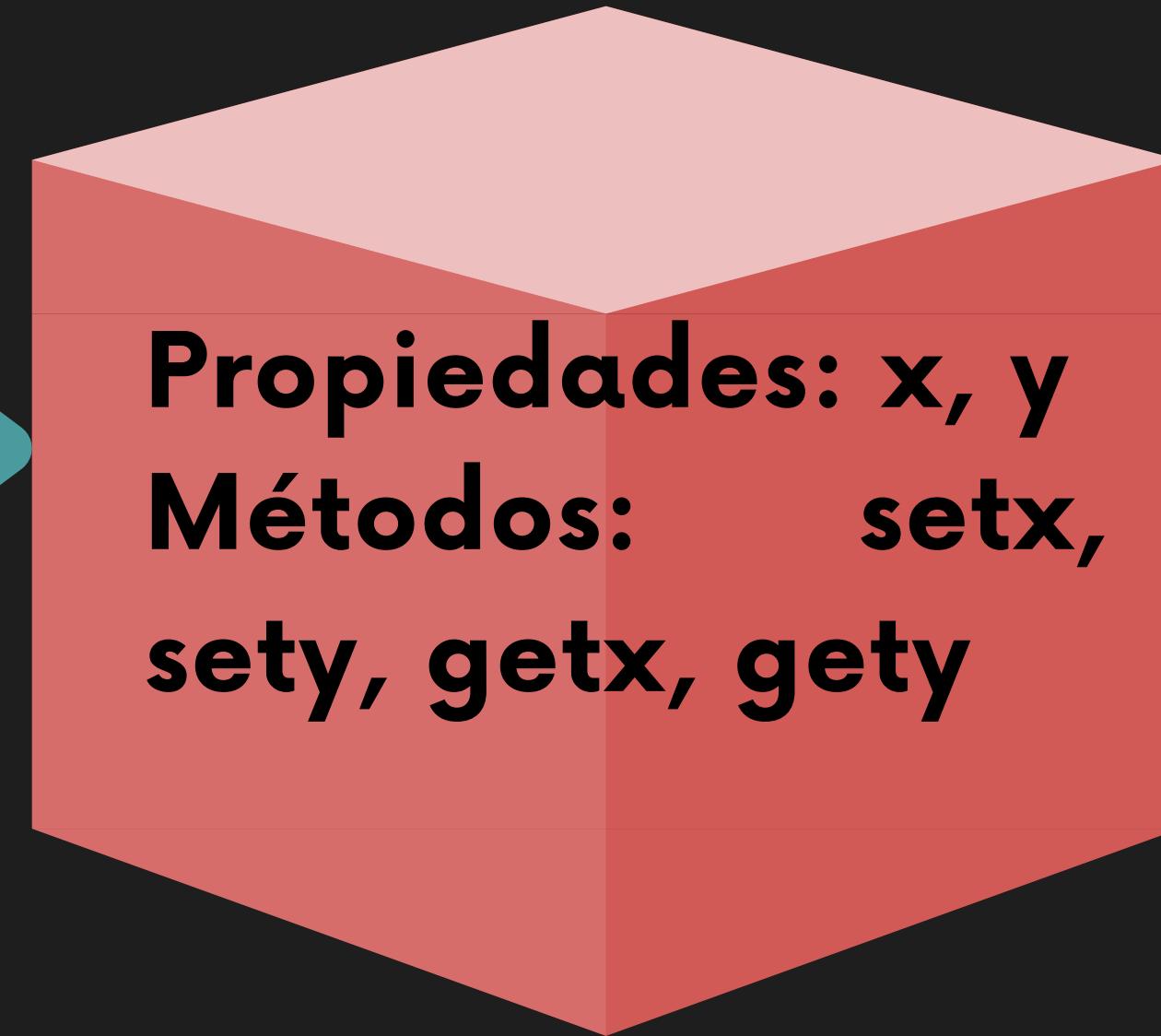
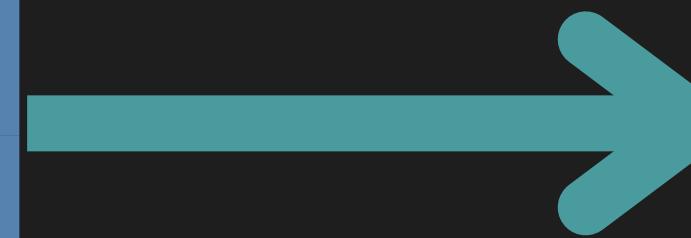
$$x = x * 45$$

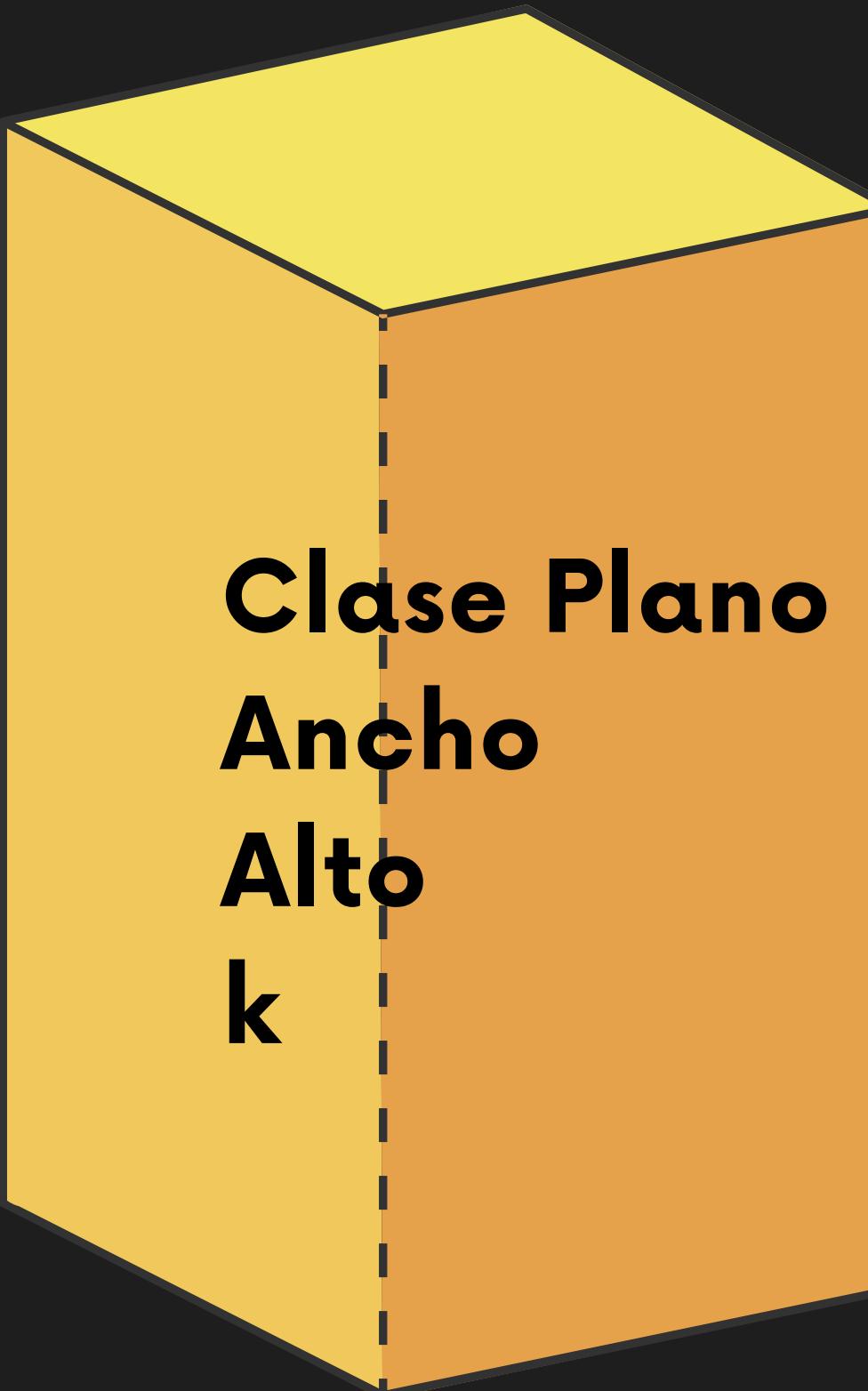
$$y = y * 45$$

Entonces:

Punto en x,y = Punto ($A/2 + x*45$, $B/2 - y*45$)

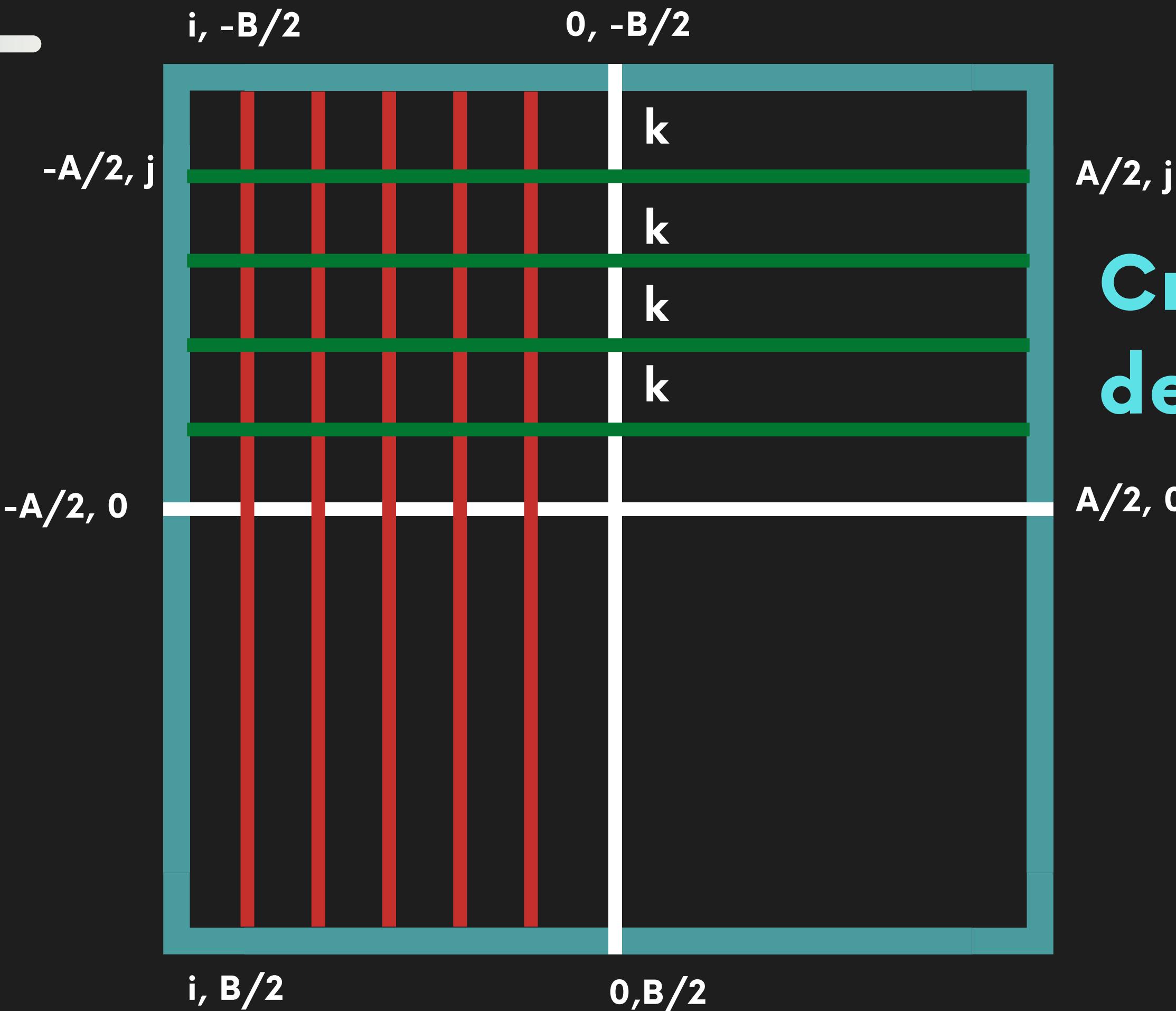
1 punto = 45px







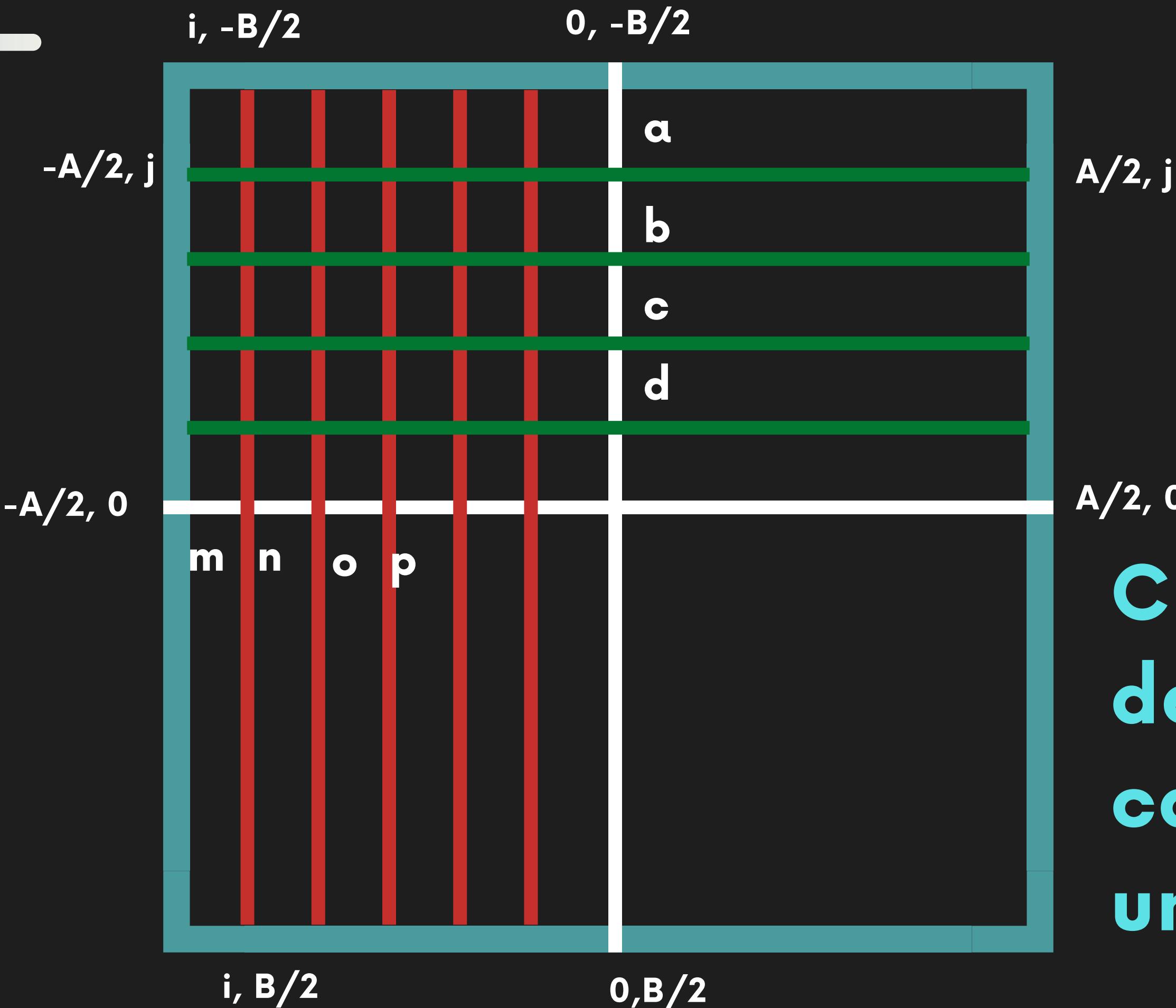
Ancho = A
Alto = B



Creación de líneas



Ancho = A
Alto = B



Creación
de letras en
cada
unidad



Estructura del menú

1. Iniciar el modo gráfico
2. Pintar gráfica
1. Punto
2. Recta
3. Limpiar ventana
4. Ingresar sistema de ecuaciones
0. Salir

Estructura de clases

```
Class Punto (x,y)
Class Plano {
  (Ancho, Alto, k)
  Función Set Ancho, Alto, k
  Función PosX, PosY
  Función Pintar Lineas
  Función Pintar Punto
  Función Pintar Recta
  Función Pintar ...
}
Class Recta (m, b){
  Función getY
```



Descripción de clases-Class Punto

Class Punto

Representa un punto en el plano cartesiano.

- Atributos:
 - **float x:** Coordenada x del punto.
 - **float y:** Coordenada y del punto.
- Métodos:
 - **void setX(float _x):** Establece la coordenada x del punto.
 - **void setY(float _y):** Establece la coordenada y del punto.
 - **float getX():** Obtiene la coordenada x del punto.
 - **float getY():** Obtiene la coordenada y del punto.



Descripción de clases-Class Recta

Representa una recta en el plano cartesiano definida por su pendiente y su intercepto.

- Atributos:
 - float m: Pendiente de la recta.
 - float b: Intercepto de la recta.
- Métodos:
 - Recta(float _m, float _b): Constructor que inicializa la pendiente y el intercepto.
 - float getY(float x): Calcula el valor de y para un valor dado de x usando la ecuación de la recta $y=mx+b$.
 - float getM(): Obtiene la pendiente de la recta.
 - float getB(): Obtiene el intercepto de la recta.



Descripción de clases-Class Plano

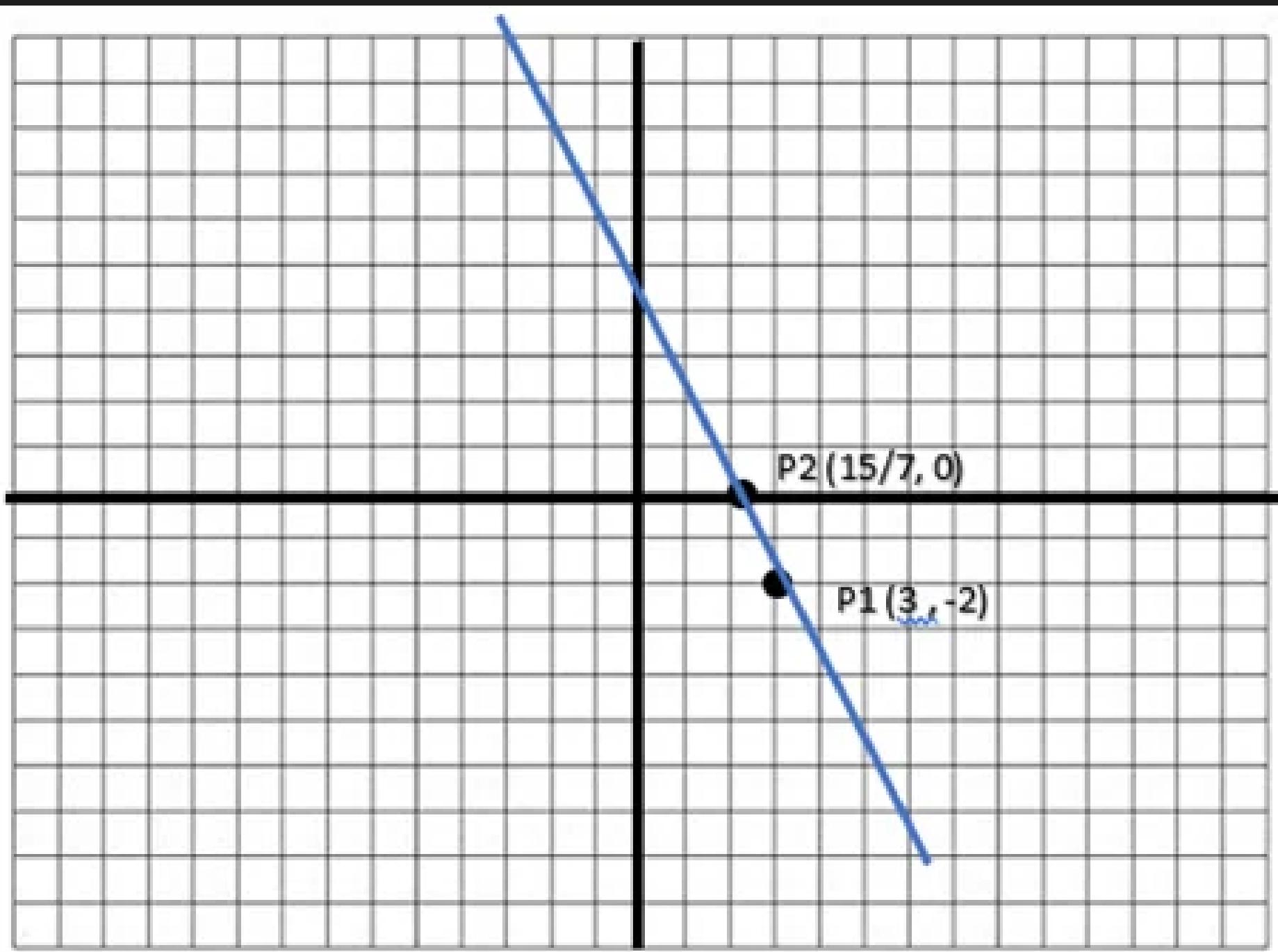
Representa el **plano cartesiano** donde se dibujan **puntos y rectas**.

- **Atributos:**
 - **int ancho:** Ancho del plano.
 - **int alto:** Alto del plano.
 - **int k:** Escala del plano.
- **Métodos:**
 - **Plano(int _ancho, int _alto, int _k):** Constructor que inicializa el ancho, alto y la escala del plano.
 - **void setAncho(int _ancho):** Establece el ancho del plano.
 - **void setAlto(int _alto):** Establece el alto del plano.
 - **void setK(int _k):** Establece la escala del plano.
 - **float posX(float _x):** Transforma la coordenada x al sistema de coordenadas de la ventana gráfica.
 - **float posY(float _y):** Transforma la coordenada y al sistema de coordenadas de la ventana gráfica.
 - **void pintar_lineas(int val_min, int val_max, bool tipo = true):** Dibuja las líneas del plano cartesiano.



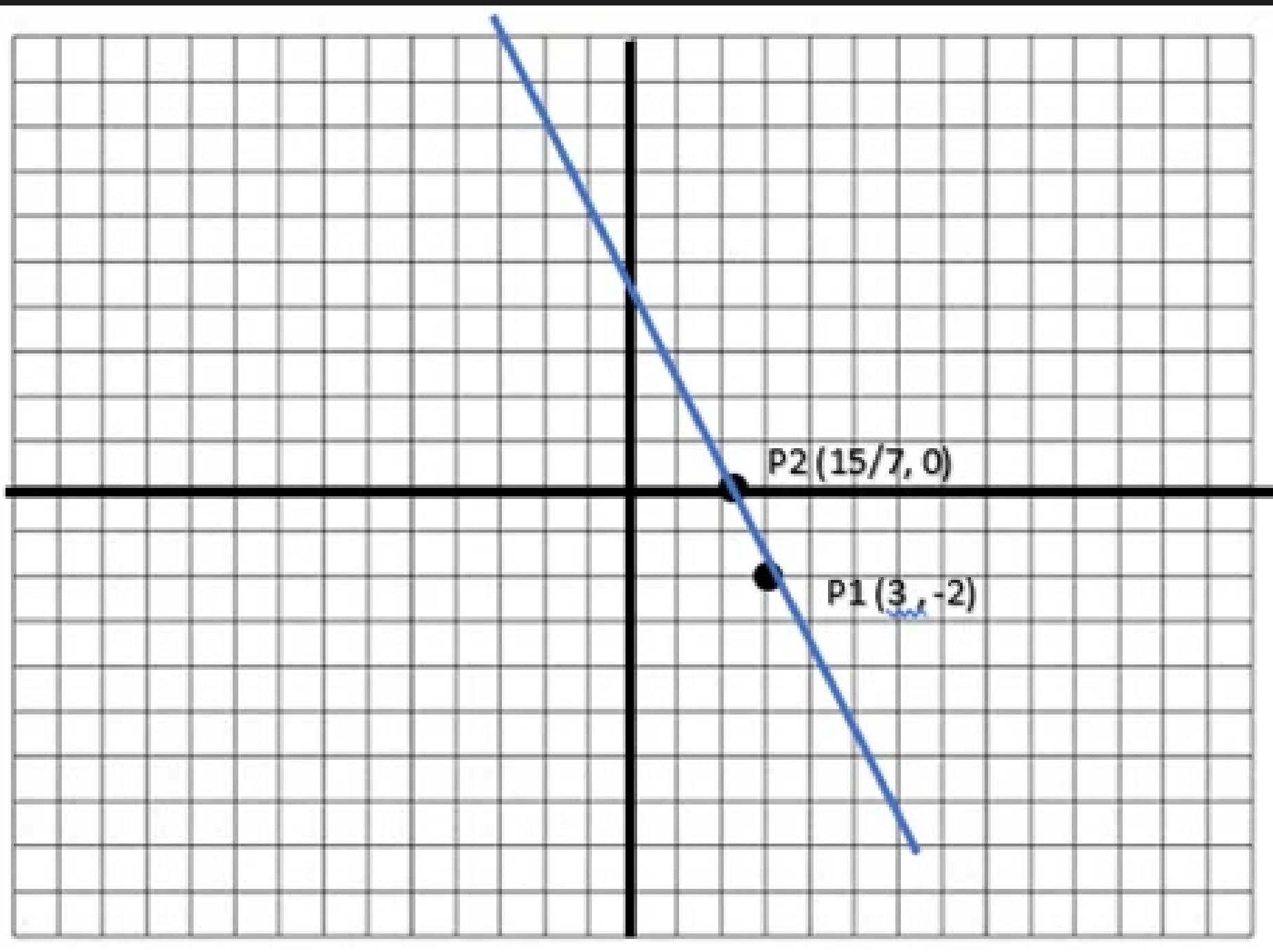
Descripción de clases-Class Plato

- **void pintar_punto(Punto p, int color = YELLOW): Dibuja un punto en el plano.**
- **void pintar_recta(float m, float b, int color = YELLOW): Dibuja una recta en el plano usando la pendiente y el intercepto.**
- **void pintar_interseccion(float m1, float b1, float m2, float b2): Calcula y dibuja el punto de intersección de dos rectas.**



Gráfica de una línea recta

Para graficar una línea recta a partir de su forma general $Ax + By + C = 0$, es necesario que al menos se determinen dos puntos a partir de esta expresión matemática, para ello es necesario asignarle un valor a cualquiera de las dos variables (ya sea x o y) para que a partir de ese valor se determine el correspondiente para la otra variable y con ellos formar las coordenadas de un punto. Dado que una línea recta se extiende de forma infinita se puede asignar cualquier valor positivo o negativo inclusive se puede dar valor de cero.

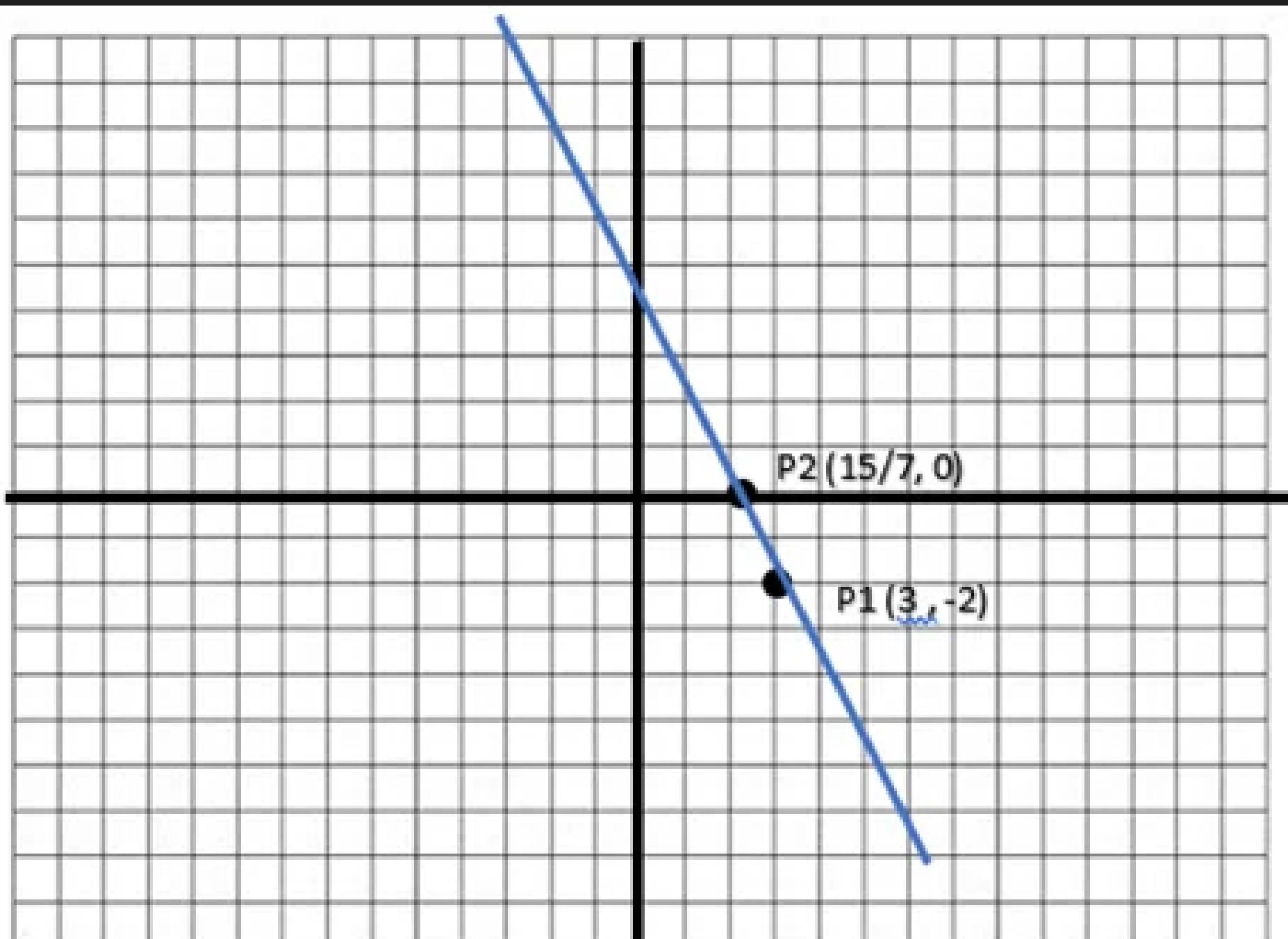


Gráfica de una línea recta

Tomemos el siguiente ejemplo: Grafica la recta $7x + 3y - 15 = 0$,

Solución: primero observemos que la expresión es una ecuación de primer grado con dos incógnitas, por lo que cumple con las características de una línea recta en su forma general. Ahora de acuerdo con lo que se describió al principio es necesario dar un valor a una de las variables por lo que para este ejemplo vamos a tomar $x = 3$ y con este valor vamos a determinar el valor de y a partir de la forma general de la recta $7x + 3y - 15 = 0$, sustituyendo el valor de x que asignamos:

$$(7)(3) + 3y - 15 = 0; 21 + 3y - 15 = 0; 3y + 6 = 0; 3y = -6; y = -6/3 = -2$$



Gráfica de una línea recta

Por lo tanto cuando la variable $x = 3$ la variable y debe tener un valor igual a -2 para hacer verdadera la ecuación y con esto formamos el primer punto:

P1 (3 , -2)

Pero un punto no es suficiente para trazar una recta, se requiere de al menos un segundo punto, por lo que es necesario repetir el procedimiento para determinarlo, para continuar con este ejemplo ahora se va a asignar el siguiente valor $y = 0$, se sustituye para determinar el correspondiente valor de x :

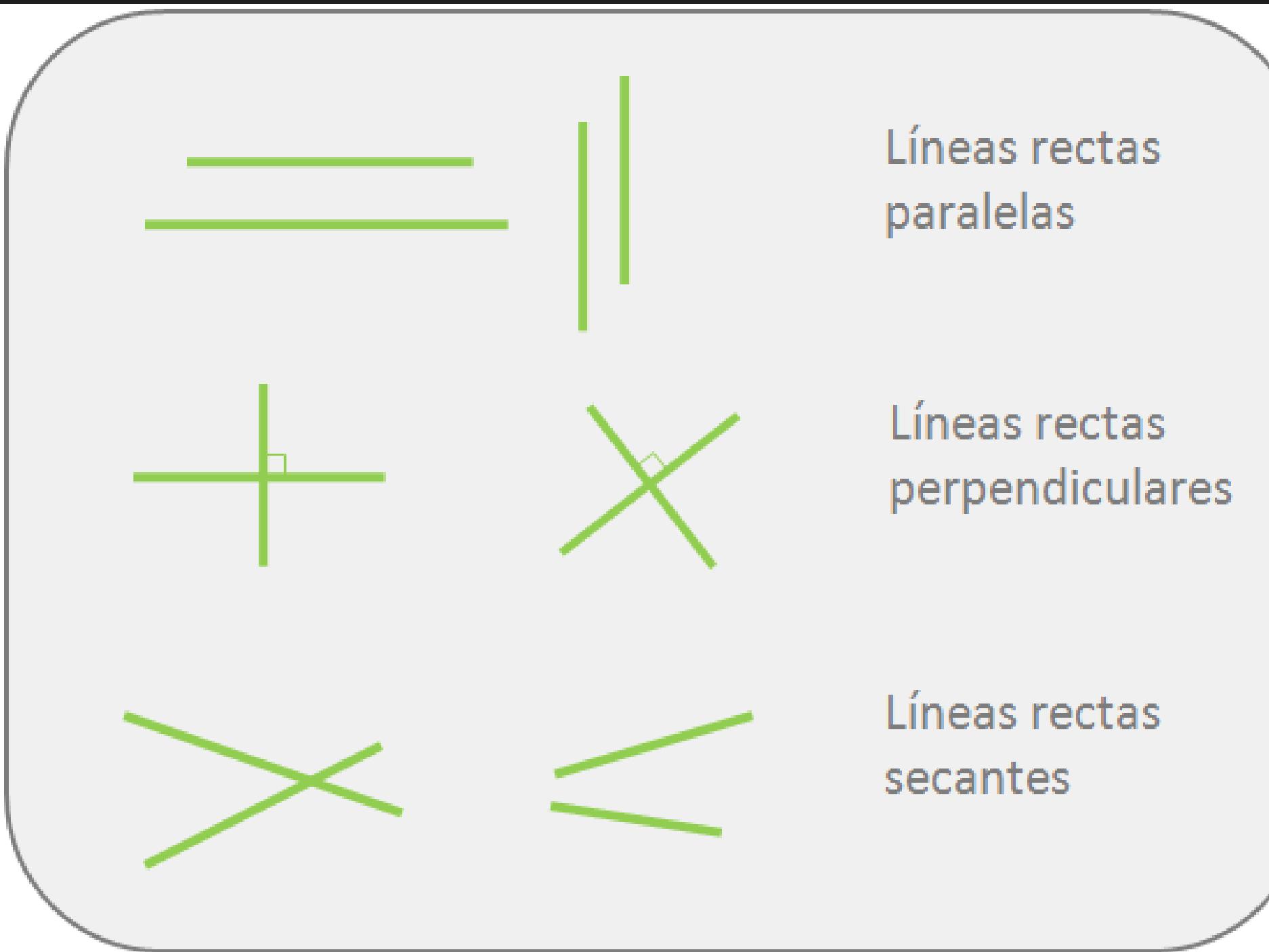
$$7x + 3y - 15 = 0; \quad 7x + (3)(0) - 15 = 0;$$

$$7x - 15 = 0; \quad 7x = 15; \quad x = 15/7, \text{ con lo}$$

que se obtiene el punto:

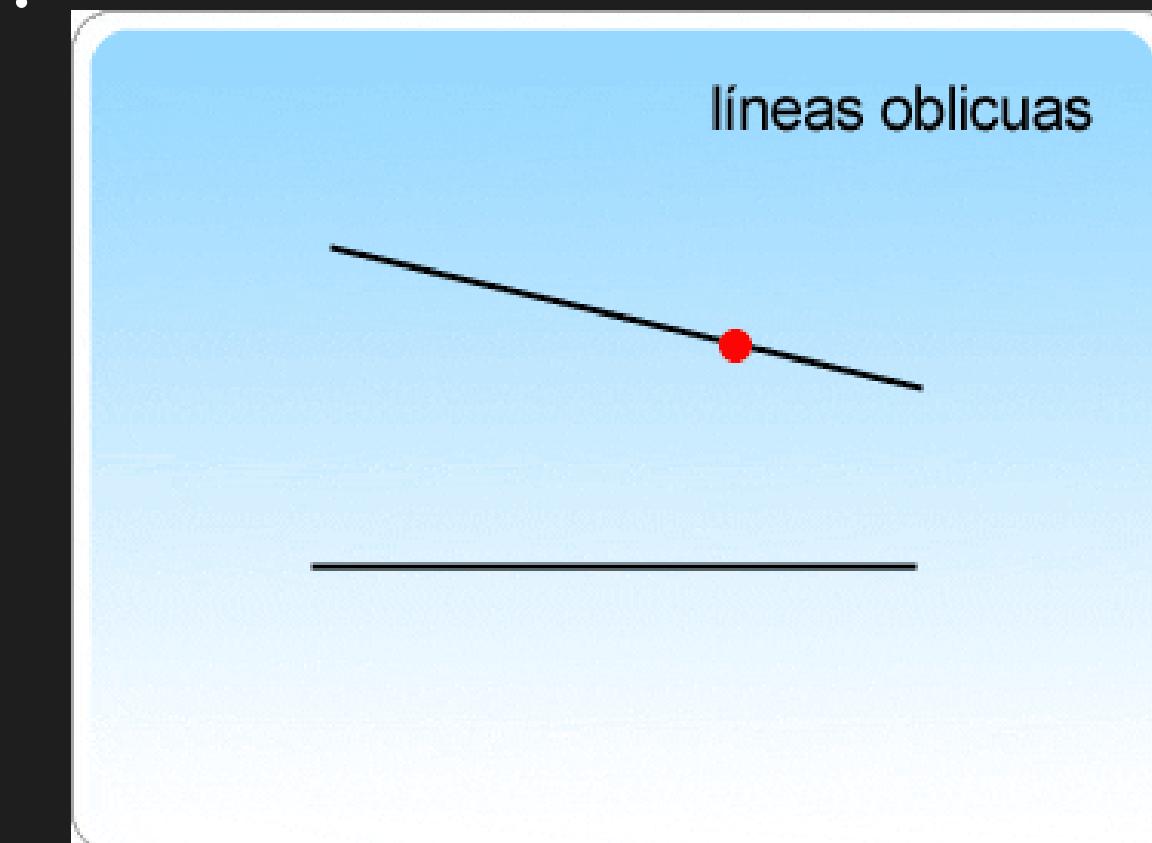
P2 (15/7 , 0)

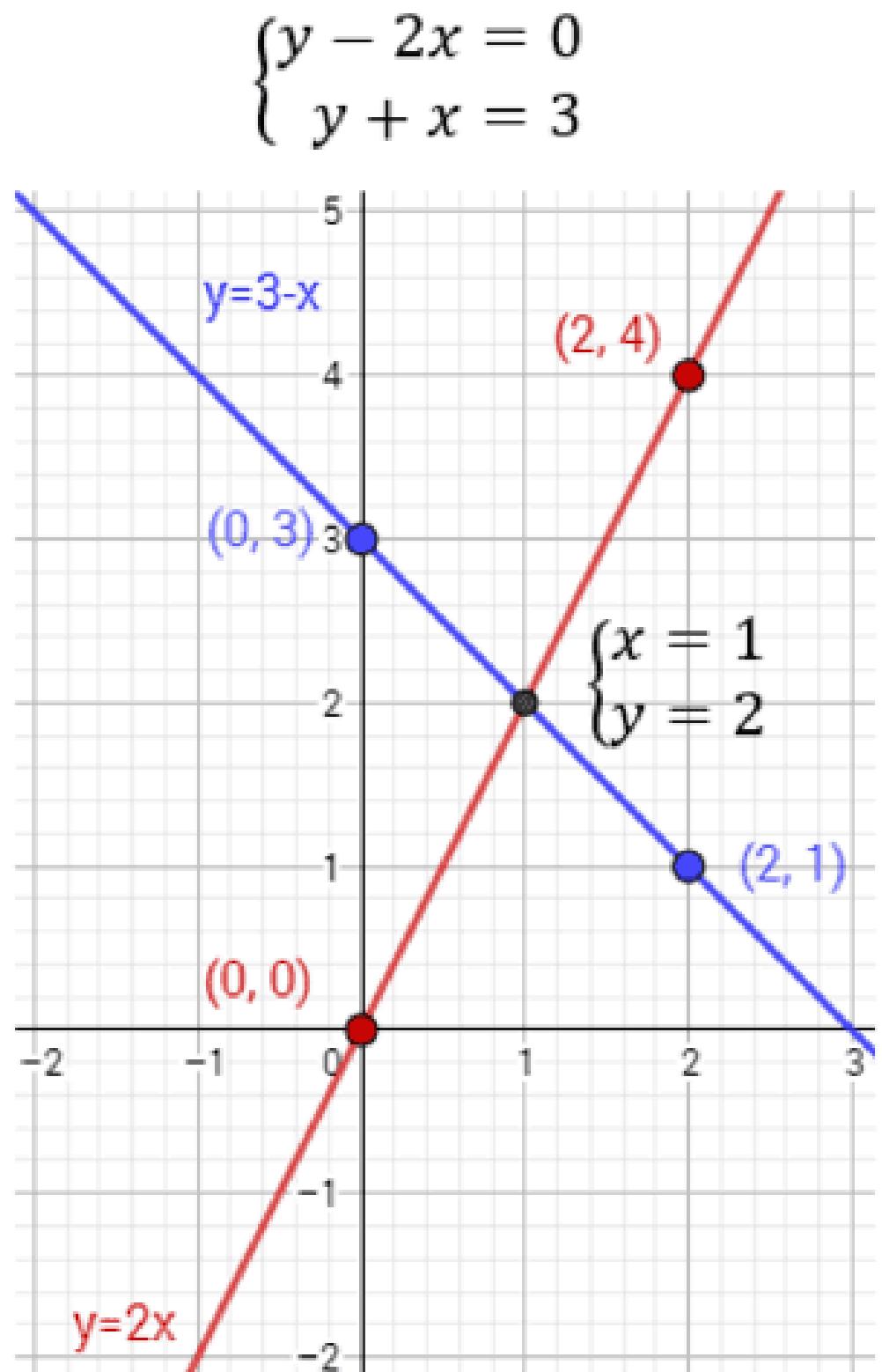
Y con estos dos puntos se puede graficar la línea recta:



Tipos de rectas

- a) **Rectas paralelas:** No se cortan NUNCA.
- b) **Rectas secantes:** Se cortan en un punto, formando CUATRO ANGULOS.
- c) **Rectas perpendiculares:** Son rectas que al cortarse forman CUATRO ÁNGULOS RECTOS.
- d) **Rectas oblicuas:** Son rectas que al cortarse forman un ÁNGULO DIFERENTE DE 90°.





Método gráfico

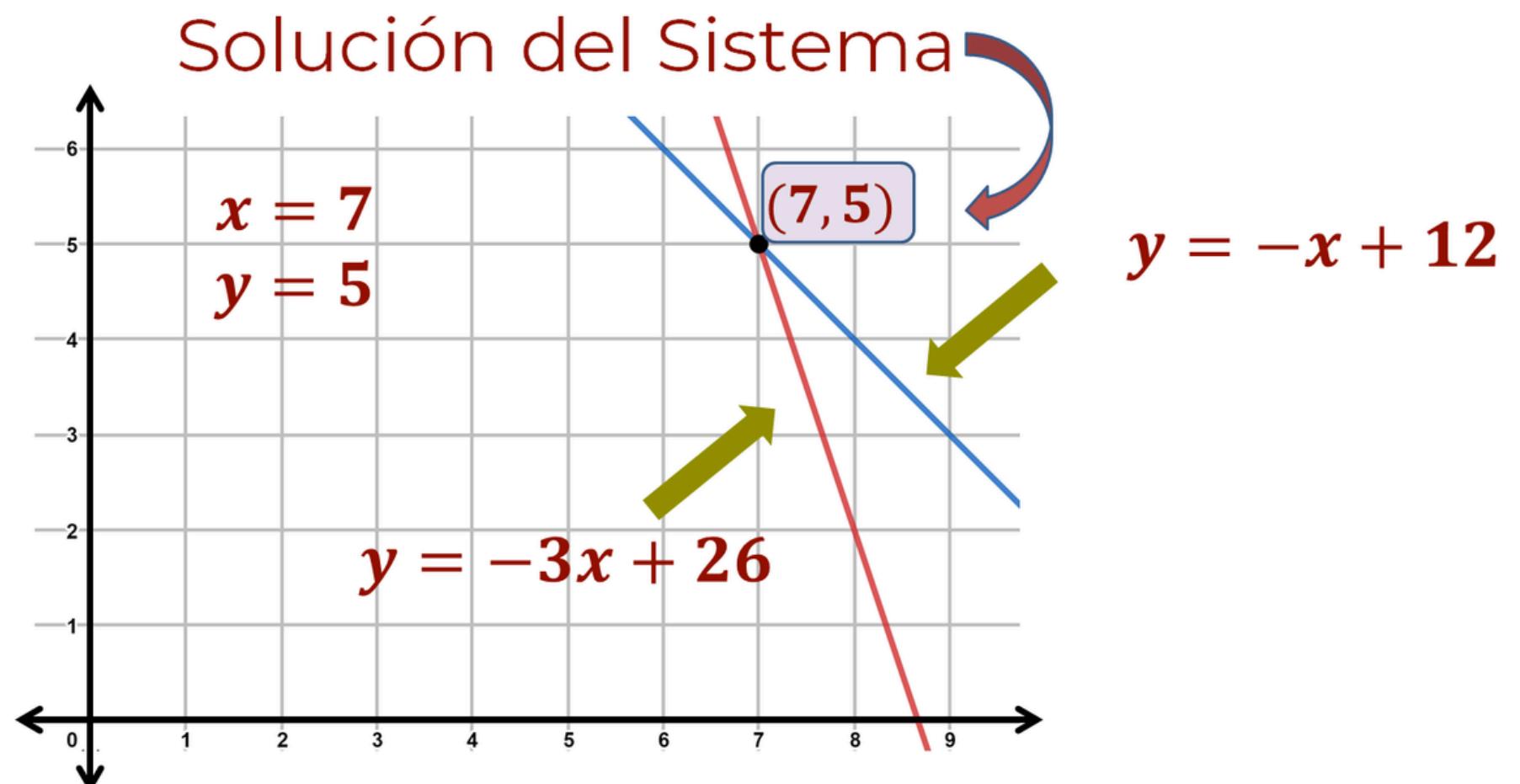
El método gráfico consiste en representar las ecuaciones de forma gráfica en un sistema de coordenadas cartesianas, siendo la solución el punto de intersección entre las gráficas.

Este método es práctico para sistemas lineales con dos incógnitas, donde la gráfica de cada ecuación es una recta, dando como resultado un único punto. Es de acotar que si las rectas son paralelas el sistema no tiene solución, y si son iguales tendrá infinitas soluciones.



Gráfica del sistema
de ecuaciones

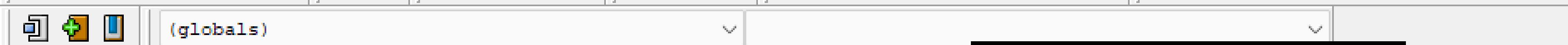
$$\begin{cases} y = -x + 12 \\ y = -3x + 26 \end{cases}$$



Pasos para aplicar el método gráfico

Para aplicar este método seguiremos los siguientes pasos:

- 1.- Como el sistema de ecuaciones está conformado con las variables (x) y (y) se procede a despejar la variable (y).
- 2.- Para cada ecuación se construye la tabla de valores considerando mínimo dos puntos.
- 3.- Se grafica cada ecuación según los valores de las tablas.
- 4.- Se verifica gráficamente el punto de intersección, siendo dicha coordenada (x,y) la solución al sistema de ecuaciones.



Project Classes Debug

- Funciones
 - constantes.cpp
 - funciones.cpp
 - main.cpp
 - utils.cpp

[*] constantes.cpp main.cpp utils.cpp funciones.cpp

```
1 // Constantes
2 const int ancho = 720;    /// A
3 const int alto = 720;     /// B
4 const int k = 45;
```

ANCHO DE LA VENTANA GRÁFICA
EN PÍXELESALTO DE LA VENTANA GRÁFICA
EN PÍXELES

```
5
6 // ATLITEC SARABIA DIEGO ALEJANDRO
```

ESCALA DEL PLANO CARTESIANO
(NÚMERO DE PÍXELES POR UNIDAD
EN EL PLANO)

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

Project Classes Debug [*] constantes.cpp main.cpp utils.cpp funciones.cpp

funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

```
1 #include "math.h" ← BIBLIOTECA ESTÁNDAR DE MATEMÁTICAS
2
3 class Recta { ← CLASE QUE REPRESENTA UNA RECTA EN EL PLANO CARTESIANO
4     private:
5         float m; ← PENDIENTE DE LA RECTA
6         float b; ← INTERSECCIÓN CON EL EJE Y
7
8     public:
9         Recta(float _m, float _b) { ← CONSTRUCTOR QUE INICIALIZA LOS
10            m = _m; ← VALORES DE M Y B
11            b = _b; ← ASIGNA LA PENDIENTE
12        } ← ASIGNA LA INTERSECCIÓN CON EL EJE Y
13
14        float getY(float x) {
15            return (m * x) + b; ← CALCULA Y = MX + B
16        }
17    }
```

MÉTODO QUE DEVUELVE EL VALOR DE Y PARA UN VALOR DADO DE X EN LA RECTA

```
17  
18     float getM() {← MÉTODO QUE DEVUELVE LA  
19         return m;← PENDIENTE DE LA RECTA  
20     }← DEVUELVE LA PENDIENTE  
21  
22     float getB() {← MÉTODO QUE DEVUELVE LA  
23         return b;← INTERSECCIÓN CON EL EJE Y  
24     }← DEVUELVE LA INTERSECCIÓN CON  
25 };← EL EJE Y  
26  
27  
28  
29
```

Funciones - [Funciones.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

BIBLIOTECA PARA FUNCIONES GRÁFICAS

(globals)

Project Classes Debug [*] constantes.cpp main.cpp utils.cpp funciones.cpp

Funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

```
1 #include <graphics.h>           ← BIBLIOTECA PARA ENTRADA Y SALIDA DE DATOS
2 #include <iostream>              ← BIBLIOTECA PARA FUNCIONES DE CONSOLA
3 #include <conio.h>               ← INCLUYE EL ARCHIVO CON UTILIDADES
4 //ATLITEC SARABIA DIEGO ALEJANDRO
5 #include "utils.cpp"             ← INCLUYE EL ARCHIVO CON LAS CONSTANTES
6 #include "constantes.cpp"         ← PERMITE USAR EL ESPACIO DE NOMBRES ESTÁNDAR
7
8 using namespace std;            ← DECLARACIÓN DE FUNCIONES
9
10 void pintarPlano();           ← DECLARACIÓN DE FUNCIONES
11 void menu_funciones(Plano);    ← DECLARACIÓN DE FUNCIONES
12 void menu_punto(Plano);       ← DECLARACIÓN DE FUNCIONES
13 void menu_recta(Plano);       ← DECLARACIÓN DE FUNCIONES
14 void menu_interseccion(Plano);← DECLARACIÓN DE FUNCIONES
15
16 int main() {                  ← VARIABLE PARA ALMACENAR LA ELECCIÓN DEL USUARIO
17     int eleccion;             ← VARIABLE PARA ALMACENAR LA ELECCIÓN DEL USUARIO
18 }
```

BIBLIOTECA PARA ENTRADA Y SALIDA DE DATOS

BIBLIOTECA PARA FUNCIONES DE CONSOLA

INCLUYE EL ARCHIVO CON UTILIDADES

INCLUYE EL ARCHIVO CON LAS CONSTANTES

PERMITE USAR EL ESPACIO DE NOMBRES ESTÁNDAR

DECLARACIÓN DE FUNCIONES

VARIABLE PARA ALMACENAR LA ELECCIÓN DEL USUARIO

Funciones - [Funciones.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

Project Classes Debug constantes.cpp [*] main.cpp utils.cpp funciones.cpp

Funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

19 Plano plano(ancho, alto, k);

20

21 do {

22 system("cls");

23 cout << " SOLUCIONADOR DE UN SISTEMA DE ECUACIONES 2X2 USAMANOS CON LA LIBRERIA GRAPH.H" << endl;

24 cout << "[1].Iniciar el modo grafico" << endl;

25 cout << "[2].Pintar grafica" << endl;

26 cout << "[3].Limpiar ventana" << endl;

27 cout << "[4].Ingresar sistema de ecuaciones" << endl;

28 cout << "[0].Salir" << endl;

29 cin >> eleccion;

30

31 switch(eleccion) {

32 case 1:

33 initwindow(ancho, alto);

34 pintarPlano();

35 break;

MENÚ

CREA UN OBJETO PLANO CON LAS DIMENSIONES ESPECIFICADAS

LIMPIA LA CONSOLA

LEE LA ELECCIÓN DEL USUARIO

EJECUTA ACCIONES BASADAS EN LA ELECCIÓN DEL USUARIO

INICIA UNA VENTANA GRÁFICA

DIBUJA EL PLANO EN LA VENTANA GRÁFICA

```
19 Plano plano(ancho, alto, k);  
20  
21 do {  
22 system("cls");  
23 cout << " SOLUCIONADOR DE UN SISTEMA DE ECUACIONES 2X2 USAMANOS CON LA LIBRERIA GRAPH.H" << endl;  
24 cout << "[1].Iniciar el modo grafico" << endl;  
25 cout << "[2].Pintar grafica" << endl;  
26 cout << "[3].Limpiar ventana" << endl;  
27 cout << "[4].Ingresar sistema de ecuaciones" << endl;  
28 cout << "[0].Salir" << endl;  
29 cin >> eleccion;  
30  
31 switch(eleccion) {  
32 case 1:  
33 initwindow(ancho, alto);  
34 pintarPlano();  
35 break;
```

Funciones - [Funciones.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

Project Classes Debug constantes.cpp [*] main.cpp utils.cpp funciones.cpp

Funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

```
36     case 2:  
37         menu_funciones(plano);  
38         break;  
39     case 3:  
40         cleardevice();  
41         pintarPlano();  
42         break;  
43     case 4:  
44         menu_interseccion(plano);  
45         break;  
46     default:  
47         break;  
48     }  
49 } while(eleccion != 0);  
50 system("pause>null");  
51 return 0;
```

MUESTRA EL MENÚ DE FUNCIONES

LIMPIA LA VENTANA GRÁFICA

VUELVE A DIBUJAR EL PLANO

MUESTRA EL MENÚ DE INTERSECCIÓN

REPITE EL CICLO HASTA QUE LA ELECCIÓN SEA 0 (SALIR)

PAUSA EL SISTEMA ANTES DE SALIR

FIN DEL PROGRAMA

DEV Funciones - [Funciones.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

Project Classes Debug constantes.cpp [*] main.cpp utils.cpp funciones.cpp

Funciones
 constantes.cpp
 funciones.cpp
 main.cpp
 utils.cpp

55 void pintarPlano() {
56 Plano plano = Plano(ancho, alto, k);
57
58 Punto p1 = Punto(plano posX(-ancho / 2 * k), plano posY(0));
59 Punto p2 = Punto(plano posX(ancho / 2 * k), plano posY(0));
60 Punto p3 = Punto(plano posX(0), plano posY(-alto / 2 * k));
61 Punto p4 = Punto(plano posX(0), plano posY(alto / 2 * k));
62
63 *// Líneas del plano*
64 setcolor(RED);
65 plano.pintar_lineas(-ancho / 2 * k, ancho / 2 * k, true);
66 plano.pintar_lineas(-alto / 2 * k, alto / 2 * k, false);
67
68 *// Líneas del eje*
69 setcolor(WHITE);
70 line((int)p1.getX(), (int)p1.getY(), (int)p2.getX(), (int)p2.getY());
71 line((int)p3.getX(), (int)p3.getY(), (int)p4.getX(), (int)p4.getY());
72 }

46

DEFINE CUATRO PUNTOS EN EL PLANO

FUNCTION PARA PINTAR EL PLANO

CREA UN OBJETO PLANO CON LAS DIMENSIONES ESPECIFICADAS

DIBUJA LAS LÍNEAS DEL PLANO

LÍNEAS HORIZONTALES

LÍNEAS VERTICALES

EJE X

EJE Y

Compiler Resources Compile Log Debug Find Results

Line: 40 Col: 21 Sel: 0 Lines: 186 Length: 4449 Insert Done parsing in 0.031 seconds

Funciones - [Funciones.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

globals

Project Classes Debug

constantes.cpp [*] main.cpp utils.cpp funciones.cpp

Funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

74 void menu_funciones(Plano plano) {

75 int eleccion;

76

77 do {

78 system("cls");

79 cout << "MENU DE FUNCIONES" << endl;

80 cout << "[1]. Punto" << endl;

81 cout << "[2]. Recta" << endl;

82 cout << "[0]. Volver" << endl;

83 cin >> eleccion;

84

85 switch(eleccion) {

86 case 1:

87 menu_punto(plano);

88 break;

89 case 2:

90 menu_recta(plano);

91 break;

FUNCIÓN QUE MUESTRA EL MENÚ DE FUNCIONES

VARIABLE PARA ALMACENAR LA ELECCIÓN DEL USUARIO

LIMPIA LA CONSOLA

LEE LA ELECCIÓN DEL USUARIO

EJECUTA ACCIONES BASADAS EN LA ELECCIÓN DEL USUARIO

MENÚ DE FUNCIONES

MUESTRA EL MENÚ DE PUNTOS

MUESTRA EL MENÚ DE RECTAS

47 Compiler Resources Compile Log Debug Find Results

```
void menu_funciones(Plano plano) {
    int eleccion;

    do {
        system("cls");
        cout << "MENU DE FUNCIONES" << endl;
        cout << "[1]. Punto" << endl;
        cout << "[2]. Recta" << endl;
        cout << "[0]. Volver" << endl;
        cin >> eleccion;

        switch(eleccion) {
            case 1:
                menu_punto(plano);
                break;
            case 2:
                menu_recta(plano);
                break;
        }
    }
}
```

Funciones - [Funciones.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

Project Classes Debug constantes.cpp [*] main.cpp utils.cpp funciones.cpp

Funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

```
92     default:  
93         break;  
94     }  
95 } while(eleccion != 0); // REPITE EL CICLO HASTA QUE LA ELECCIÓN SEA 0 (VOLVER)  
96  
97 void menu_punto(Plano p) { // FUNCIÓN QUE MUESTRA EL MENÚ DE PUNTOS  
98     float x, y; // VARIABLES PARA ALMACENAR LAS COORDENADAS DEL PUNTO  
99     int color; // VARIABLE PARA ALMACENAR EL COLOR DEL PUNTO  
100    int opcion; // VARIABLE PARA ALMACENAR LA OPCIÓN DEL USUARIO  
101  
102    do {  
103        system("cls");  
104        cout << "MENU PUNTO" << endl; // MENÚ PUNTO  
105        cout << "Punto (x, y)" << endl;  
106        cout << " X: ";  
107        cin >> x; // LEE LA COORDENADA X  
108        cout << " Y: ";  
109    }
```

LIMPIA LA CONSOLA

LEE LA COORDENADA X

MENÚ PUNTO

48

Funciones - [Funciones.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

(globals)

Project Classes Debug constantes.cpp [*] main.cpp utils.cpp funciones.cpp

Funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

110 `cin >> y;`
111 `cout << " Color: ";`
112 `cin >> color;` LEE EL COLOR DEL PUNTO

113
114 `p.pintar_punto(Punto(x, y), color);` DIBUJA EL PUNTO EN EL PLANO

115
116 `cout << "Presione 0 para volver al menu anterior: ";`
117 `cin >> opcion;` LEE LA OPCIÓN DEL USUARIO

118
119 `} while (opcion != 0);` REPITE EL CICLO HASTA QUE LA OPCIÓN SEA 0 (VOLVER)

120 `}`

121
122 `void menu_recta(Plano p) {` FUNCIÓN QUE MUESTRA EL MENÚ DE RECTAS

123 `float m, b;`
124 `int color;`
125 `int opcion;` VARIABLES PARA ALMACENAR LA PENDIENTE Y LA INTERSECCIÓN DE LA RECTA

126
127 `do {`

LEE LA COORDENADA Y

VARIABLE PARA ALMACENAR EL COLOR DE LA RECTA

VARIABLE PARA ALMACENAR LA OPCIÓN DEL USUARIO

49 Compiler Resources Compile Log Debug Find Results

Funciones - [Funciones.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

Project Classes Debug constantes.cpp [*] main.cpp utils.cpp funciones.cpp

Funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

MENÚ RECTA

```
128 system("cls");           ← LIMPIA LA CONSOLA
129 cout << "MENU RECTA" << endl;
130 cout << " y = mx + b " << endl;
131 cout << " m: ";
132 cin >> m;                ← LEE LA PENDIENTE
133 cout << " b: ";
134 cin >> b;                ← LEE LA INTERSECCIÓN
135 cout << " Color: ";
136 cin >> color;            ← LEE EL COLOR DE LA RECTA
137
138 p.pintar_recta(m, b, color); ← DIBUJA LA RECTA EN EL PLANO
139
140 cout << "Presione 0 para volver al menu anterior: ";
141 cin >> opcion;            ← LEE LA OPCIÓN DEL USUARIO
142 }
143 } while (opcion != 0);      ← REPITE EL CICLO HASTA QUE LA OPCIÓN SEA 0 (VOLVER)
144
145
```

Compiler Resources Compile Log Debug Find Results

50

Funciones - [Funciones.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

(globals)

Project Classes Debug constantes.cpp [*] main.cpp utils.cpp funciones.cpp

Funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

VARIABLE PARA ALMACENAR LA OPCIÓN DEL USUARIO

MENÚ SISTEMA DE ECUACIONES

```
146 void menu_interseccion(Plano p) {  
147     float m1, b1, m2, b2;  
148     int opcion;  
149  
150     do {  
151         system("cls");  
152         cout << "MENU SISTEMA DE ECUACIONES" << endl;  
153         cout << "Ingrese la primera ecuacion lineal de la forma: y = m1"  
154         cout << " m1: ";  
155         cin >> m1;  
156         cout << " b1: ";  
157         cin >> b1;  
158         cout << "Ingrese la segunda ecuacion lineal de la forma: y = m2"  
159         cout << " m2: ";  
160         cin >> m2;  
161         cout << " b2: ";  
162         cin >> b2;  
163     } while (opcion != 4);  
164 }
```

FUNCTION QUE MUESTRA EL MENÚ DE INTERSECCIÓN DE RECTAS

VARIABLES PARA ALMACENAR LAS PENDIENTES E INTERSECCIONES DE LAS RECTAS

LIMPIA LA CONSOLA

LEE LA PENDIENTE DE LA PRIMERA RECTA

LEE LA INTERSECCIÓN DE LA PRIMERA RECTA

LEE LA PENDIENTE DE LA SEGUNDA RECTA

LEE LA INTERSECCIÓN DE LA SEGUNDA RECTA

51

Compiler Resources Compile Log Debug Find Results

Funciones - [Funciones.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

Project Classes Debug constantes.cpp [*] main.cpp utils.cpp funciones.cpp

DIBUJA LA PRIMERA RECTA EN AMARILLO

DIBUJA LA SEGUNDA RECTA EN AMARILLO

DIBUJA EL PUNTO DE INTERSECCIÓN

LEE LA OPCIÓN DEL USUARIO

REPITE EL CICLO HASTA QUE LA OPCIÓN SEA 0 (VOLVER)

```
163
164     p.pintar_recta(m1, b1, YELLOW);
165     p.pintar_recta(m2, b2, YELLOW);
166
167     p.pintar_interseccion(m1, b1, m2, b2);
168
169     cout << "Presione 0 para volver al menu anterior: ";
170     cin >> opcion;
171
172 } while (opcion != 0);
173
174
175
176
177
178
179
180
```

Funciones - [Funciones.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

Project Classes Debug constantes.cpp main.cpp utils.cpp funciones.cpp

Funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

```
1 #include <graphics.h>           INCLUYE LA BIBLIOTECA PARA FUNCIONES GRÁFICAS
2 #include "funciones.cpp"          INCLUYE EL ARCHIVO CON LA CLASE RECTA
3 #include <iostream>              INCLUYE LA BIBLIOTECA PARA ENTRADA Y SALIDA DE DATOS
4
5 using namespace std;             PERMITE USAR EL ESPACIO DE NOMBRES ESTÁNDAR
6
7 class Punto {                   CLASE PUNTO PARA REPRESENTAR UN PUNTO EN EL PLANO
8     private:
9         float x;                  COORDENADA X DEL PUNTO
10        float y;                 COORDENADA Y DEL PUNTO
11
12    public:
13        Punto(float _x, float _y) { CONSTRUCTOR QUE INICIALIZA EL PUNTO CON COORDENADAS X E Y
14            setX(_x);
15            setY(_y);
16        }
17
18        void setX(float _x) {      ESTABLECE LA COORDENADA X
19            // Código para establecer la coordenada X
20        }
21
22        float getX() {           GETTER PARA LA COORDENADA X
23            return x;
24        }
25
26        void setY(float _y) {      ESTABLECE LA COORDENADA Y
27            // Código para establecer la coordenada Y
28        }
29
30        float getY() {           GETTER PARA LA COORDENADA Y
31            return y;
32        }
33
34        void moverse(float dx, float dy) { MUEVE EL PUNTO DEDICANDO LOS DESPLAZAMIENTOS X E Y
35            x += dx;
36            y += dy;
37        }
38
39        void dibujar() {           DIBUJA EL PUNTO EN LA CONSOLA O EN UNA GRÁFICA
40            cout << "Punto(" << x << ", " << y << ")";
41        }
42
43    };
44
45 int main() {
46     Punto punto(100, 200);
47     punto.dibujar();
48     punto.moverse(50, 50);
49     punto.dibujar();
50 }
```

53



TDM-GCC 4.9.2 32-bit Release

Project Classes Debug constantes.cpp main.cpp utils.cpp funciones.cpp

Funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

```
19     x = _x;
20 }
21
22 void setY(float _y) {
23     y = _y;
24 }
25
26 float getX() {
27     return x;
28 }
29
30 float getY() {
31     return y;
32 }
33 };
34
35 class Plano {
36     private:
```

ESTABLECE LA COORDENADA Y

DEVUELVE LA COORDENADA X

DEVUELVE LA COORDENADA Y

CLASE PLANO PARA REPRESENTAR UN PLANO CARTESIANO

Funciones - [Funciones.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

Project Classes Debug constantes.cpp main.cpp utils.cpp funciones.cpp

Funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

37 int ancho;

38 int alto;

39 int k;

40

41 public:

42 Plano(int _ancho, int _alto, int _k) {

43 ancho = _ancho;

44 alto = _alto;

45 k = _k;

46 }

47 void setAncho(int _ancho) {

48 ancho = _ancho;

49 }

50 void setAlto(int _alto) {

51 alto = _alto;

52 }

53 }

54 }

ANCHO DEL PLANO

ALTO DEL PLANO

FACTOR DE ESCALA PARA EL PLANO

CONSTRUCTOR QUE INICIALIZA EL PLANO CON ANCHO, ALTO Y FACTOR DE ESCALA

ESTABLECE EL ANCHO DEL PLANO

ESTABLECE EL ALTO DEL PLANO

55

Compiler Resources Compile Log Debug Find Results



ESTABLECE EL FACTOR DE ESCALA DEL PLANO

Project Classes Debug

constantes.cpp main.cpp utils.cpp funciones.cpp

Funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

```
56 void setk(int _k) {  
57     k = _k;  
58 }  
59
```

CONVIERTA LA COORDENADA X DEL MUNDO REAL A LA POSICIÓN EN EL PLANO

```
60 float posX(float _x) {  
61     return ((ancho / 2) + (_x * k));  
62 }  
63
```

CONVIERTA LA COORDENADA Y DEL MUNDO REAL A LA POSICIÓN EN EL PLANO

```
64 float posY(float _y) {  
65     return ((alto / 2) - (_y * k));  
66 }  
67
```

DIBUJA LAS LÍNEAS DEL PLANO

```
68 void pintar_lineas(int val_min, int val_max, bool tipo = true) {  
69     Punto plano_linea_1 = Punto(0.0f, 0.0f);  
70     Punto plano_linea_2 = Punto(0.0f, 0.0f);  
71 }
```

PUNTOS PARA LAS LÍNEAS DEL PLANO

```
72     Punto pos_letra = Punto(0.0f, 0.0f);  
73     char unidad_texto[4];
```

PUNTO PARA LAS ETIQUETAS DE LAS UNIDADES

ALMACENA LA ETIQUETA DE LA UNIDAD



Project Classes Debug

constantes.cpp main.cpp utils.cpp funciones.cpp

Funciones
 constantes.cpp
 funciones.cpp
 main.cpp
 utils.cpp

BUCLE PARA DIBUJAR LAS LÍNEAS

```
75 for (int i = val_min; i <= val_max; i++) {  
76     plano_linea_1.setX(posX((tipo) ? i : val_min));  
77     plano_linea_1.setY(posY((tipo) ? val_min : i));  
78  
79     plano_linea_2.setX(posX((tipo) ? i : val_max));  
80     plano_linea_2.setY(posY((tipo) ? val_max : i));  
81  
82     if (tipo) {  
83         // Horizontal  
84         pos_letra.setX(posX(i));  
85         pos_letra.setY(posY(0.0f));  
86     } else {  
87         // Vertical  
88         pos_letra.setX(posX(0.0f));  
89         pos_letra.setY(posY(i));  
90     }  
91  
92     sprintf(unidad_texto, "%d", i);
```

COORDENADAS PARA LAS LÍNEAS HORIZONTALES

COORDENADAS PARA LAS LÍNEAS VERTICALES

CONVIERTE LA UNIDAD A TEXTO

DEV Funciones - [Funciones.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

Project Classes Debug constantes.cpp main.cpp utils.cpp funciones.cpp

Funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

```
94
95 DIBUJA LA LÍNEA
96
97 GUARDA EL COLOR ACTUAL
98
99 CAMBIA EL COLOR A BLANCO
100
101 DIBUJA LA ETIQUETA DE LA UNIDAD
102
103 RESTAURA EL COLOR ORIGINAL
104
105
106 DIBUJA UN PUNTO EN EL PLANO
107
108 DIBUJA UNA RECTA EN EL PLANO
109
110 PUNTO INICIAL
111
```

94
95 **DIBUJA LA LÍNEA**
96
97 **GUARDA EL COLOR ACTUAL**
98
99 **CAMBIA EL COLOR A BLANCO**
100
101 **DIBUJA LA ETIQUETA DE LA UNIDAD**
102
103 **RESTAURA EL COLOR ORIGINAL**
104
105
106 **DIBUJA UN PUNTO EN EL PLANO**
107
108 **DIBUJA UNA RECTA EN EL PLANO**
109
110 **PUNTO INICIAL**
111

Compiler Resources Compile Log Debug Find Results

Line: 1 Col: 1 Sel: 0 Lines: 140 Length: 3514 Insert Done parsing in 0.015 seconds

58



```

112     Recta recta(m, b);          CALCULA LA
113
114     float longitud = (float)ancho / (2 * k);    LONGITUD DE
115
116     for (float i = -longitud; i <= longitud; i += 0.01) {    LA RECTA
117         p.setX(i);
118         p.setY(recta.getY(i));
119
120         putpixel(posX(p.getX()), posY(p.getY()), color);
121     }
122
123
124     void pintar_interseccion(float m1, float b1, float m2, float b2) {    DIBUJA EL PUNTO
125         float x_interseccion = (b2 - b1) / (m1 - m2);    DE
126         float y_interseccion = m1 * x_interseccion + b1;    INTERSECCIÓN
127
128         Punto interseccion(x_interseccion, y_interseccion);    CREA UN PUNTO
129         pintar_punto(interseccion, WHITE);    DE
130
131
132
133
134
135
136
137
138
139
140

```

DIBUJA EL PUNTO DE INTERSECCIÓN DE DOS RECTAS

CREA UNA RECTA CON PENDIENTE M E INTERSECCIÓN B

BUCLE PARA DIBUJAR LA RECTA

DIBUJA EL PUNTO DE INTERSECCIÓN EN BLANCO

CALCULA LA LONGITUD DE LA RECTA

CALCULA LA INTERSECCIÓN DE LAS DOS RECTAS

DIBUJA EL PUNTO DE LA RECTA

DEV Funciones - [Funciones.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

Project Classes Debug constantes.cpp main.cpp utils.cpp funciones.cpp

Funciones

- constantes.cpp
- funciones.cpp
- main.cpp
- utils.cpp

```
123
124     void pintar_interseccion(float m1, float b1, float m2, float b2) {
125         float x_interseccion = (b2 - b1) / (m1 - m2);
126         float y_interseccion = m1 * x_interseccion + b1;
127
128         Punto interseccion(x_interseccion, y_interseccion);
129         pintar_punto(interseccion, WHITE);
130
131         cout << "Punto de intersección: (" << x_interseccion << ", " << y_i
132     }
133 };
134
135
136
137
138
139
140 |
```

IMPRIME LAS COORDENADAS DEL PUNTO DE INTERSECCIÓN

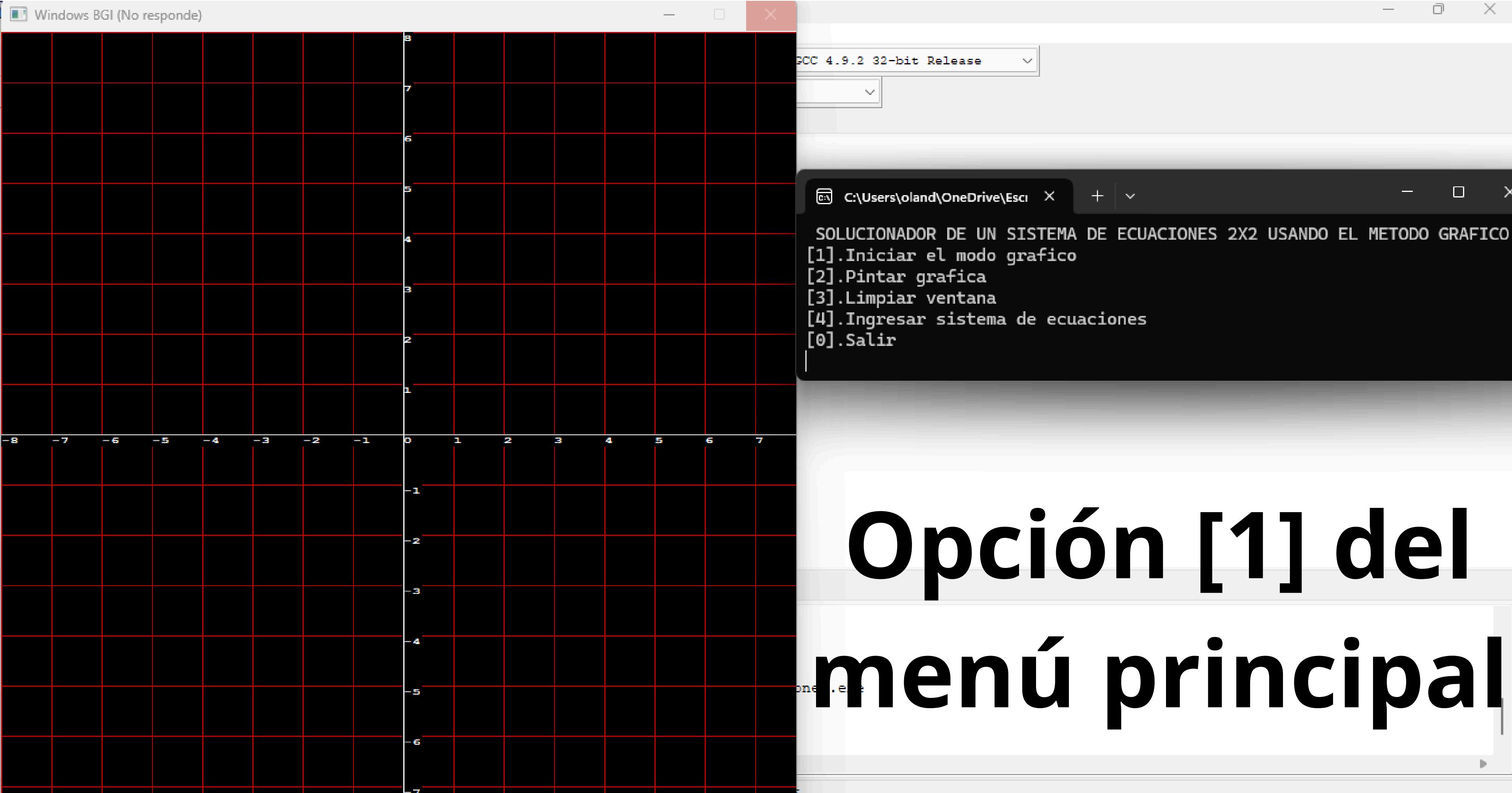
PRUEBAS

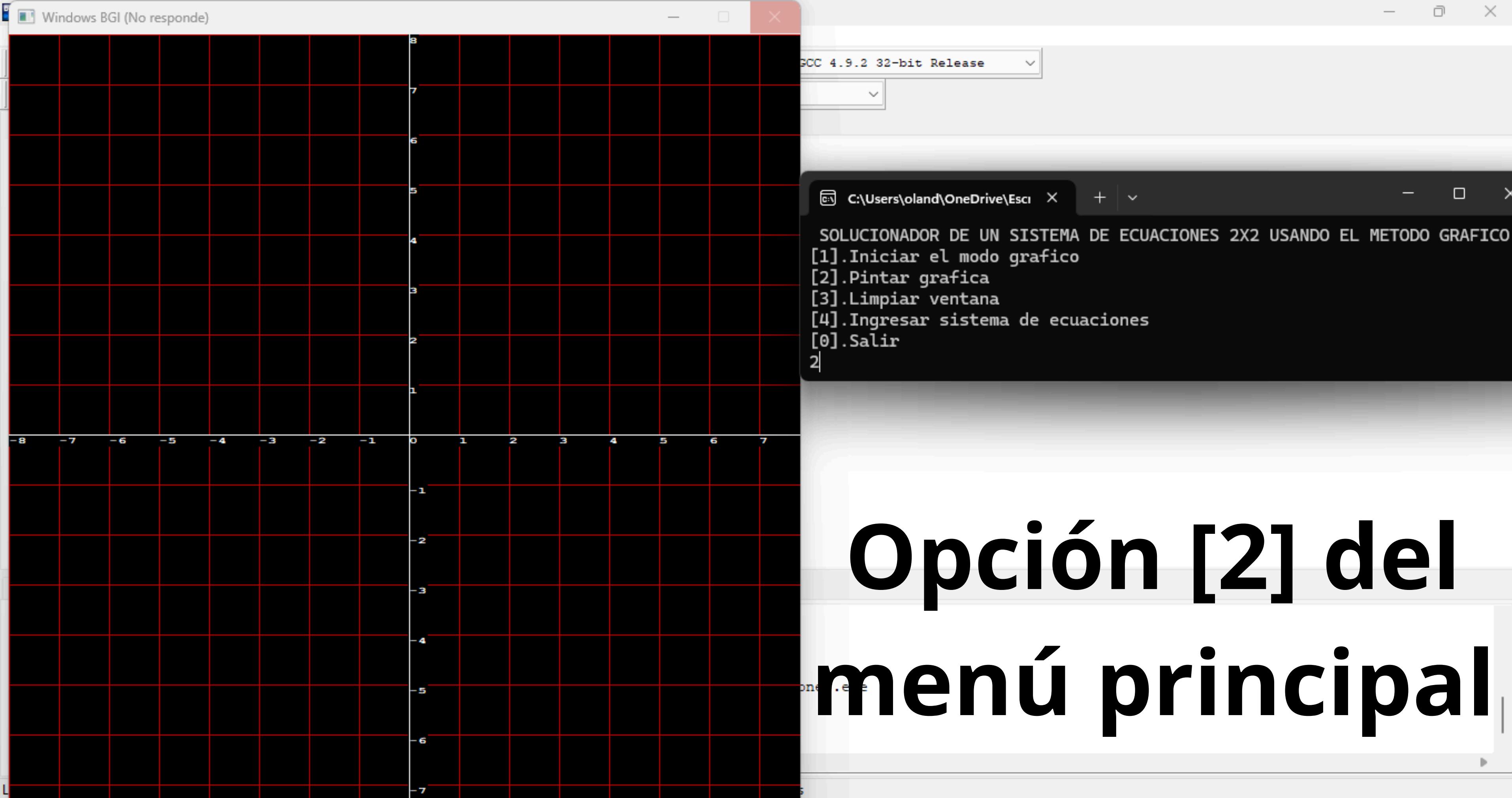
Compilación del programa

The screenshot shows a terminal window with a dark background and white text. At the top, it displays the path 'C:\Users\oland\OneDrive\Escritorio' followed by a file icon, a close button ('X'), and a '+' button. Below this, the title 'SOLUCIONADOR DE UN SISTEMA DE ECUACIONES 2X2 USANDO EL METODO GRAFICO' is printed in capital letters. Underneath the title, a numbered list of options is shown:

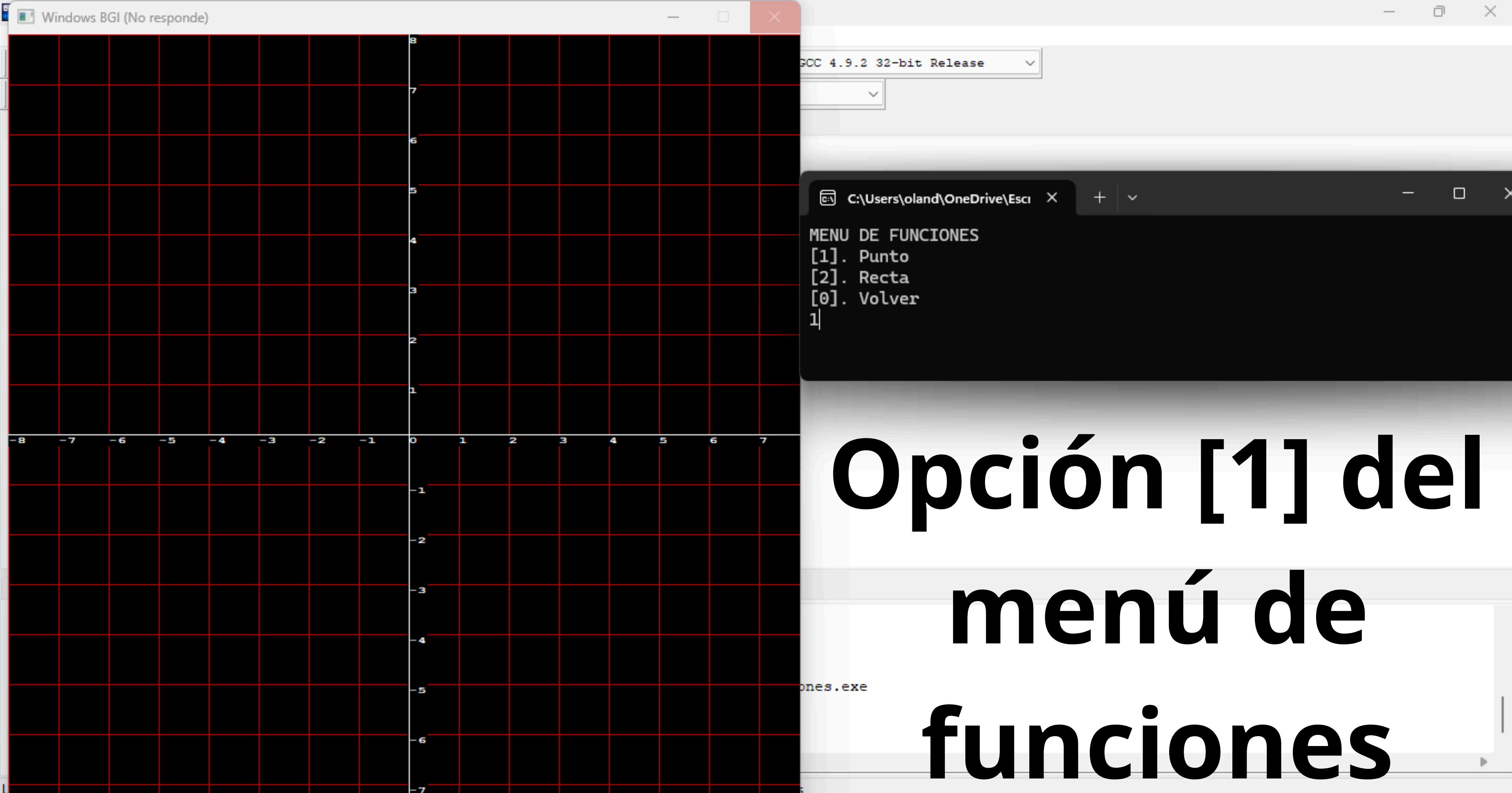
- [1]. Iniciar el modo grafico
- [2]. Pintar grafica
- [3]. Limpiar ventana
- [4]. Ingresar sistema de ecuaciones
- [0]. Salir

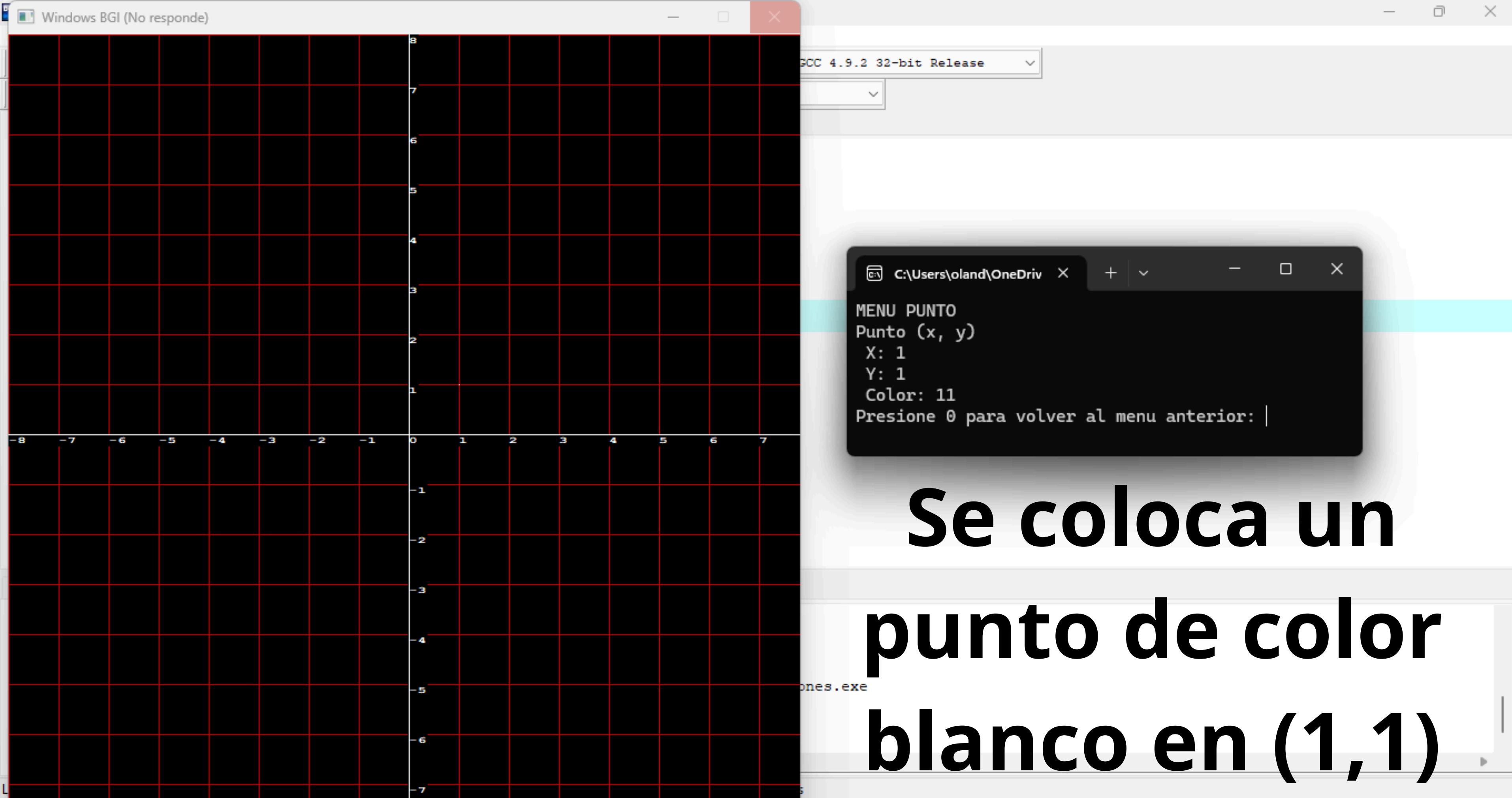
A vertical cursor bar is visible on the left side of the terminal window.



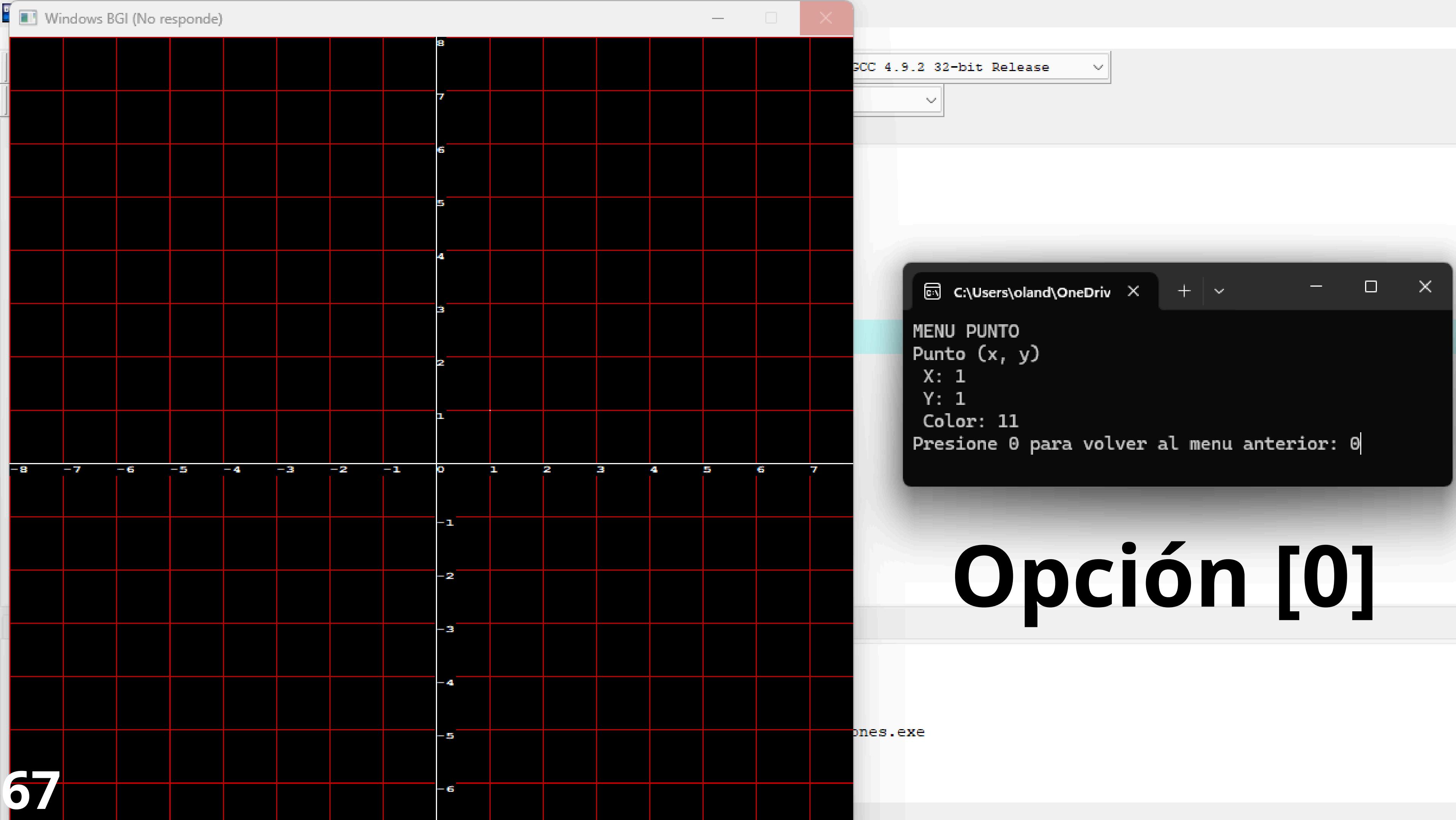


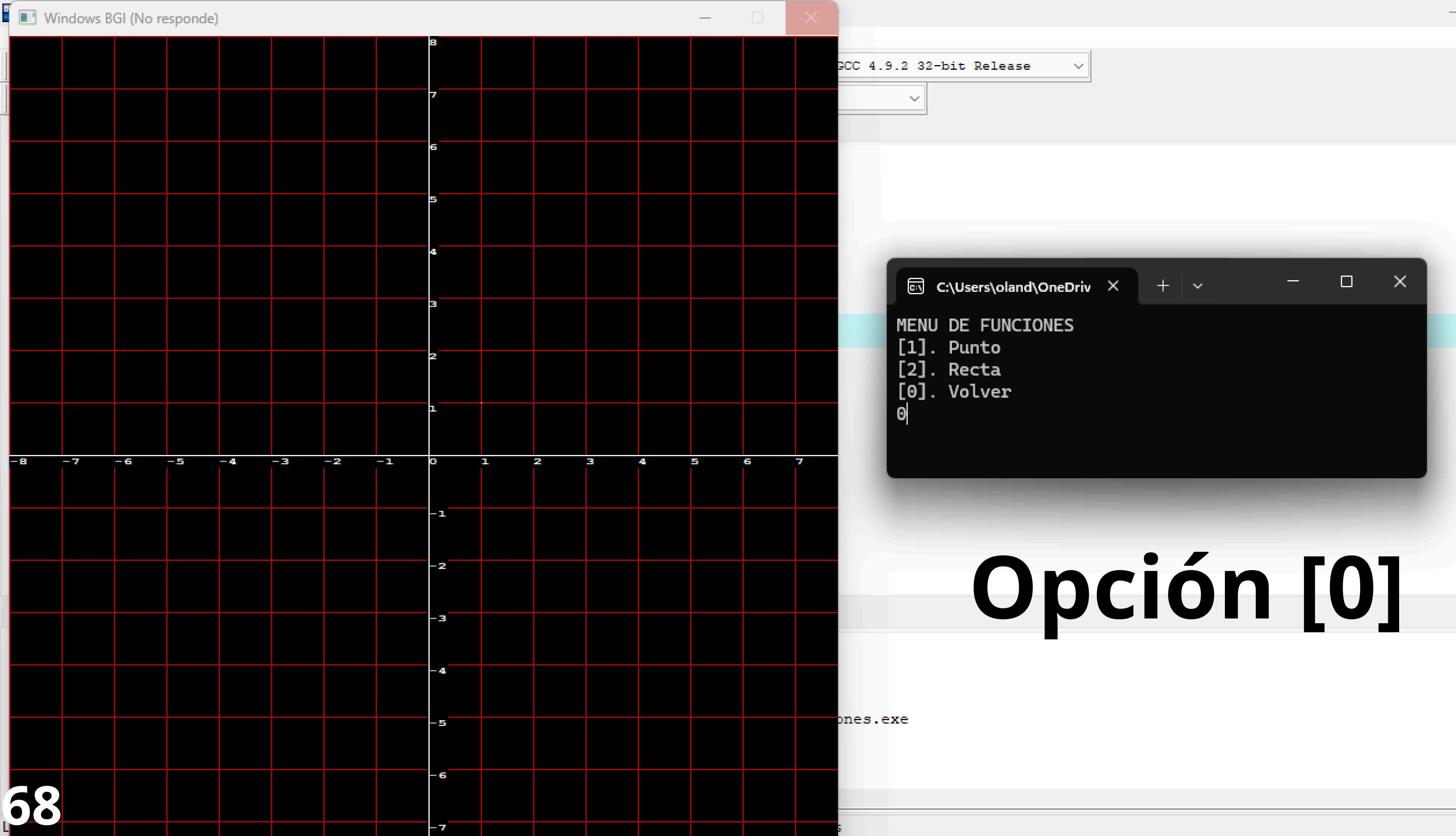
Opción [2] del menú principal

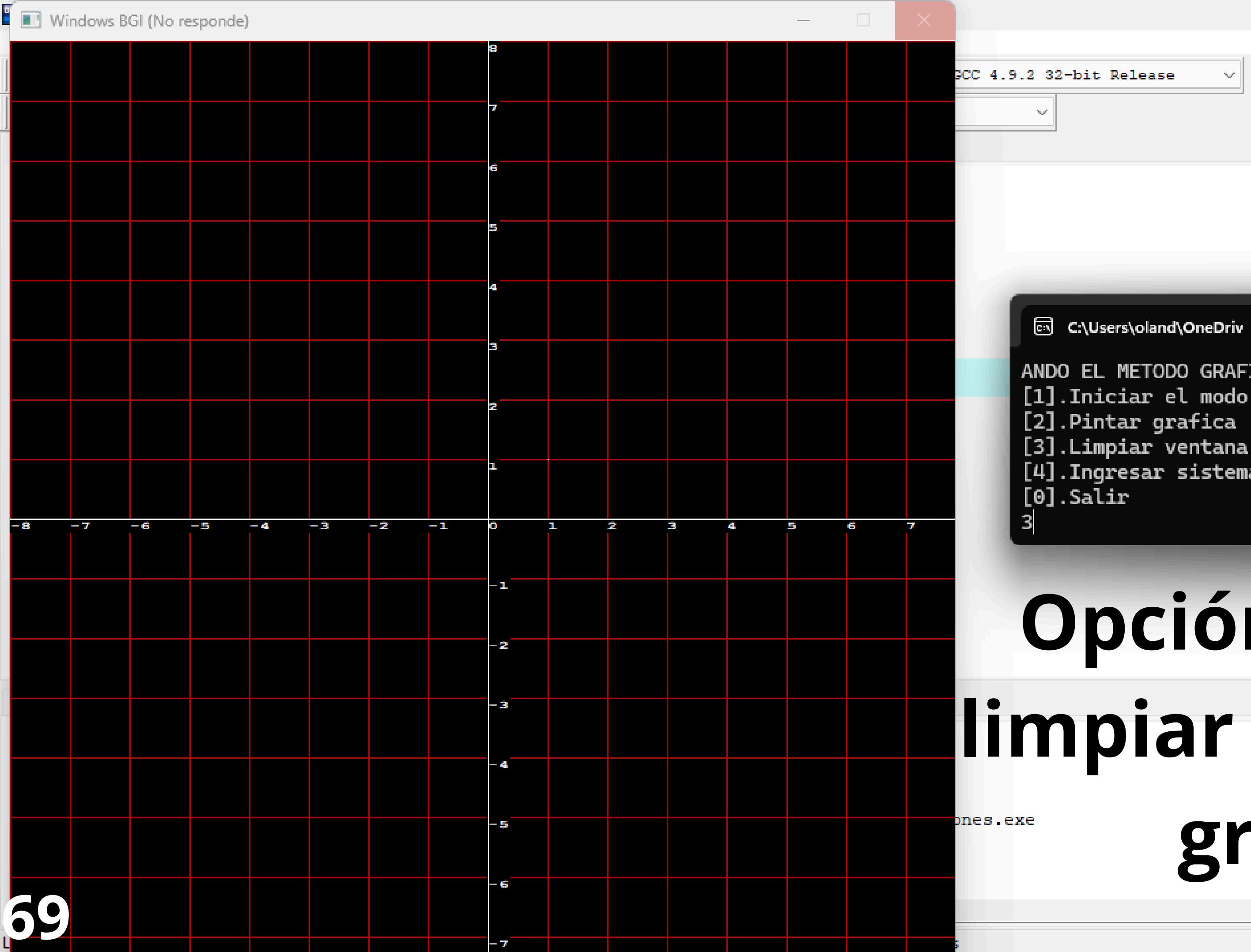




Se coloca un
punto de color
blanco en (1,1)

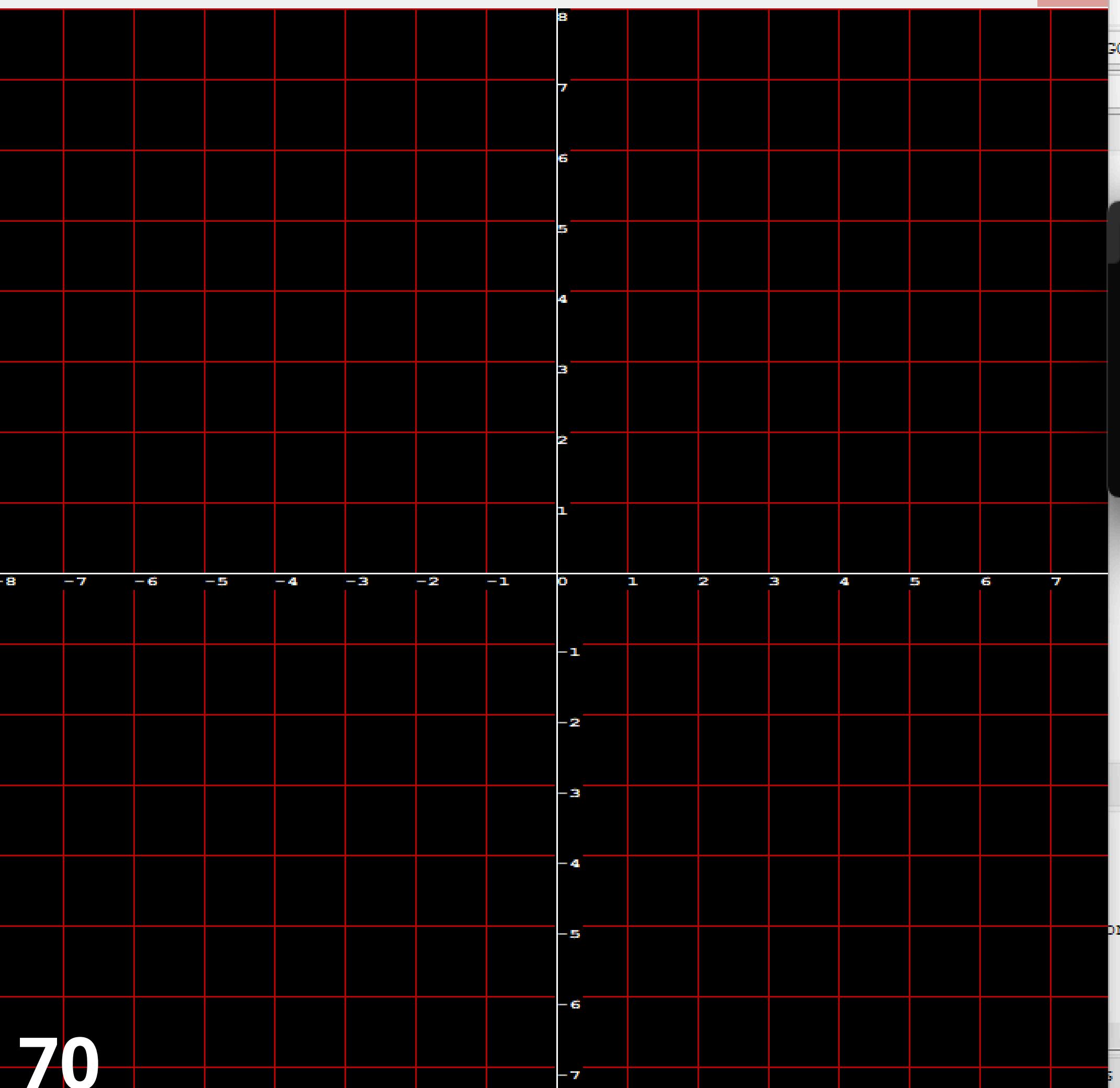






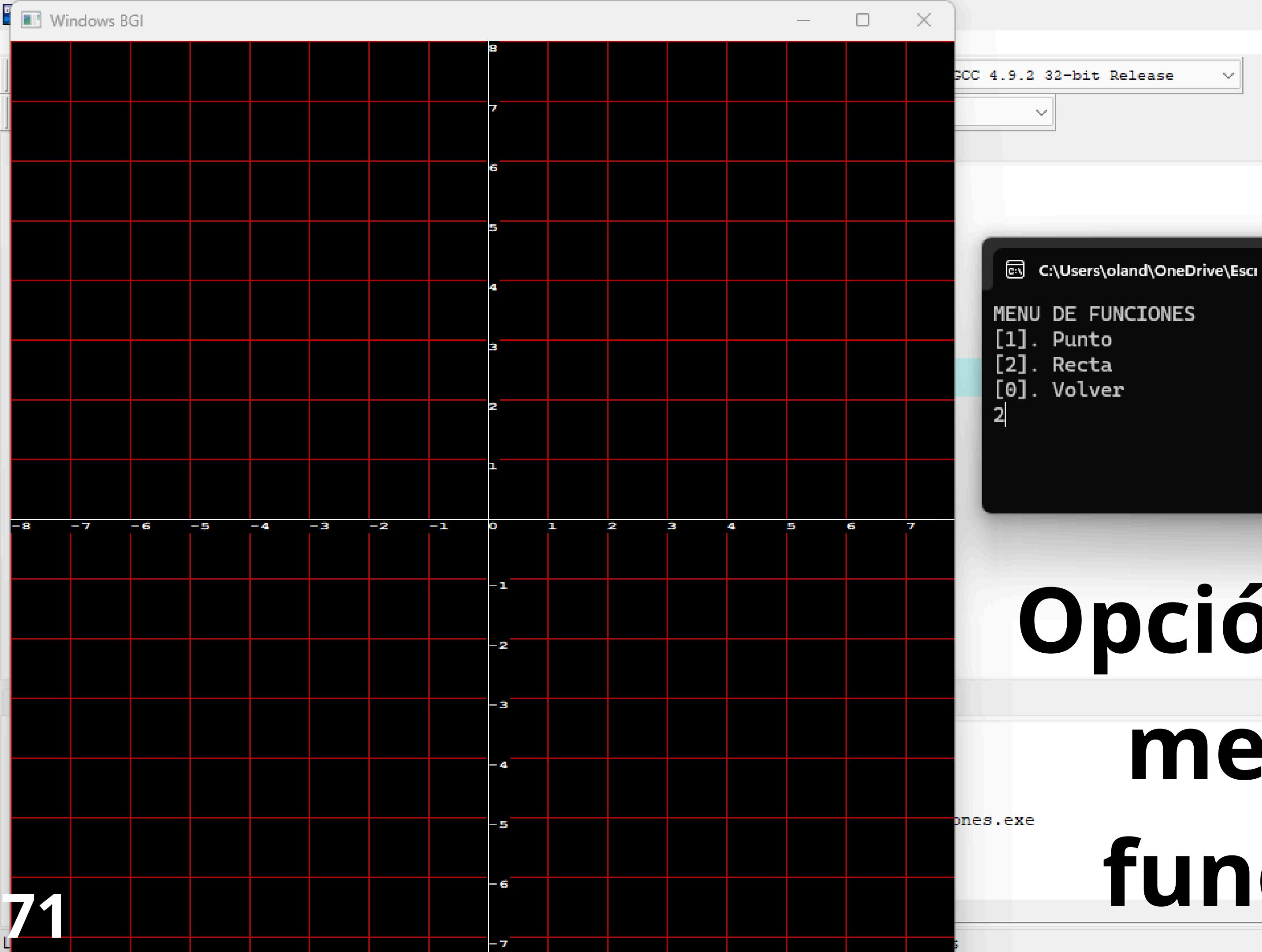
```
C:\Users\oland\OneDrive ANDO EL METODO GRAFICO
[1].Iniciar el modo grafico
[2].Pintar grafica
[3].Limpiar ventana
[4].Ingresar sistema de ecuaciones
[0].Salir
3
```

Opción [3] para
limpiar la ventana
gráfica

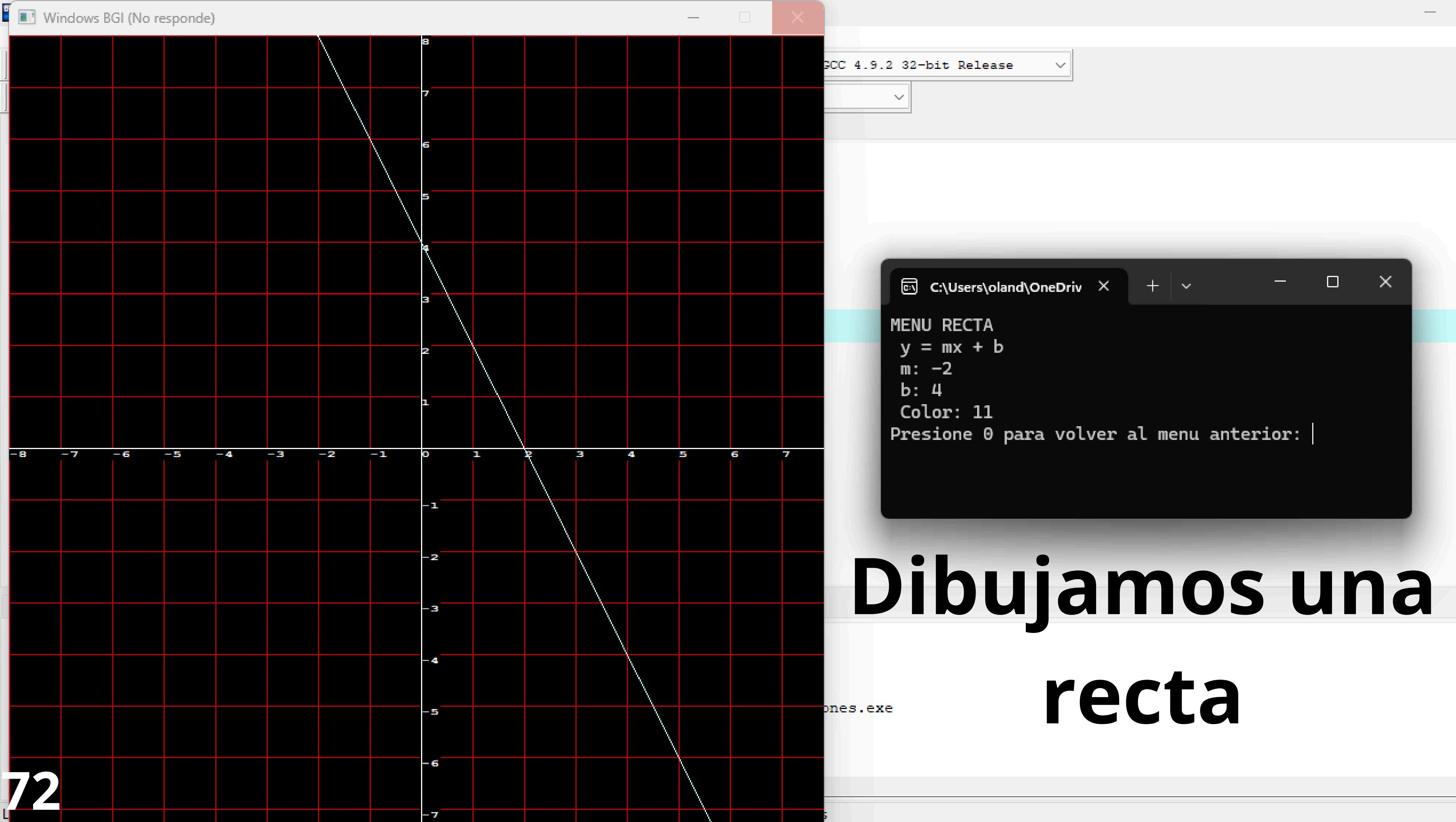


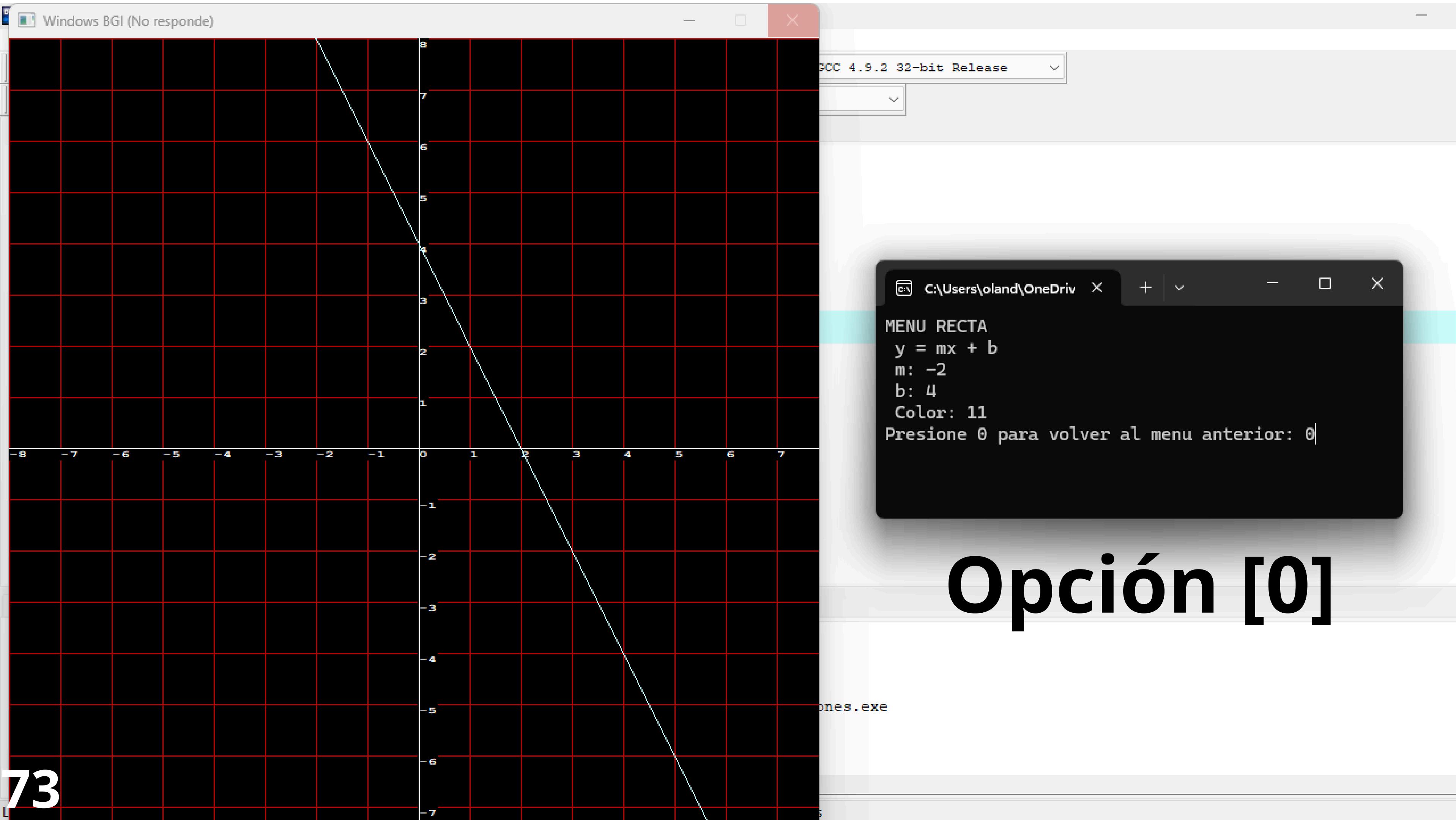
```
GCC 4.9.2 32-bit Release
C:\Users\oland\OneDrive\Escritorio\ex1>+<
SOLUCIONADOR DE UN SISTEMA DE ECUACIONES 2X2 USANDO EL METODO
[1].Iniciar el modo grafico
[2].Pintar grafica
[3].Limpiar ventana
[4].Ingresar sistema de ecuaciones
[0].Salir
2
```

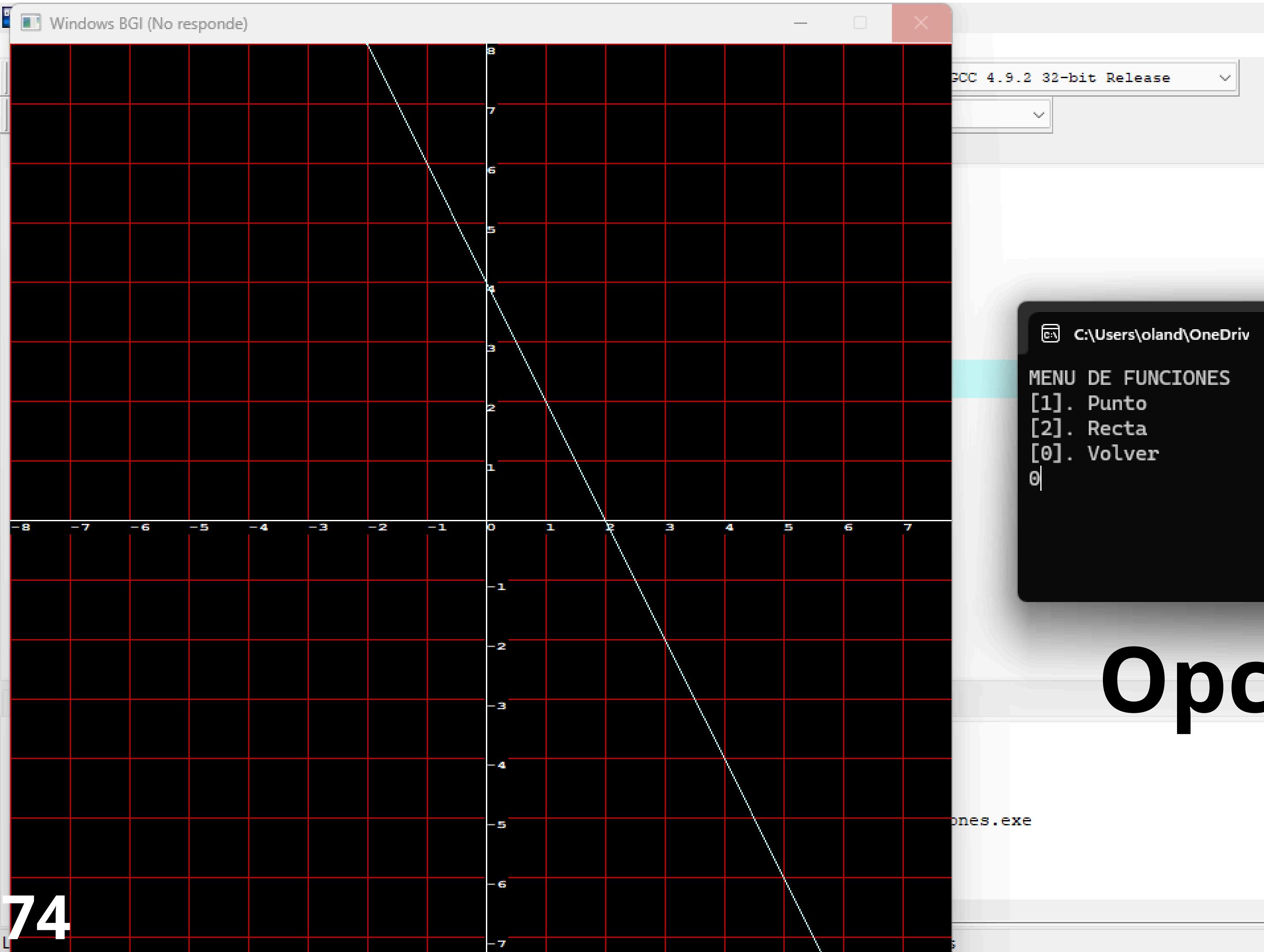
Opción [2] del
menú principal



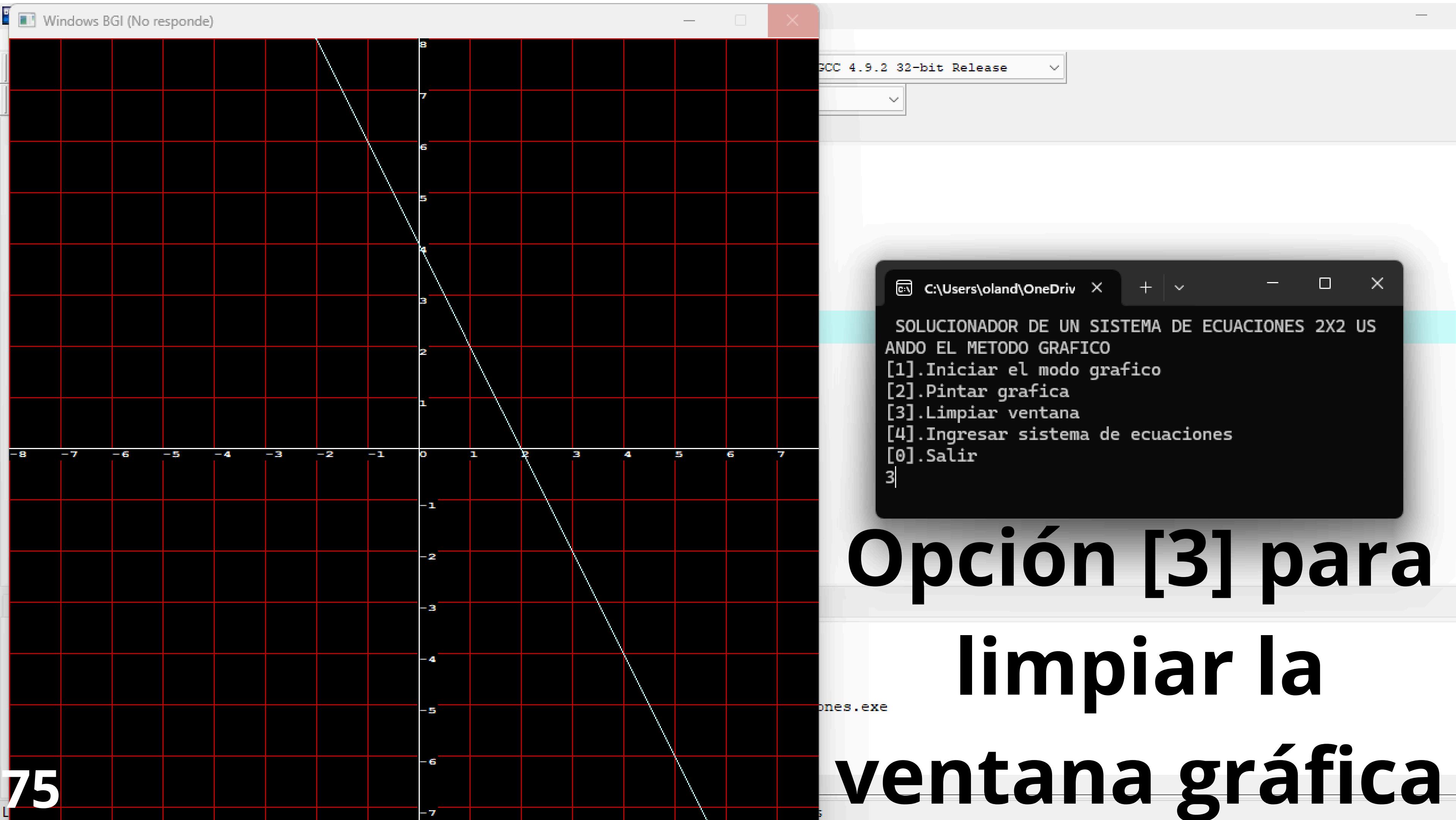
Opción [2] del
menú de
funciones

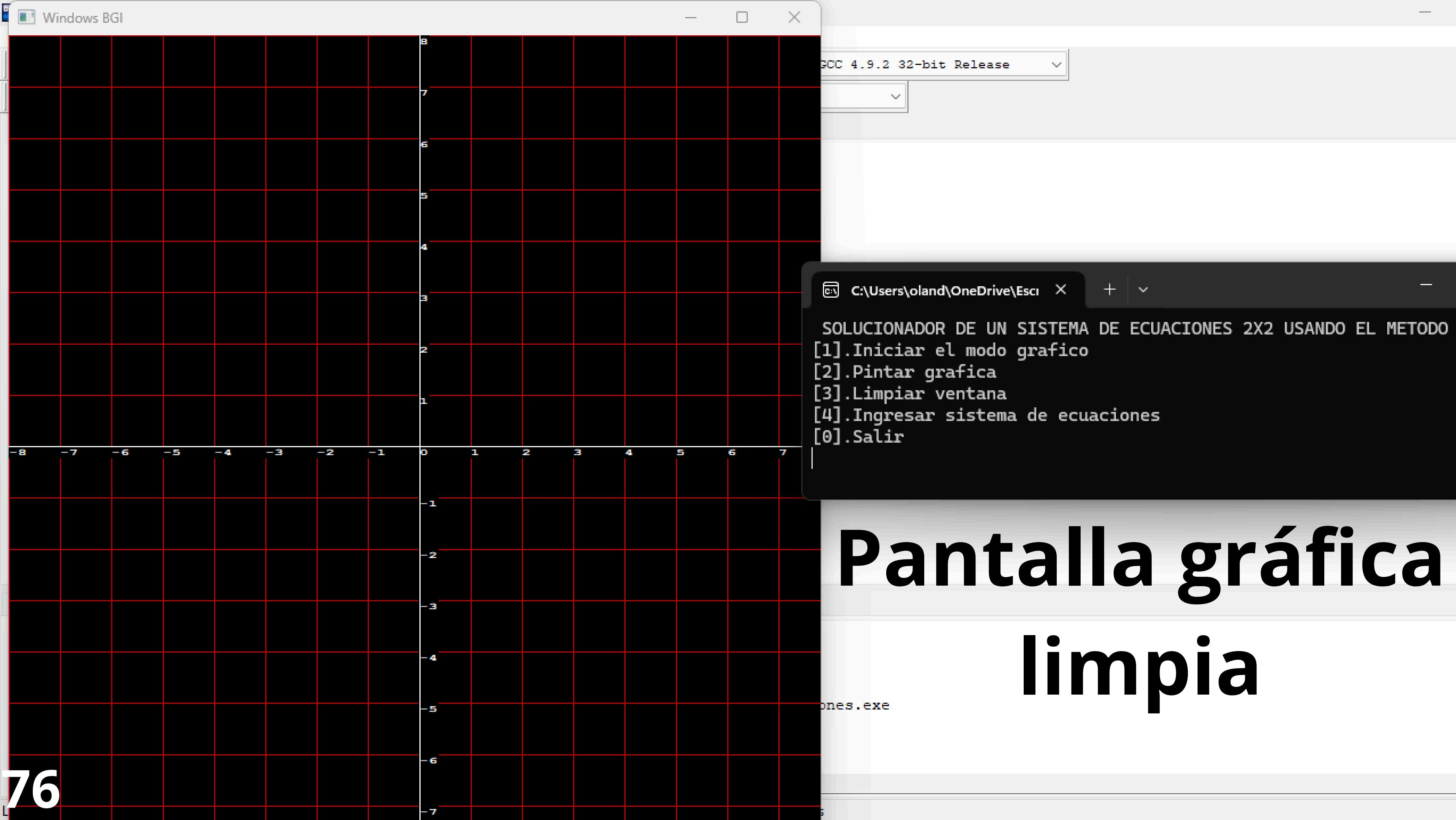






Opción [0]





Pantalla gráfica limpia

Definición

Método gráfico

Sistema de ecuaciones

$$\begin{cases} x - y = -1 \\ 2x - y = 1 \end{cases}$$

Despejamos y de la ecuación 1 Despejamos y de la ecuación 2

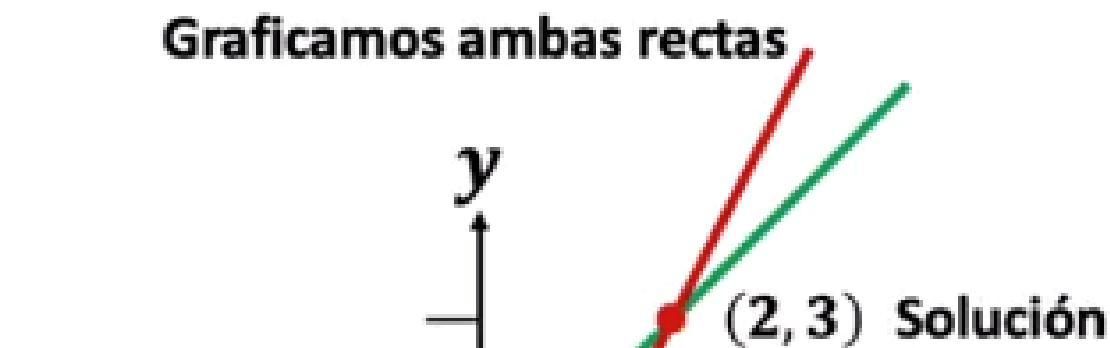
$$y = x + 1$$

$$y = 2x - 1$$

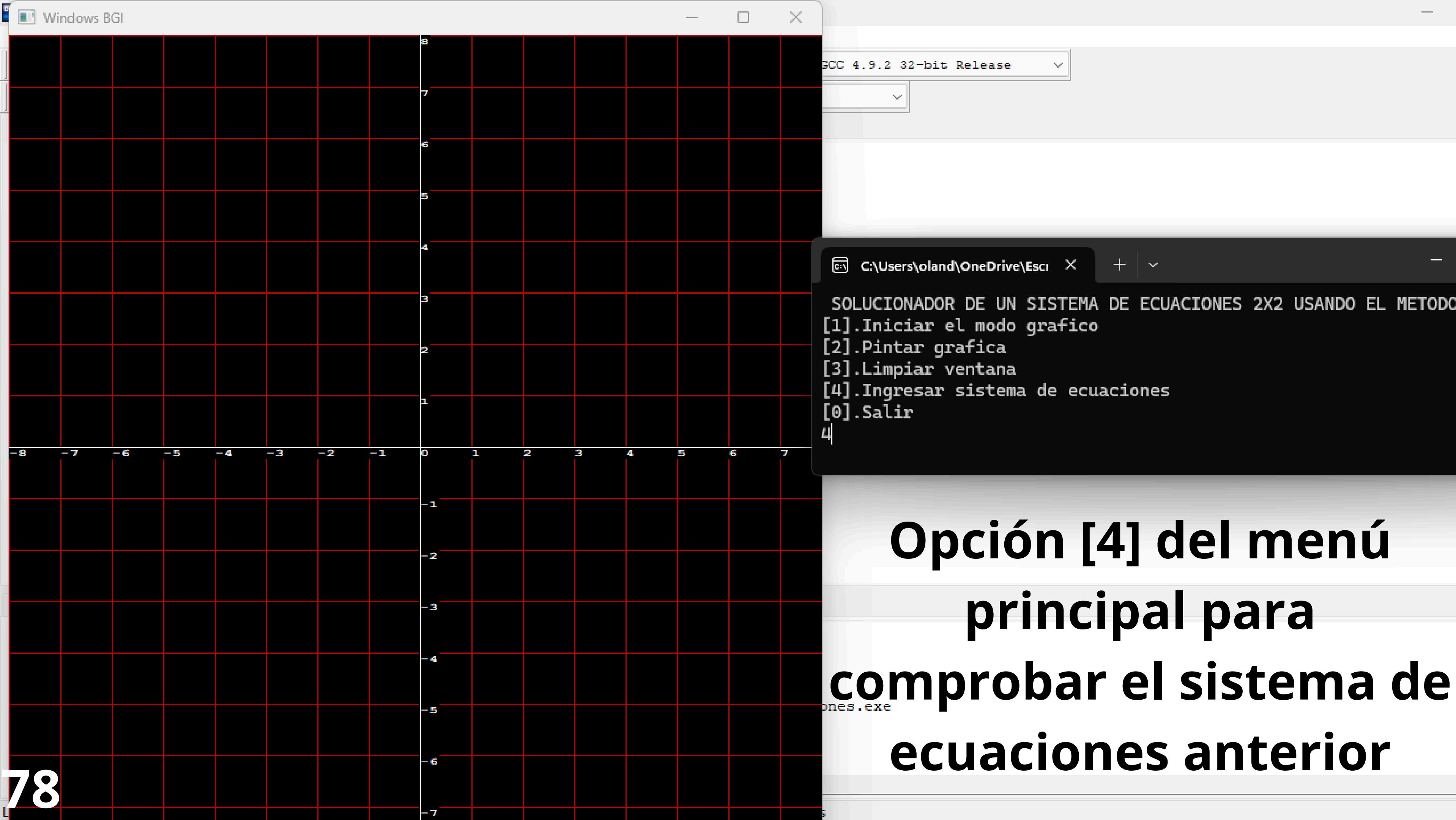
Tabulamos y encontramos los valores de y

x	y
-2	-1
-1	0
0	1
1	2

x	y
-1	-3
0	-1
1	1
2	3



A continuación verificaremos que la solución es (2,3) con nuestro programa; $x= 2$ e $y = 3$



Opción [4] del menú principal para comprobar el sistema de ecuaciones anterior

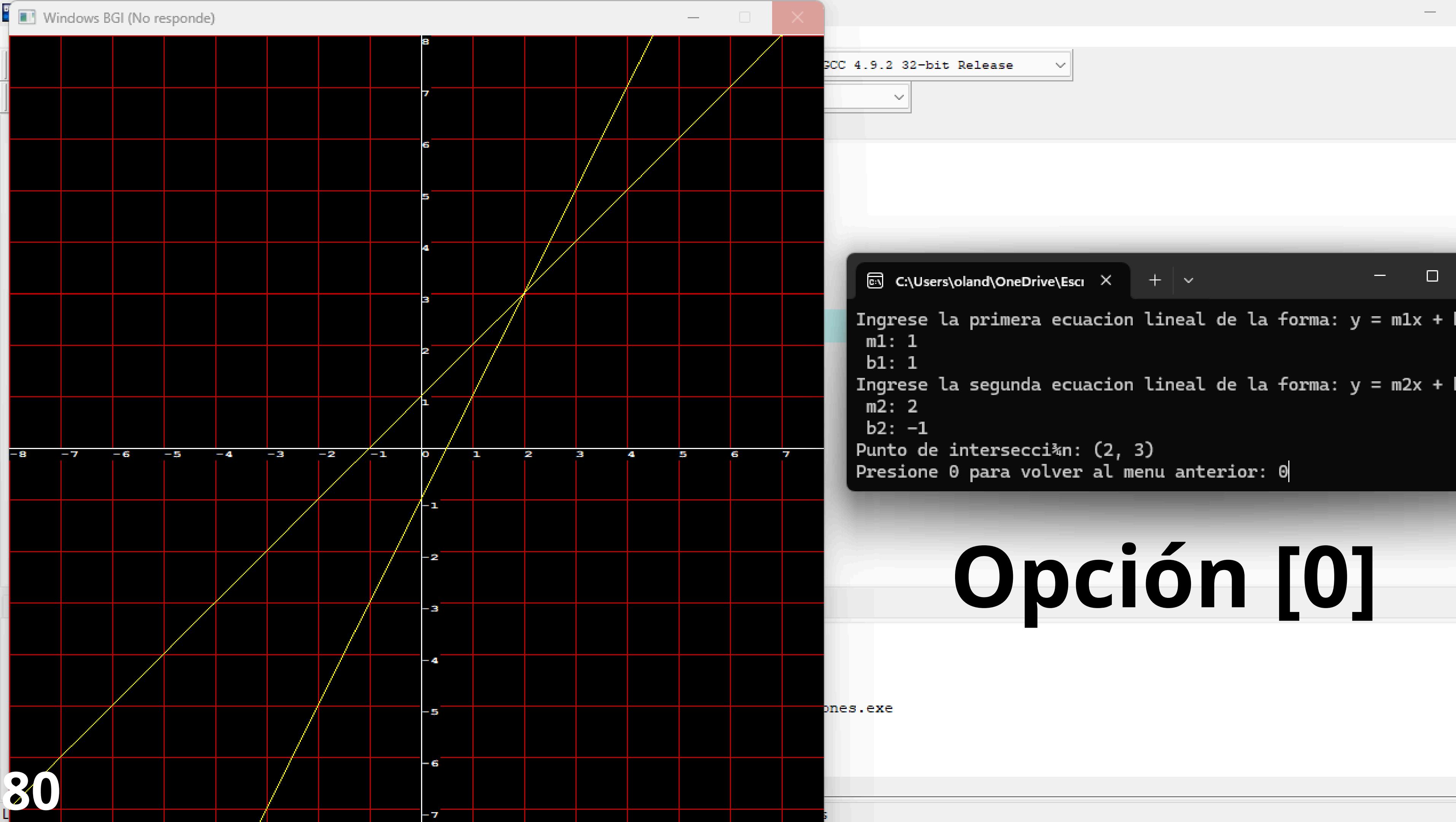


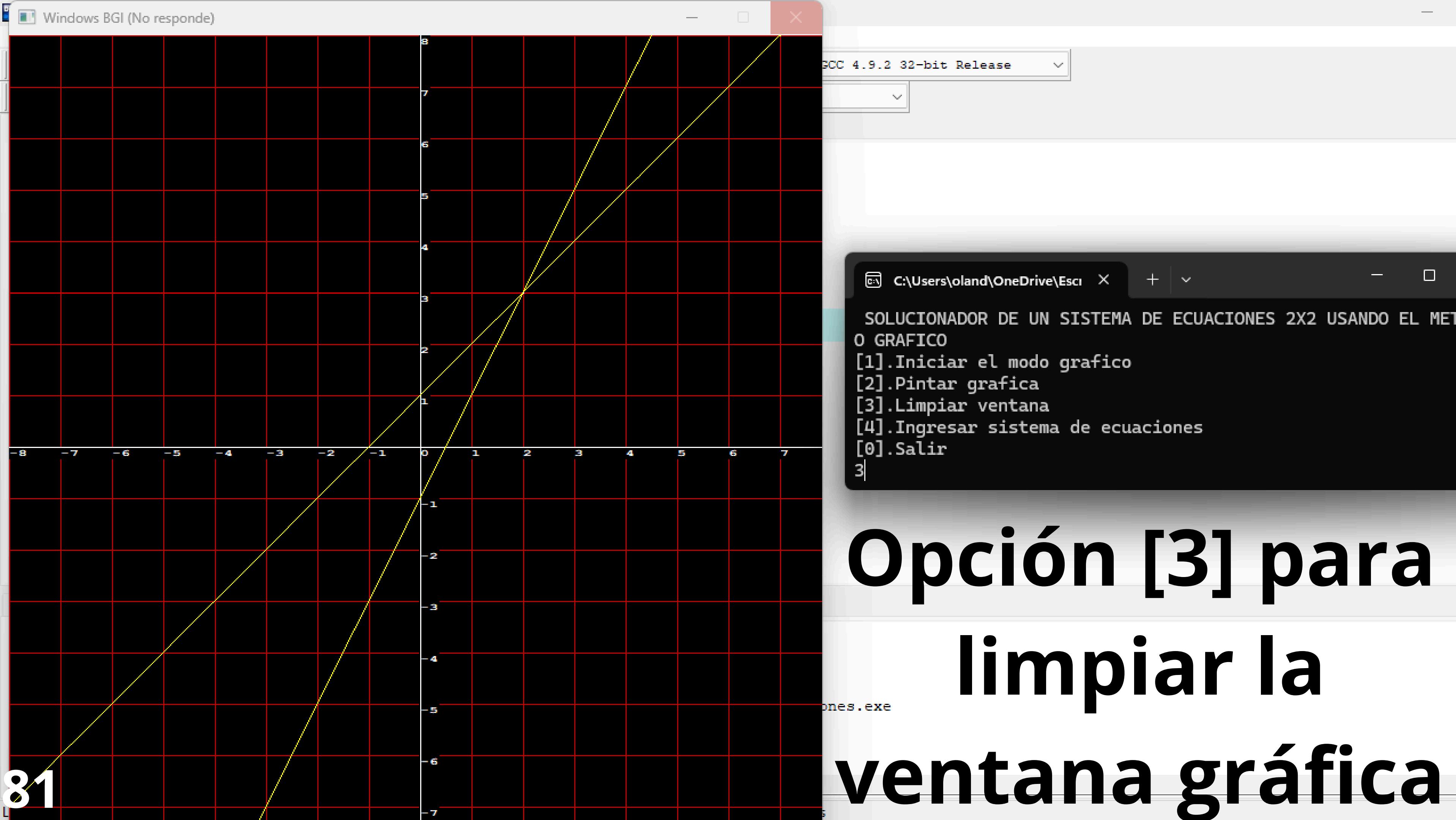
GCC 4.9.2 32-bit Release

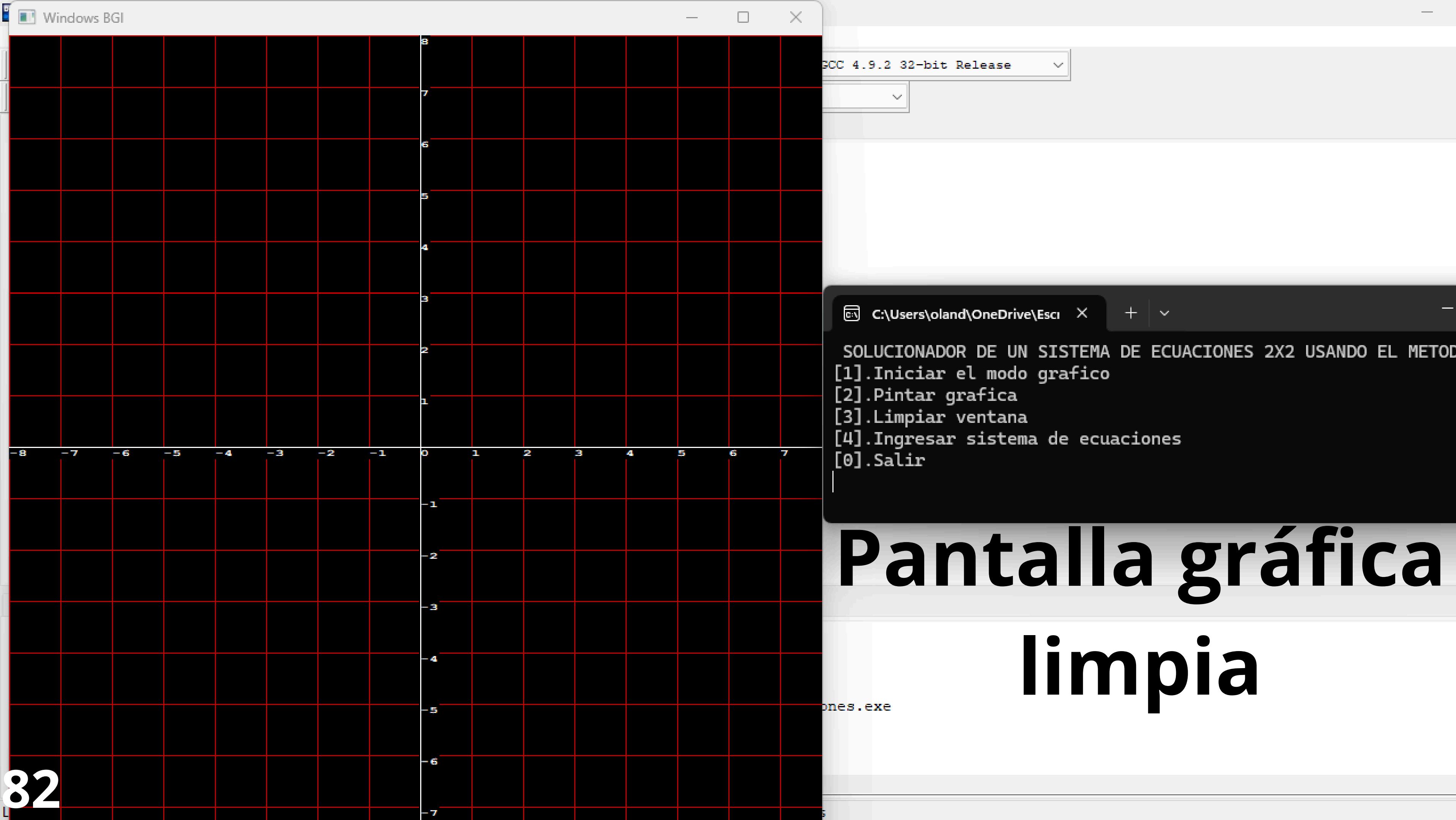
C:\Users\oland\OneDrive\Escritorio\Exe

```
Ingrese la primera ecuacion lineal de la forma: y = m1x + b  
m1: 1  
b1: 1  
Ingrese la segunda ecuacion lineal de la forma: y = m2x + b  
m2: 2  
b2: -1  
Punto de intersección: (2, 3)  
Presione 0 para volver al menu anterior:
```

**Ingresamos los valores
de las rectas de acuerdo
con el sistema 2x2 y
podemos ver que la sol es
correcta**







El presente sistema de ecuaciones 2x2 tiene dicha solución en decimales, la cual nuevamente será comprobada por nuestro programa.

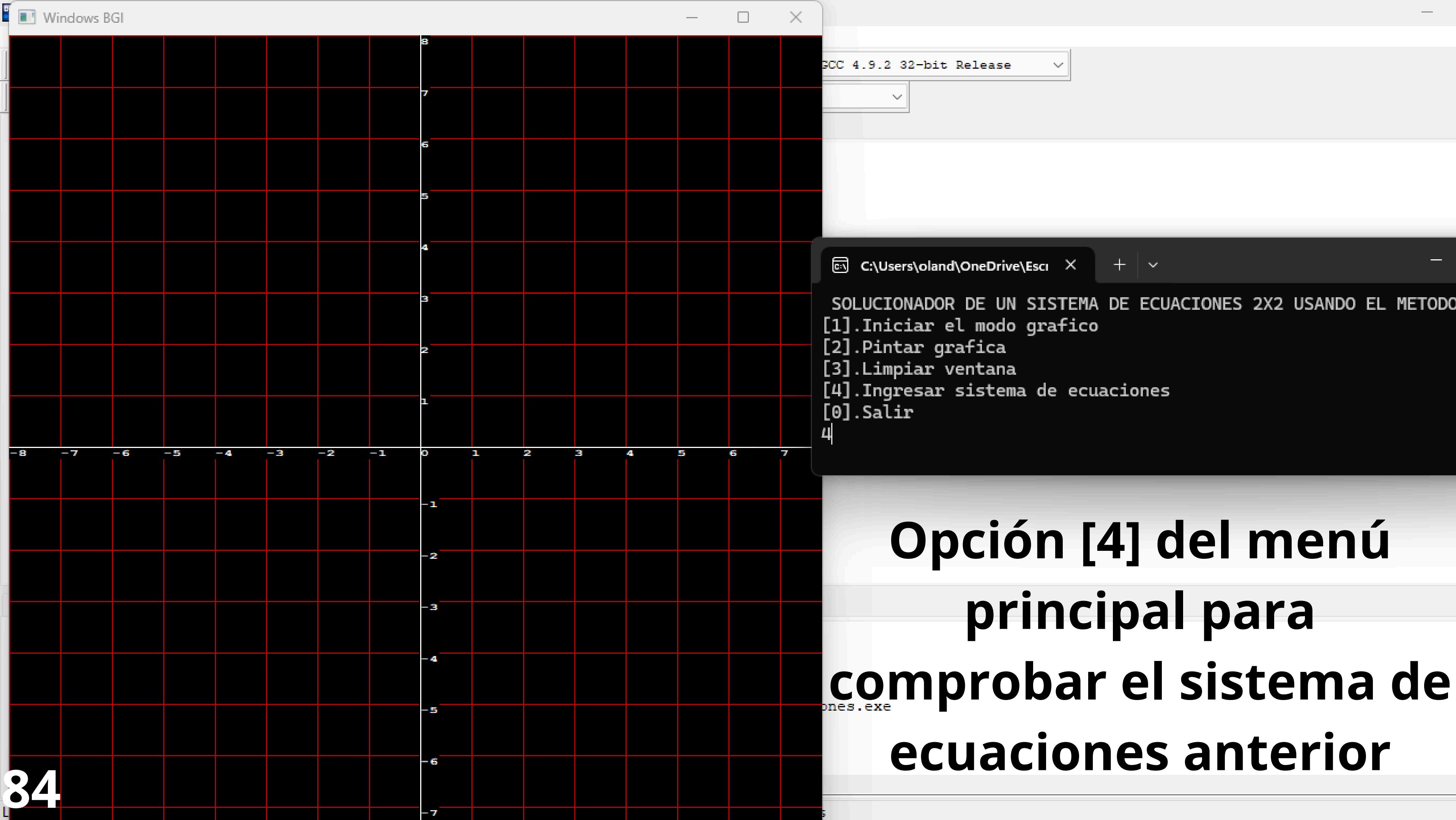
$$\left\{ \begin{array}{l} 2x + 3y = 7 \\ 4x - 5y = -3 \end{array} \right.$$

$$y = -0.667x + 2.33$$

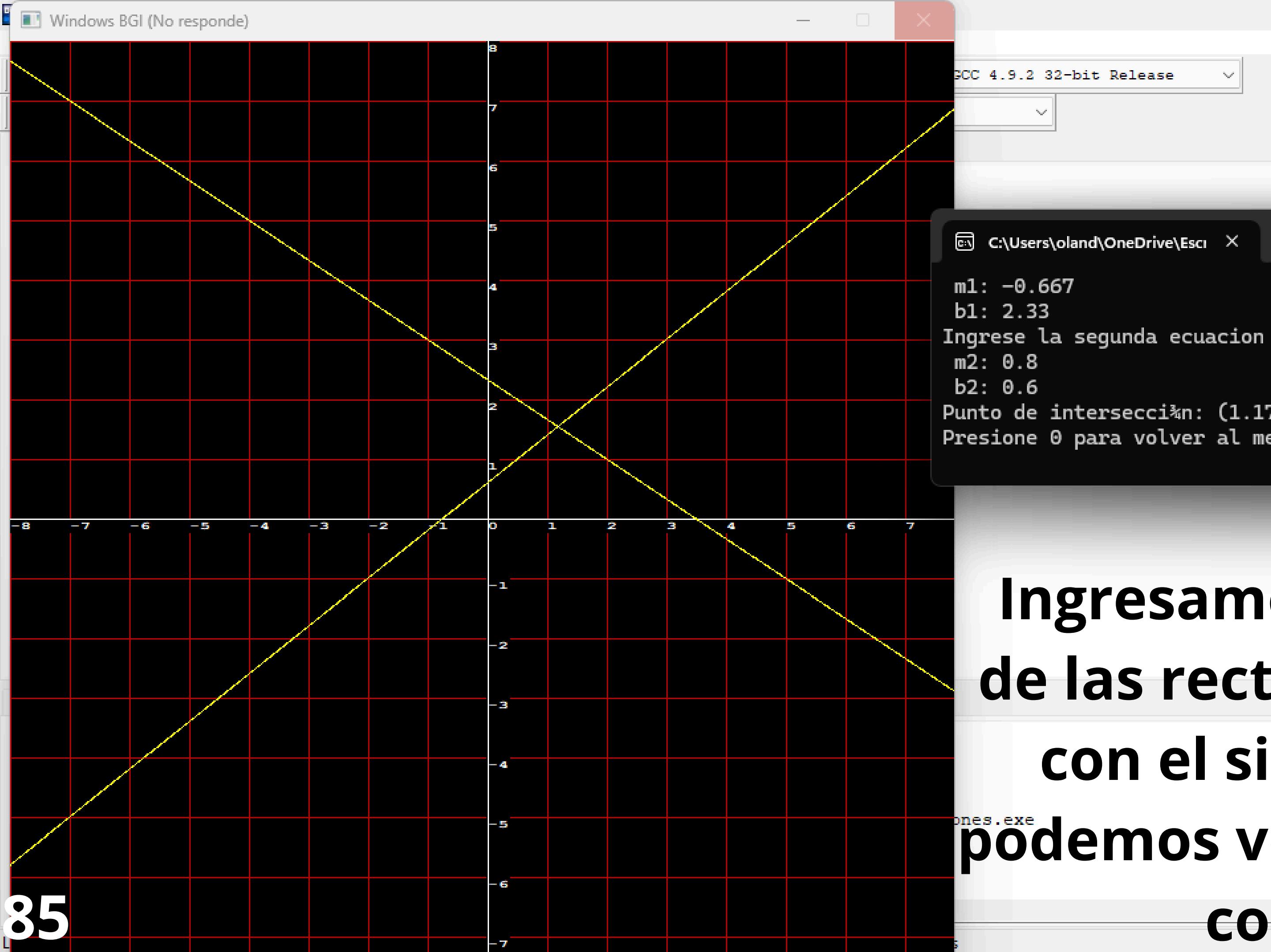
$$y = 0.8x + 0.6$$

$$y = 17/11 = 1.54545454545$$

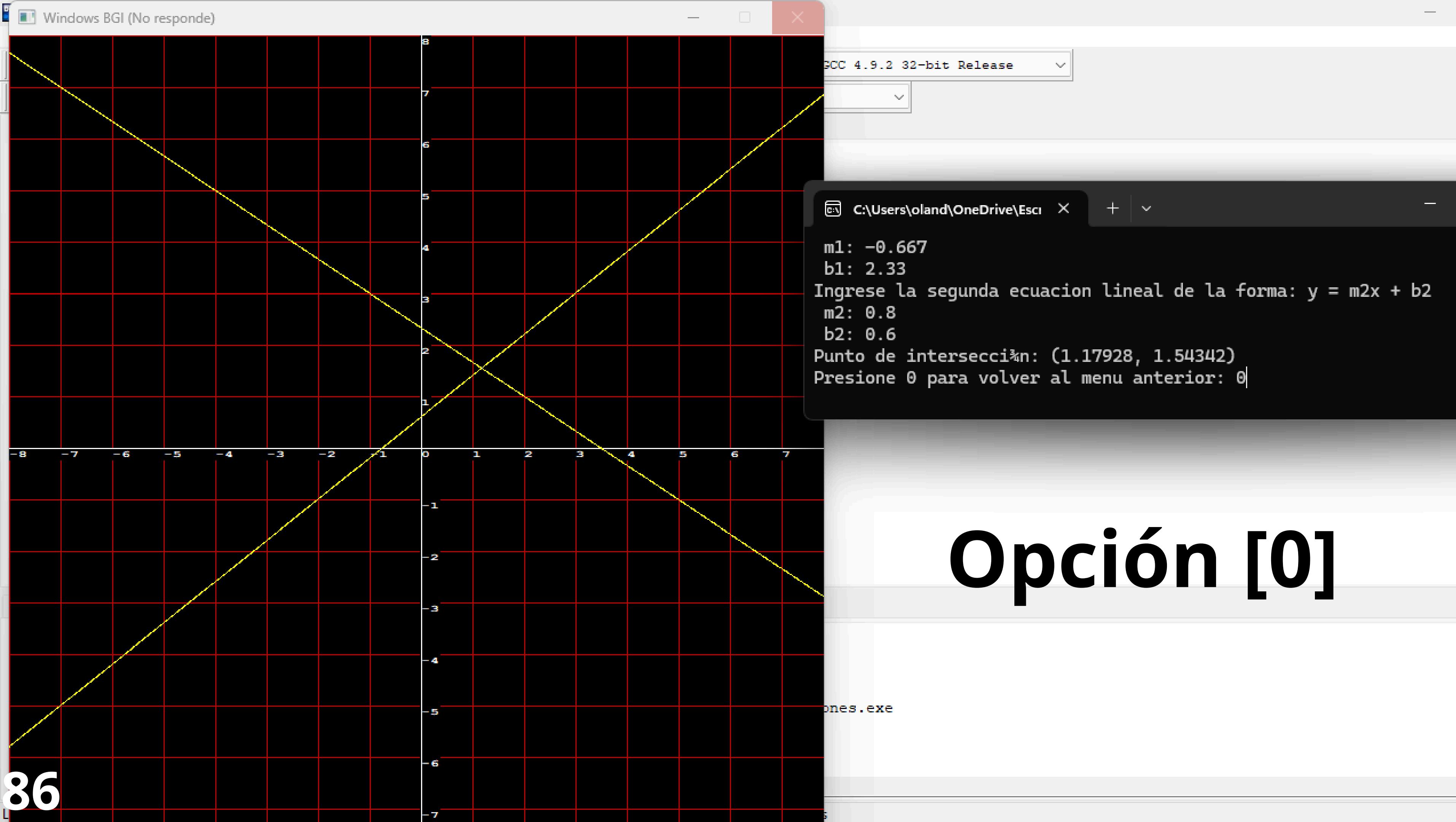
$$x = 13/11 = 1.18181818182$$

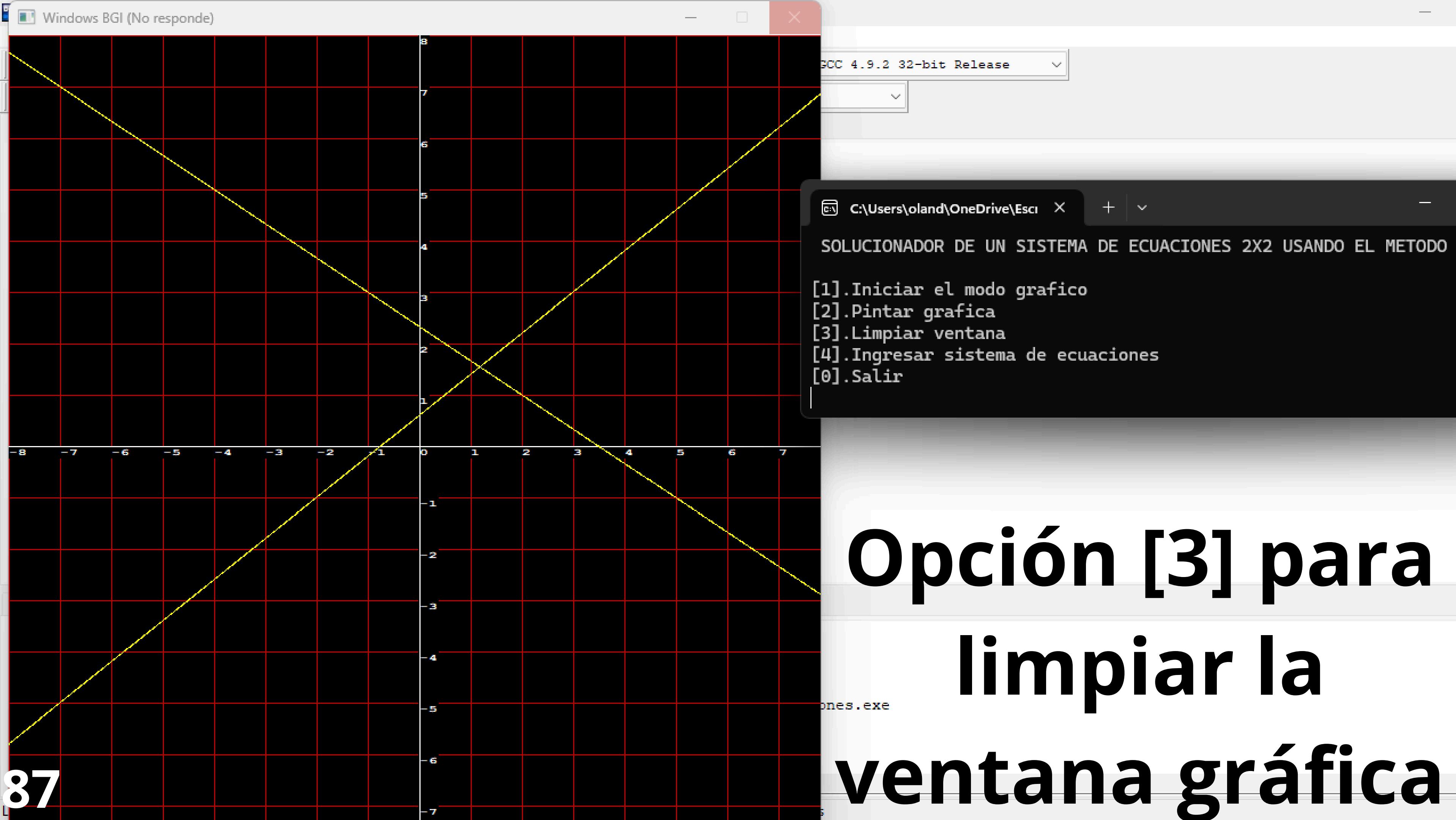


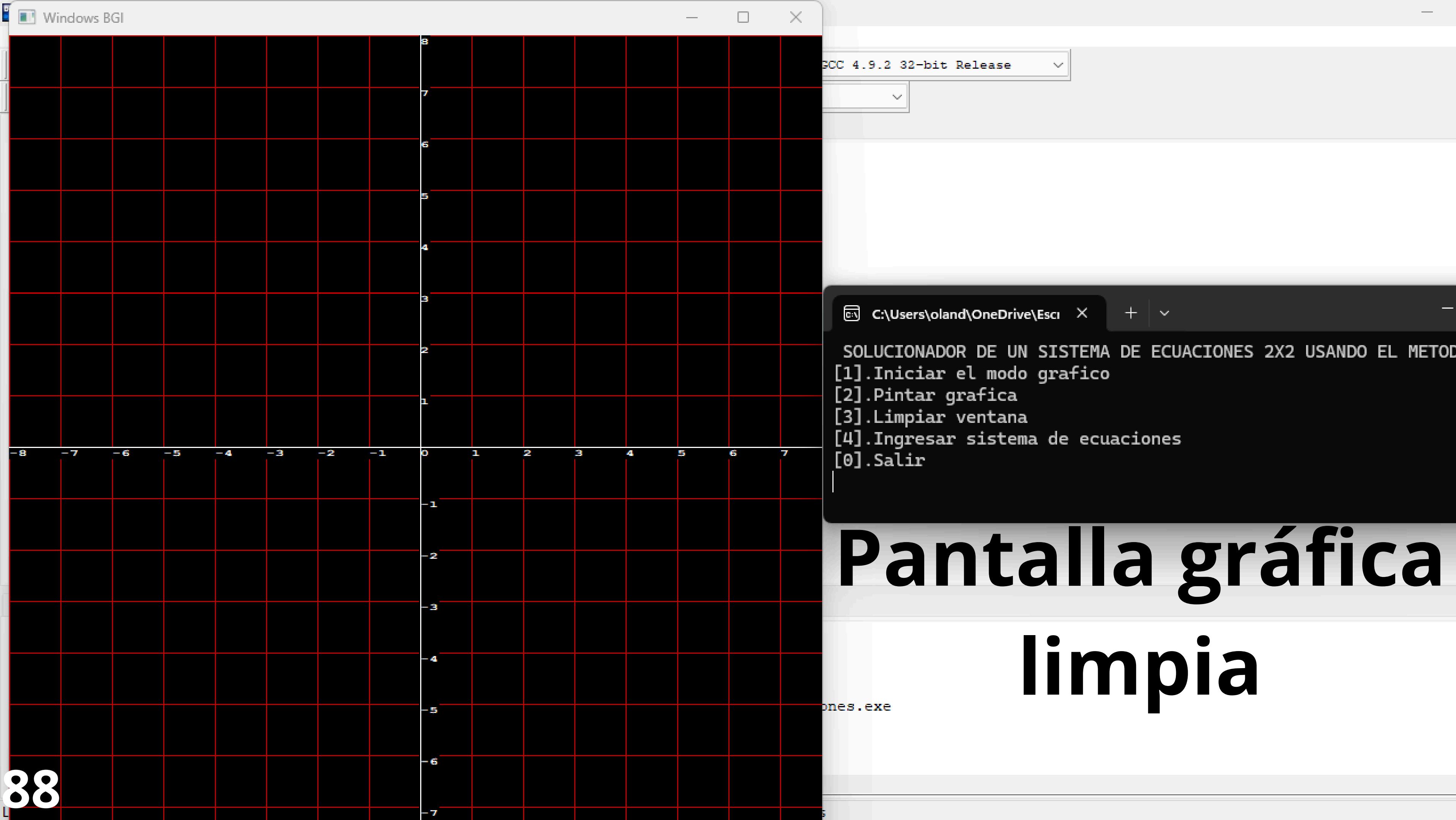
Opción [4] del menú principal para comprobar el sistema de ecuaciones anterior



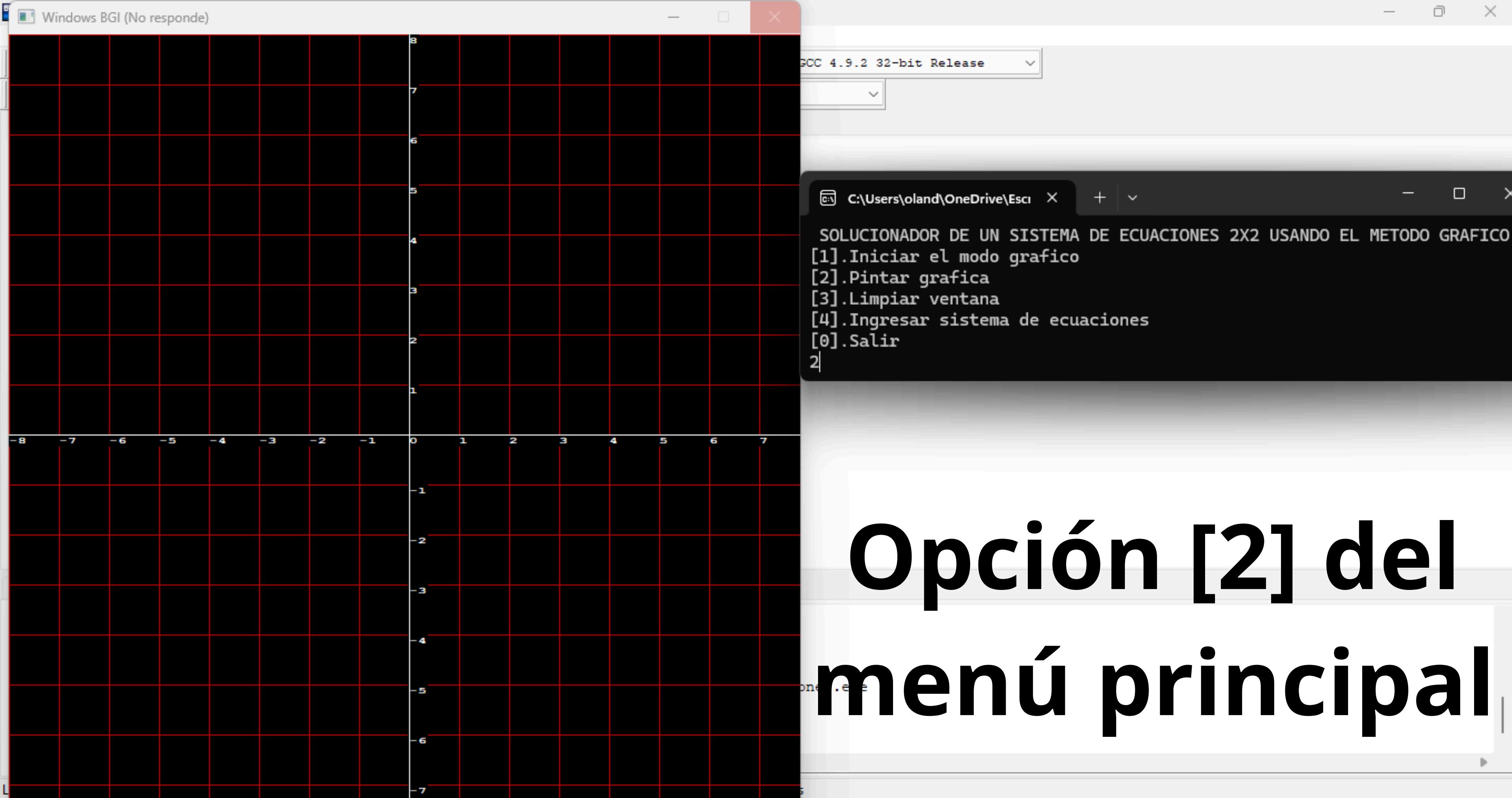
Ingresamos los valores
de las rectas de acuerdo
con el sistema 2×2 y
podemos ver que la sol es
correcta

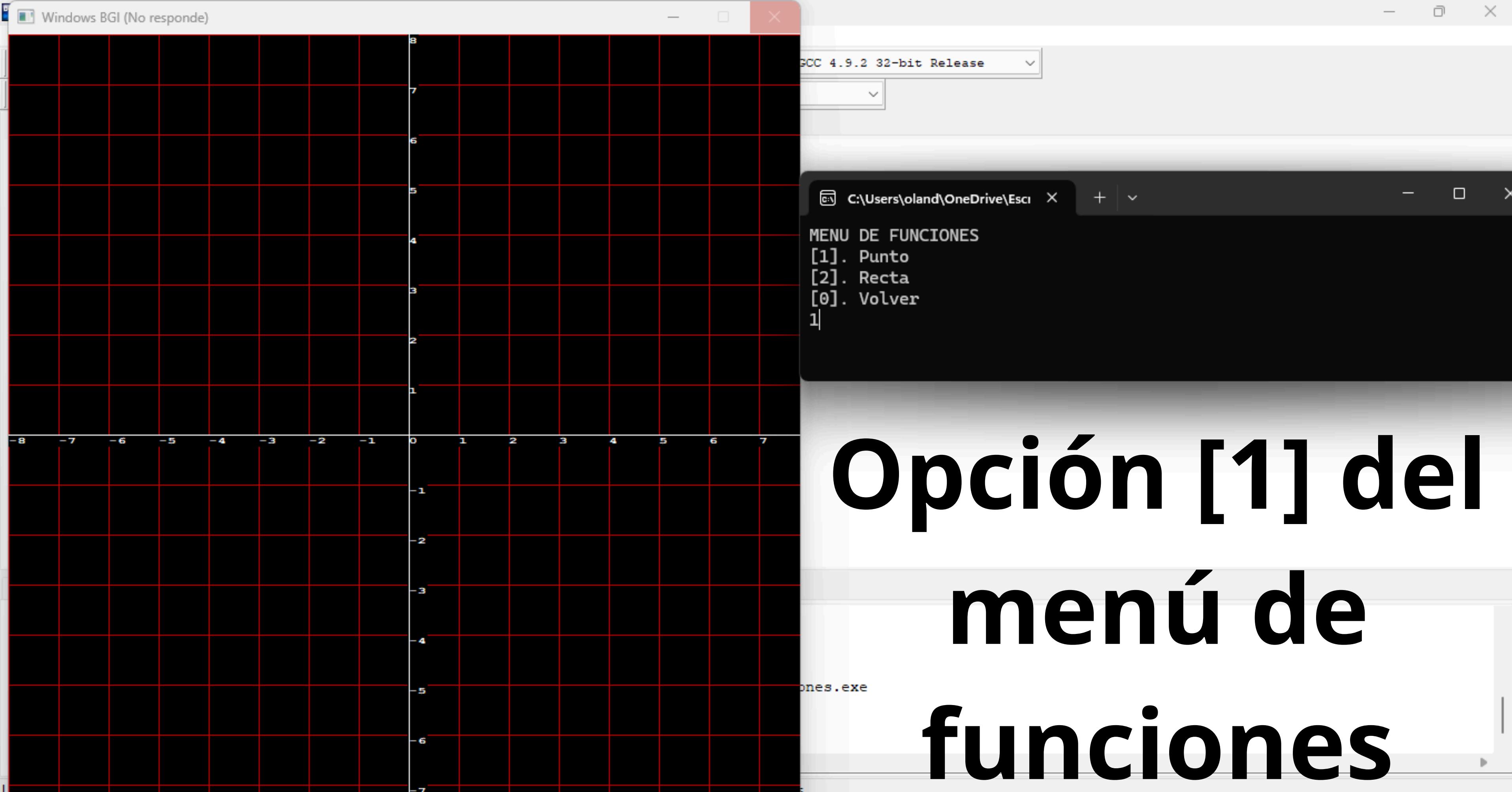




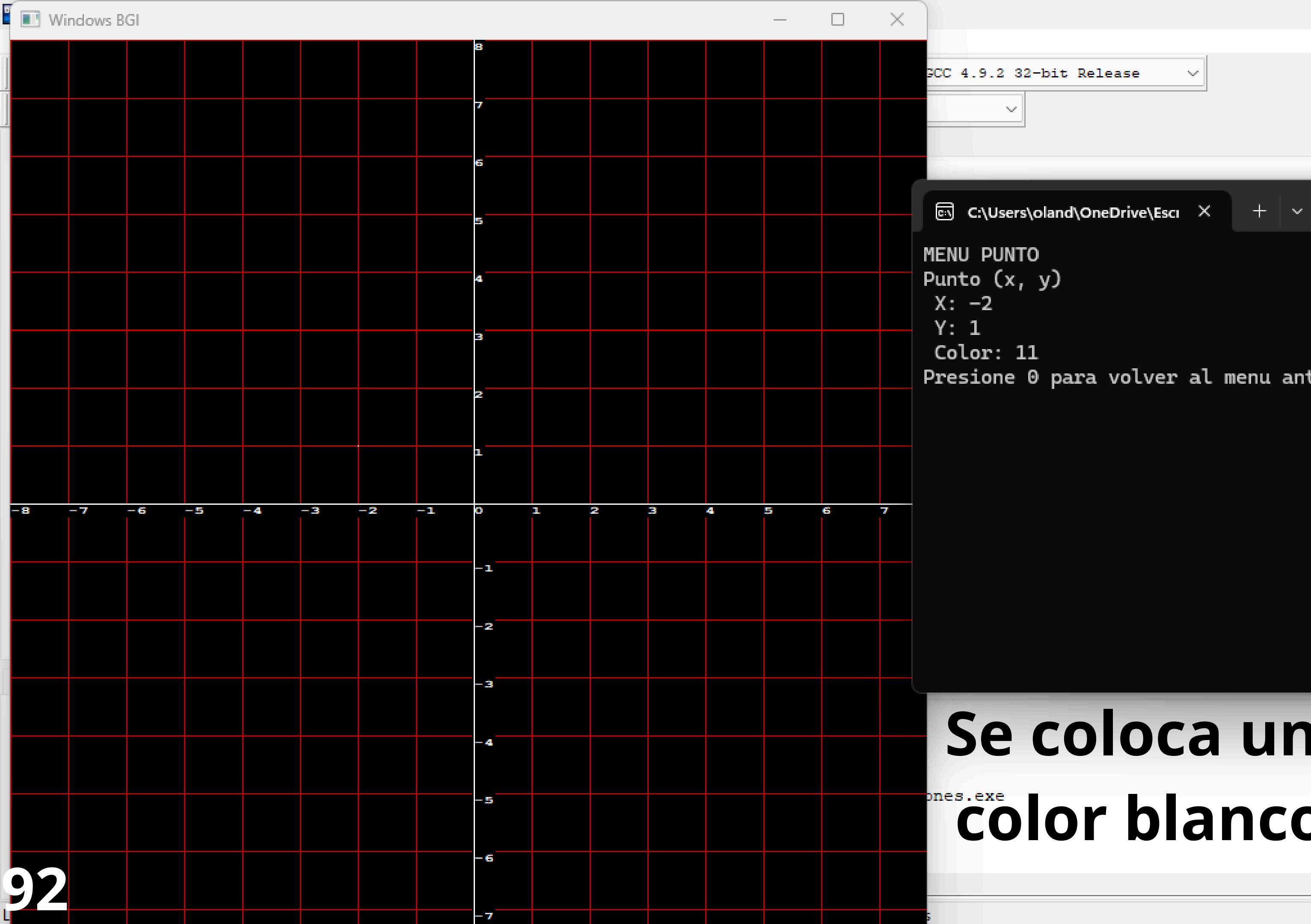


A continuación se reportará un caso en donde se muestren 3 gráficas y 3 puntos en el plano cartesiano

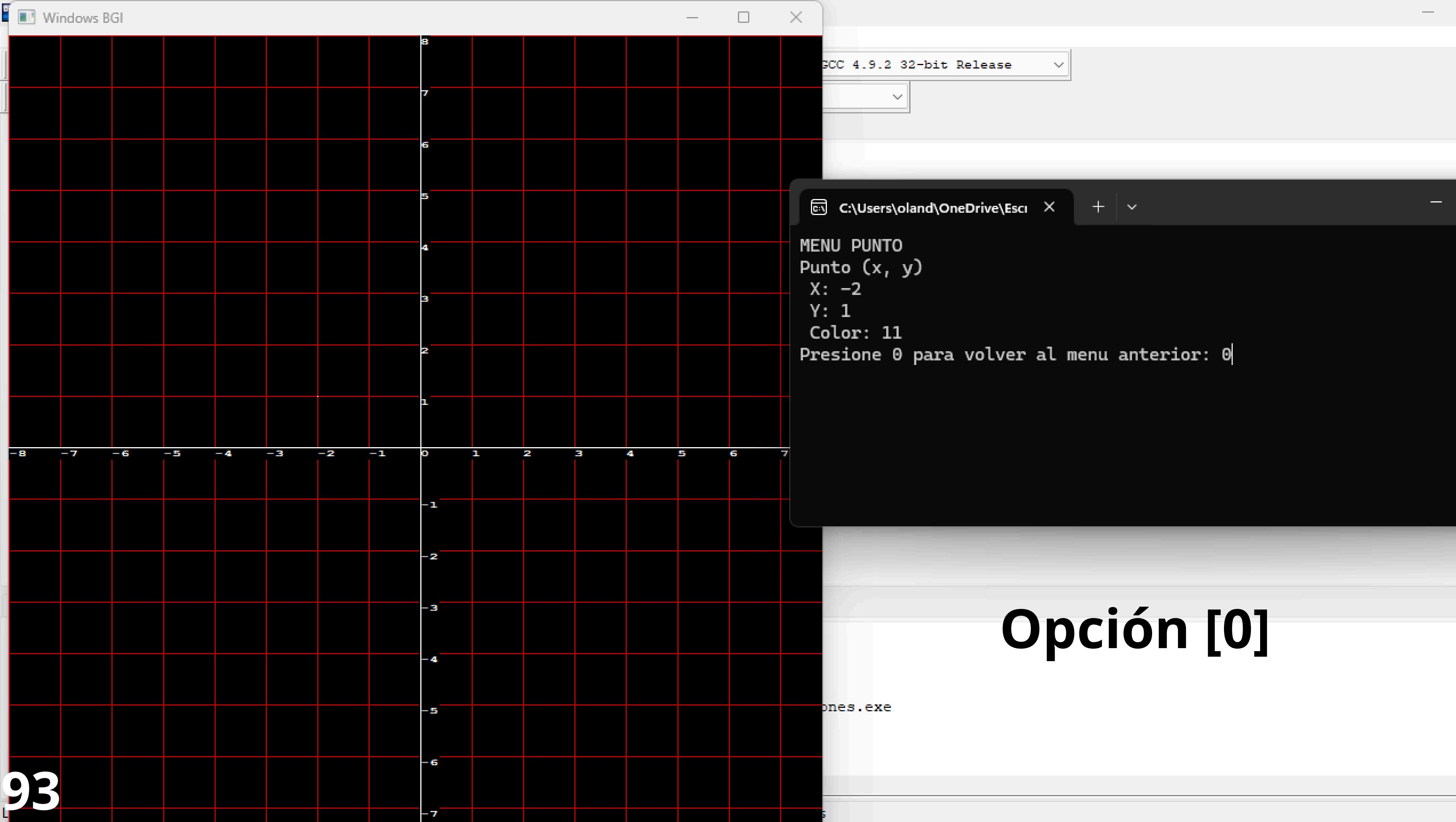


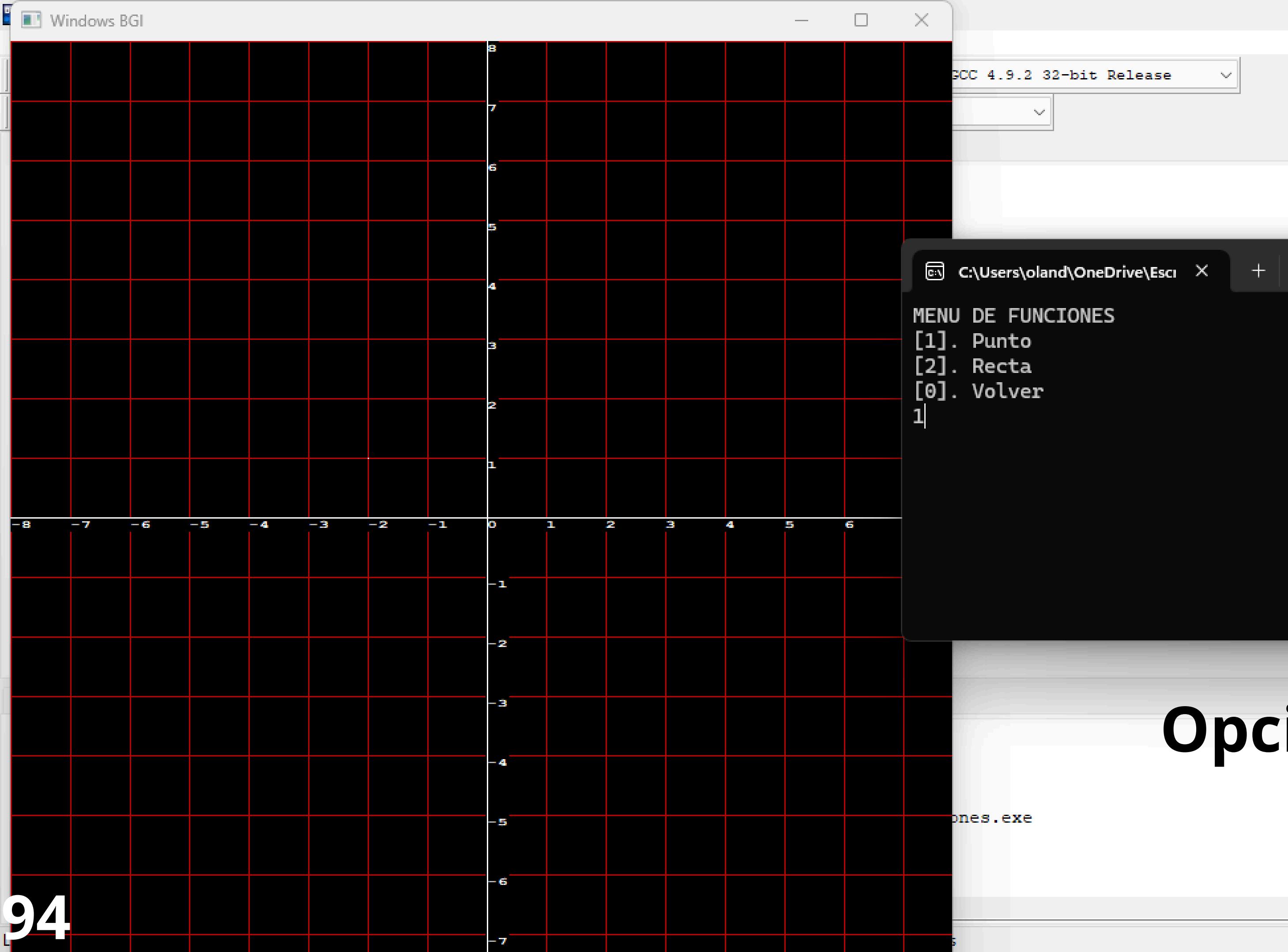


Opción [1] del
menú de
funciones



Se coloca un punto de
color blanco en (-2, 1)

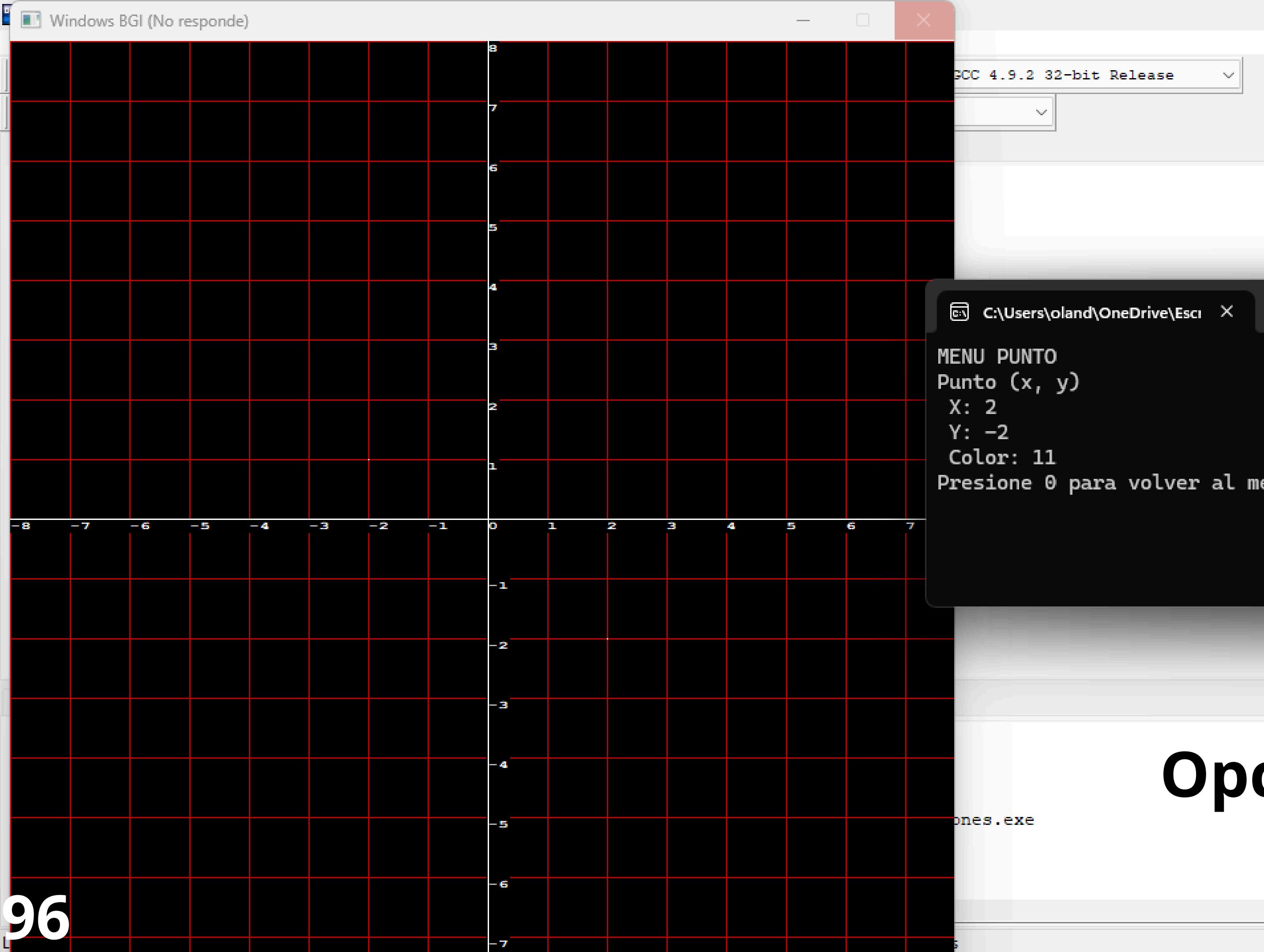




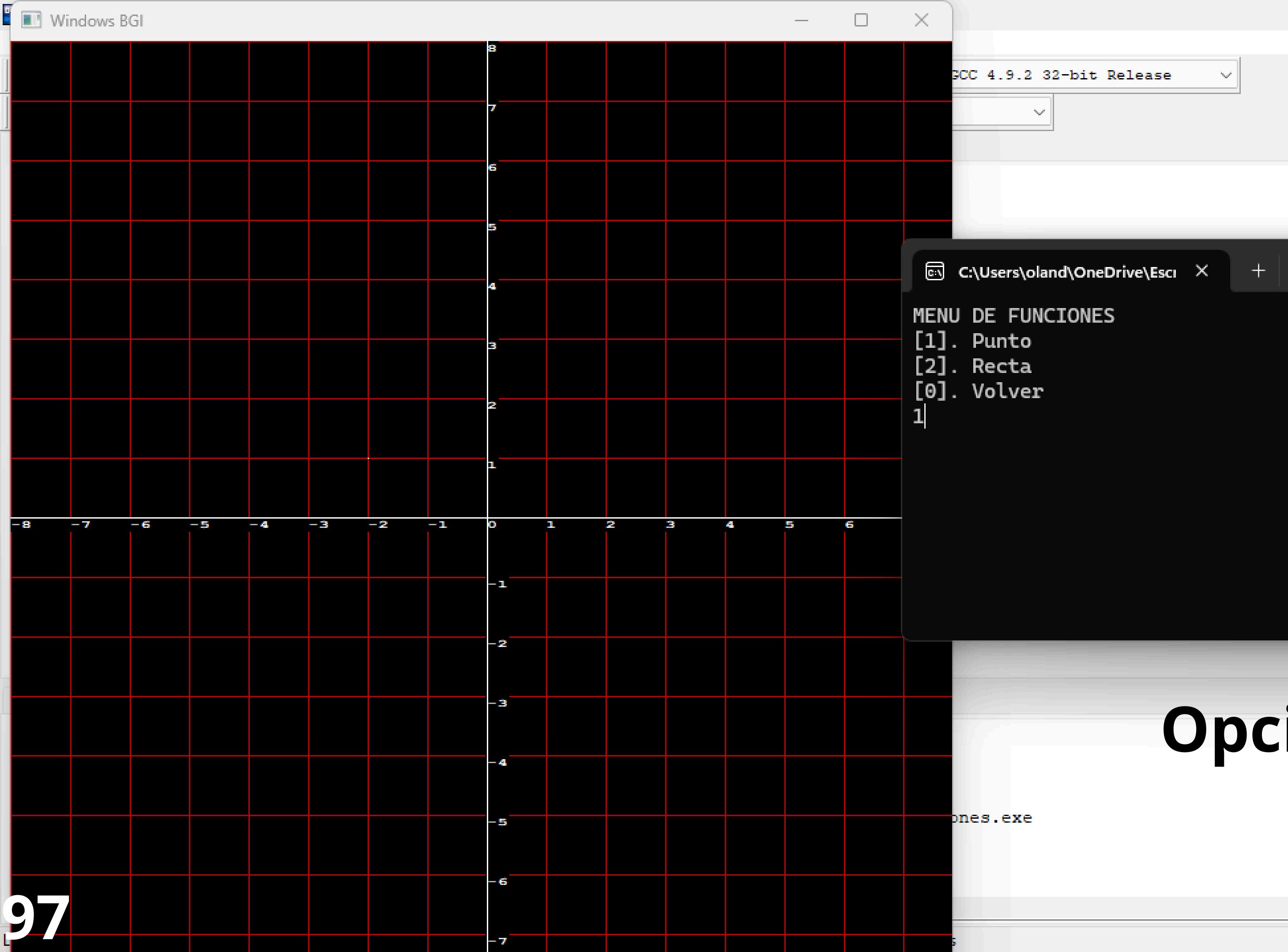
Opción [1]



**Se coloca un punto
de color blanco en
(2,-2)**



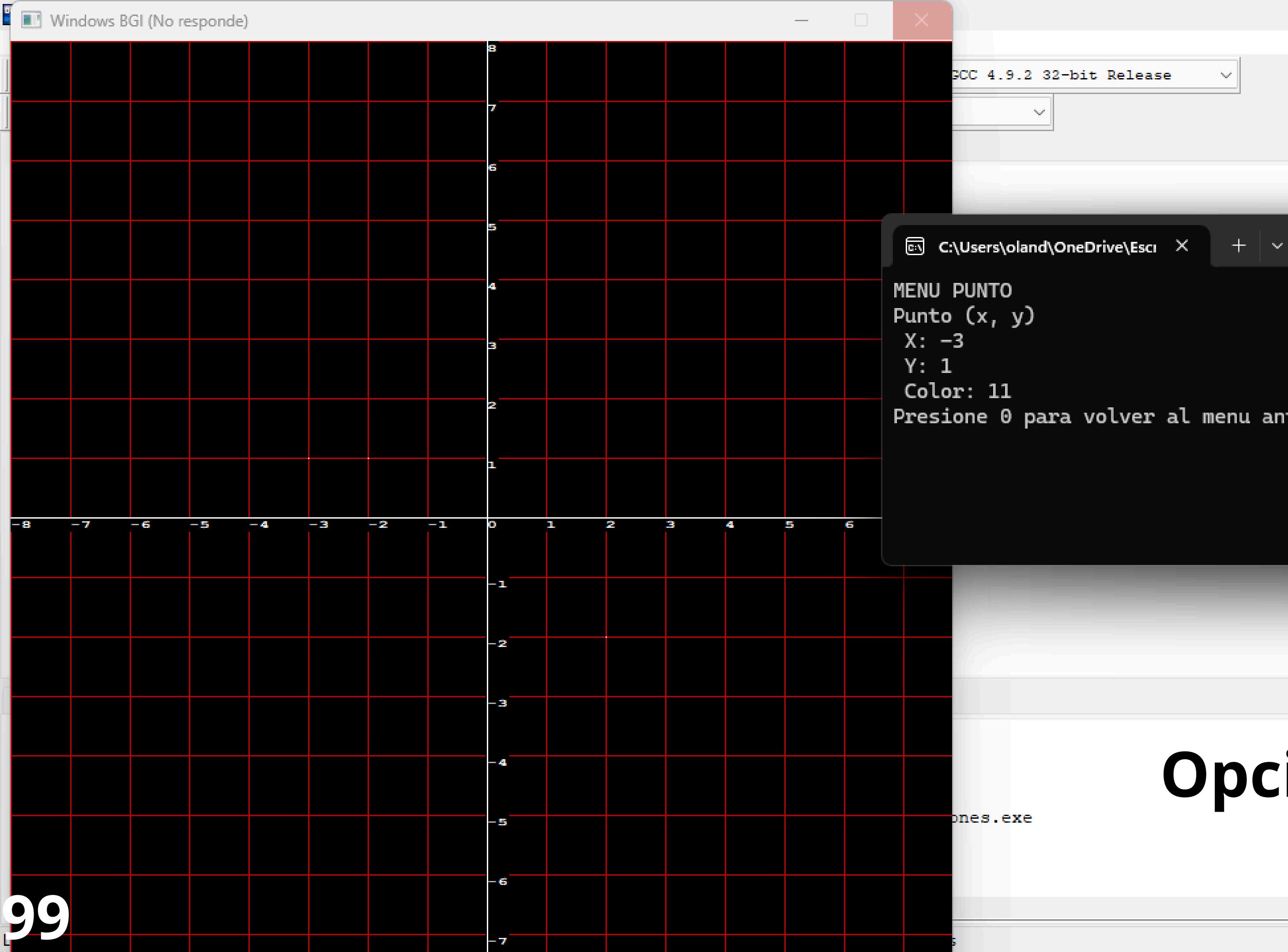
Opción [0]



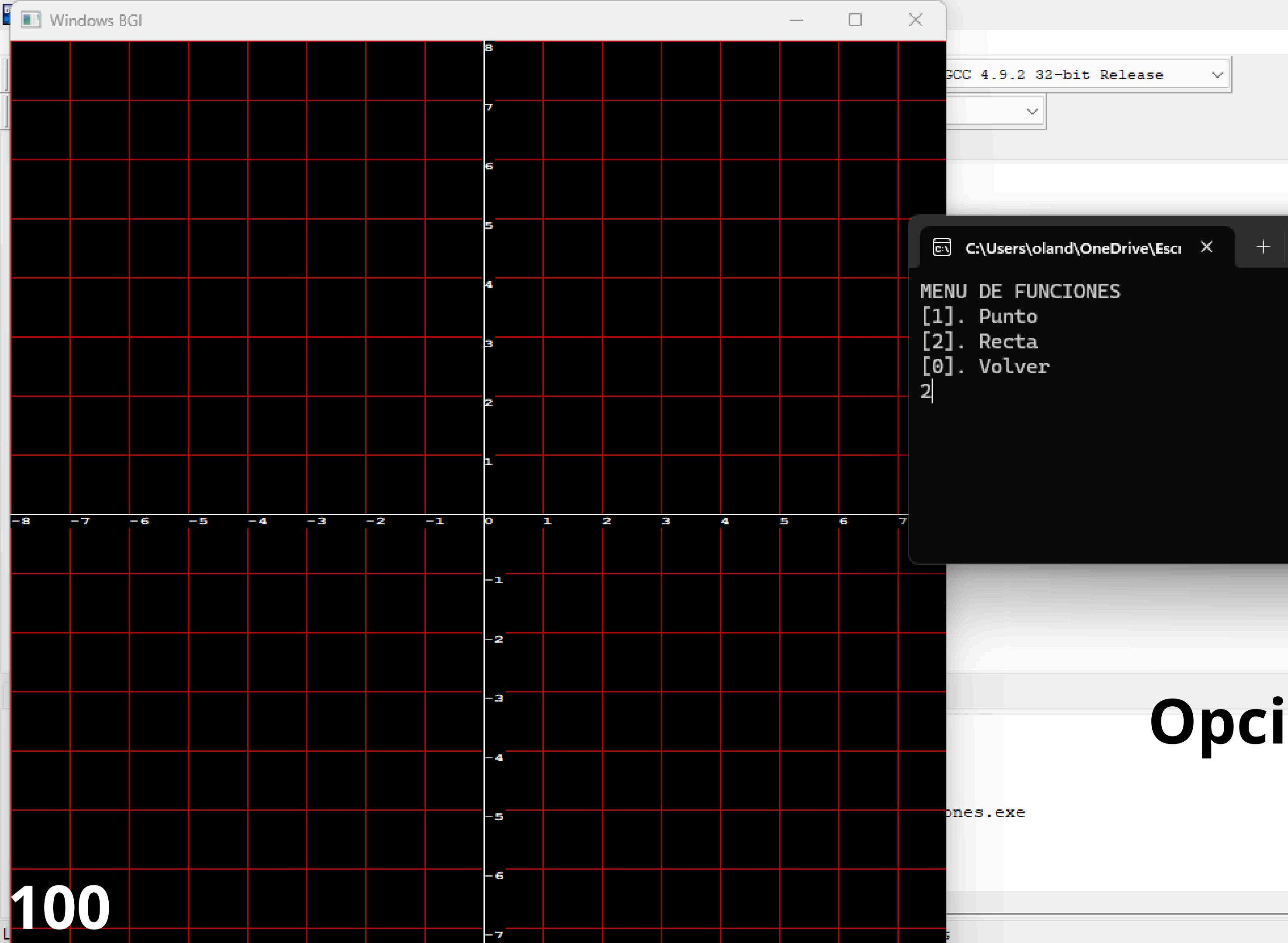
Opción [1]



**Se coloca un punto de
color blanco en (-3, 1)**



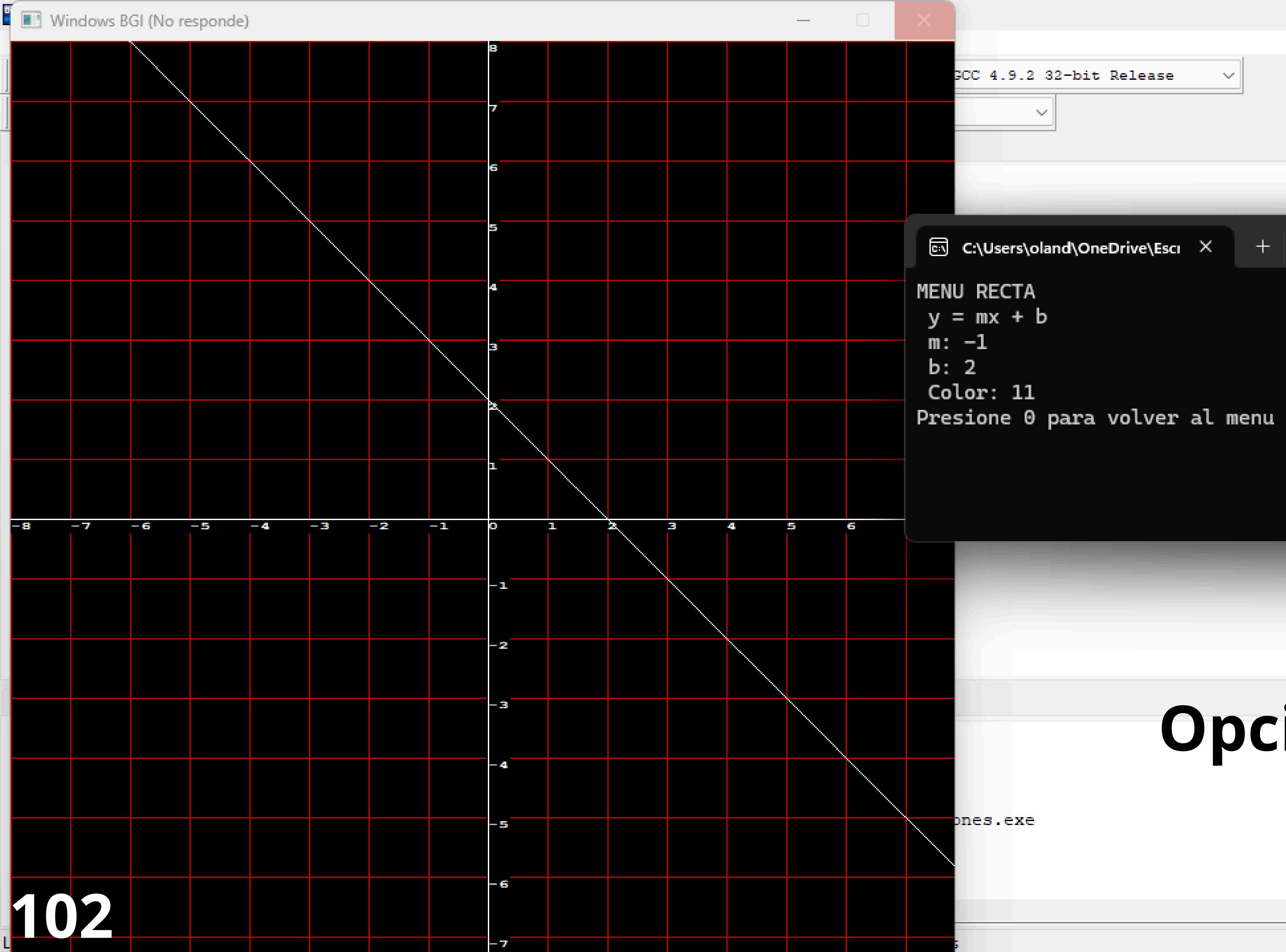
Opción [0]



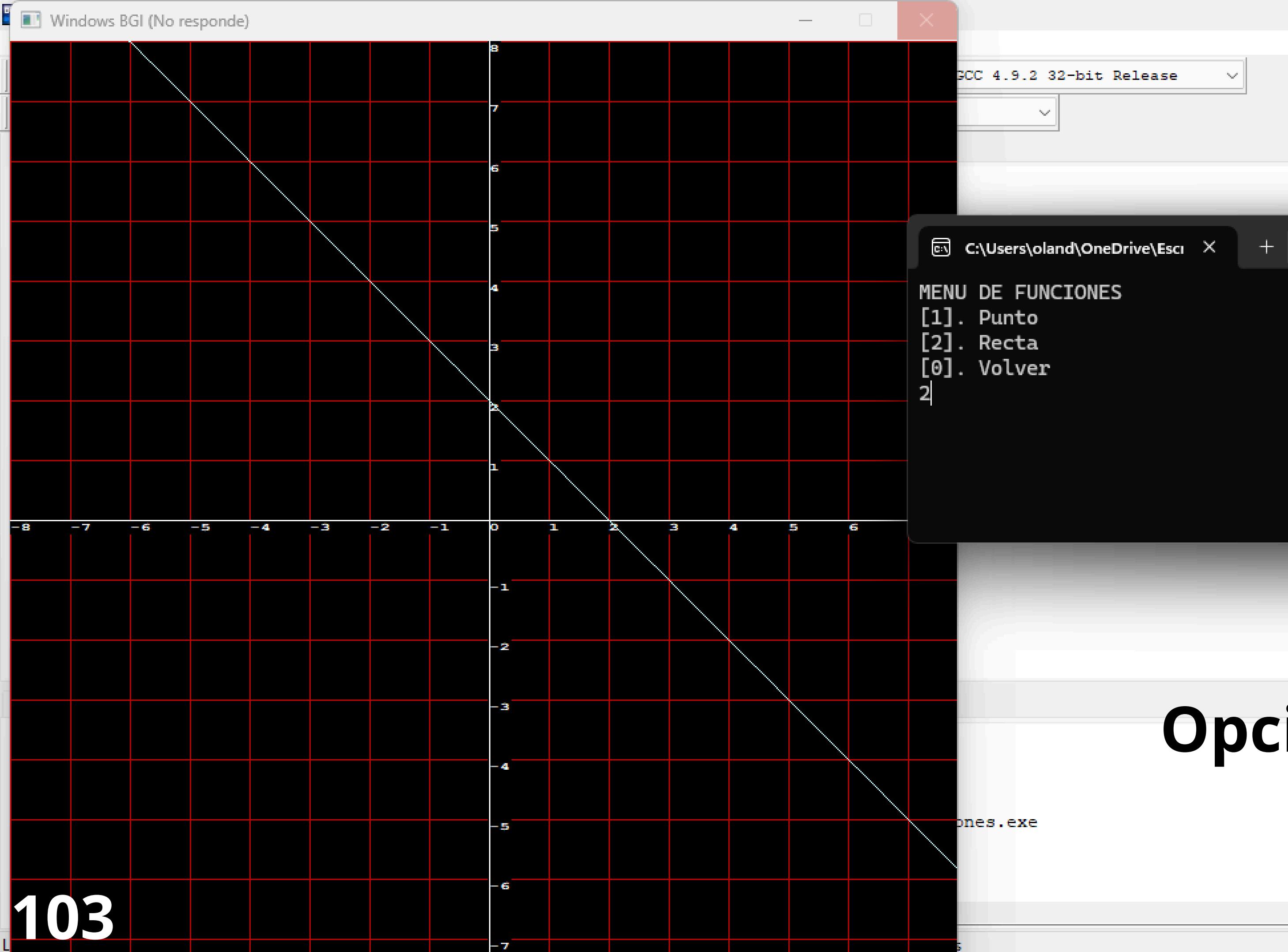
Opción [2]



Ingresamos valores
de la primera recta



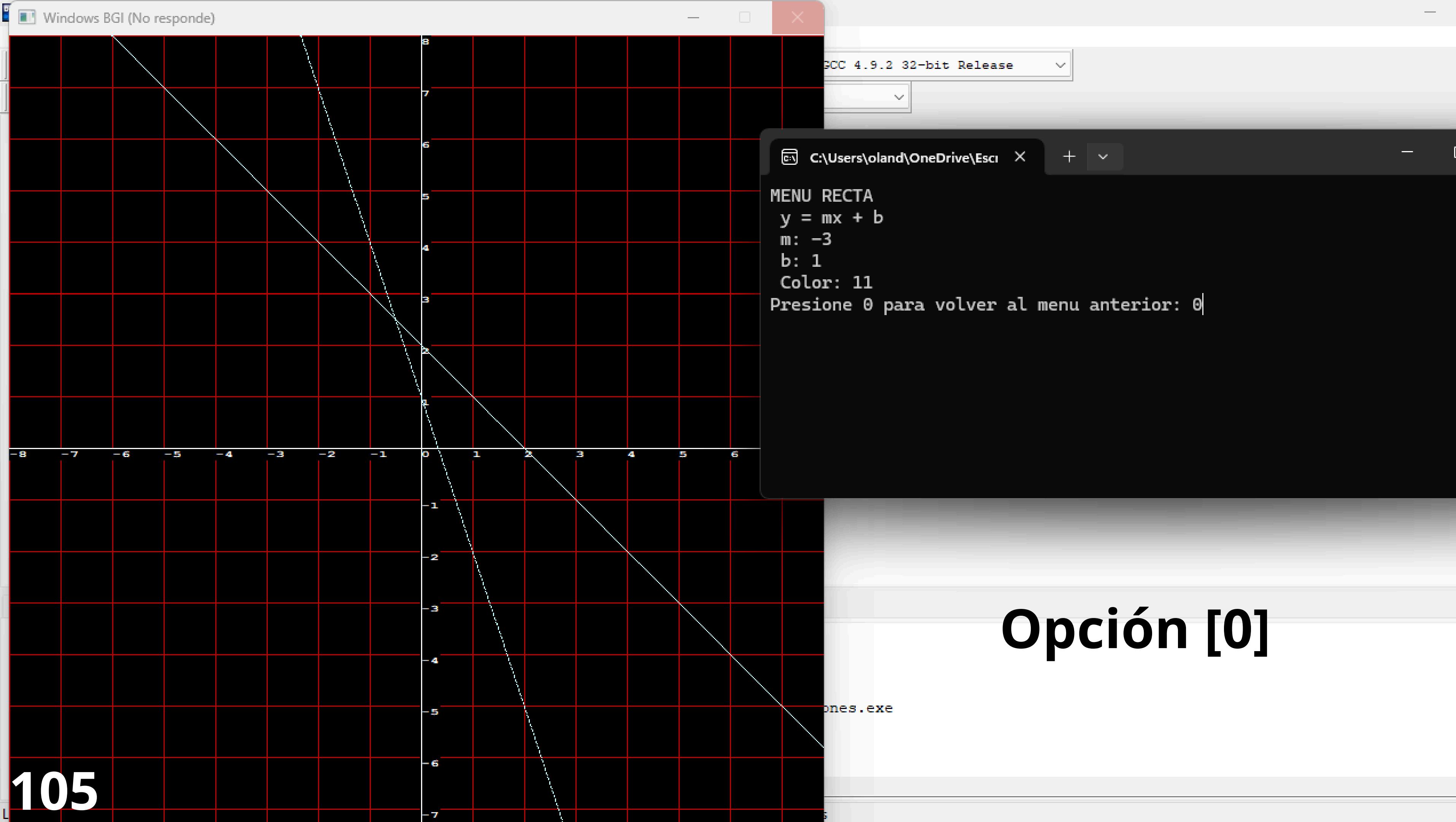
Opción [0]

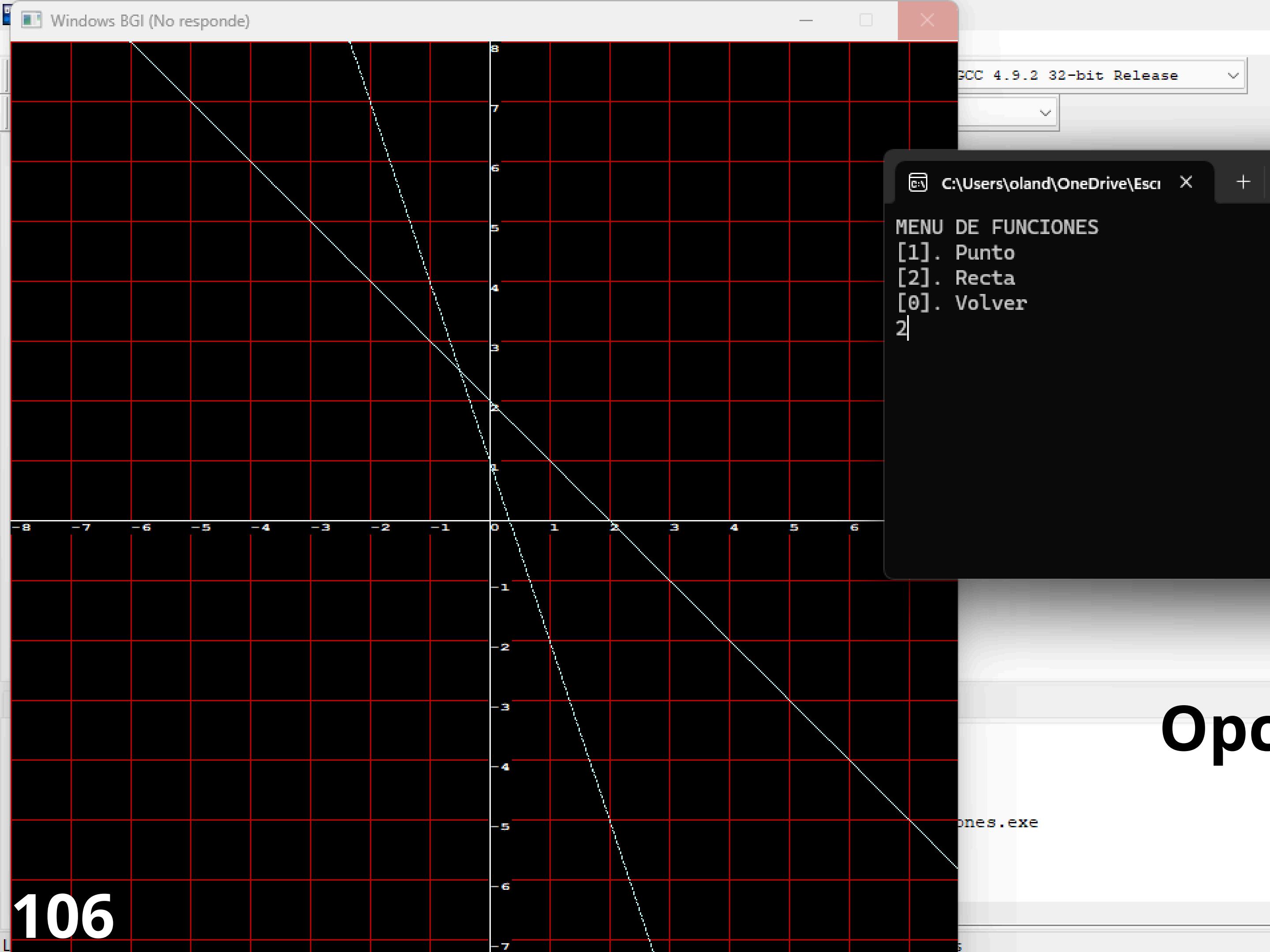


Opción [2]

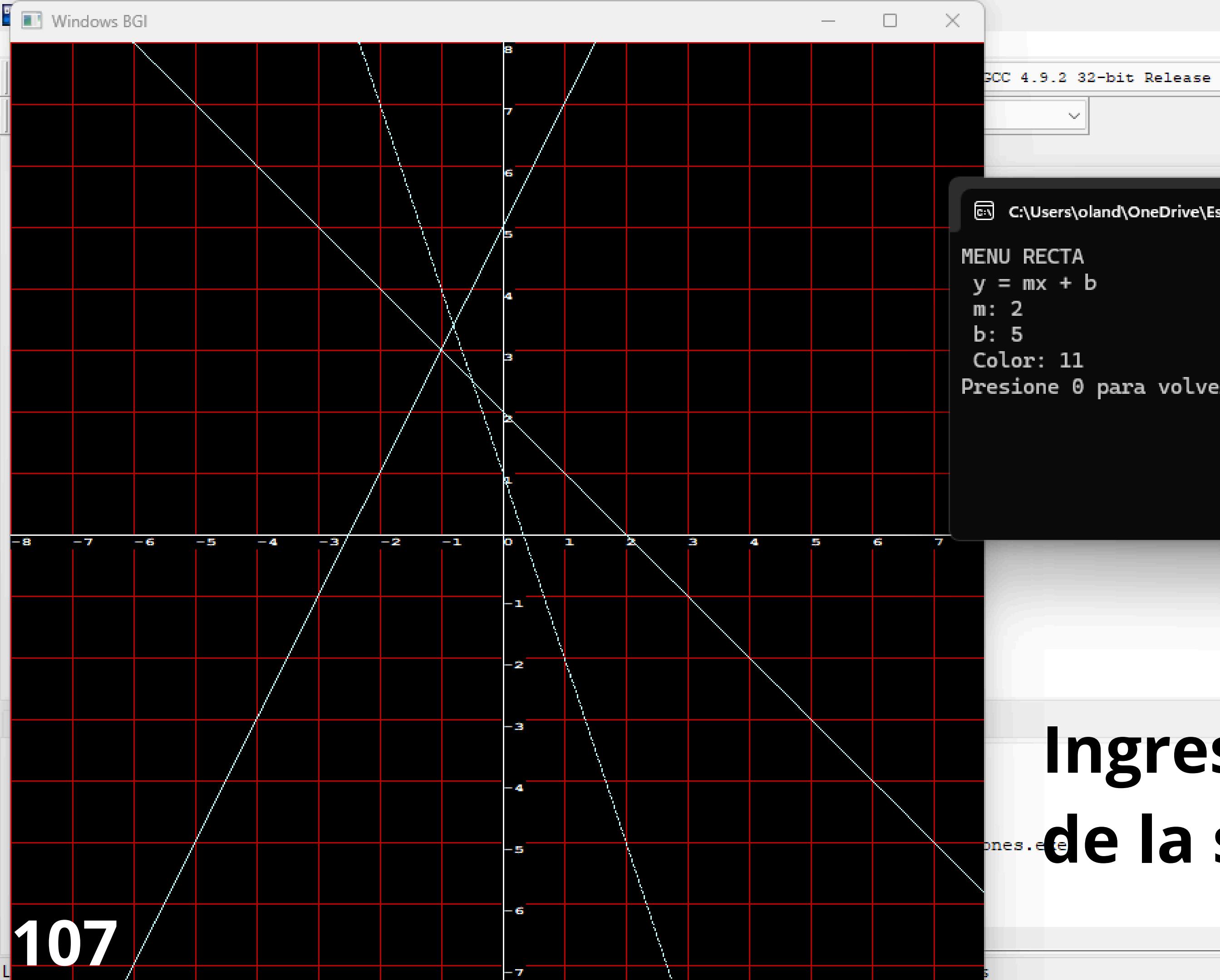


Ingresamos valores
de la segunda recta

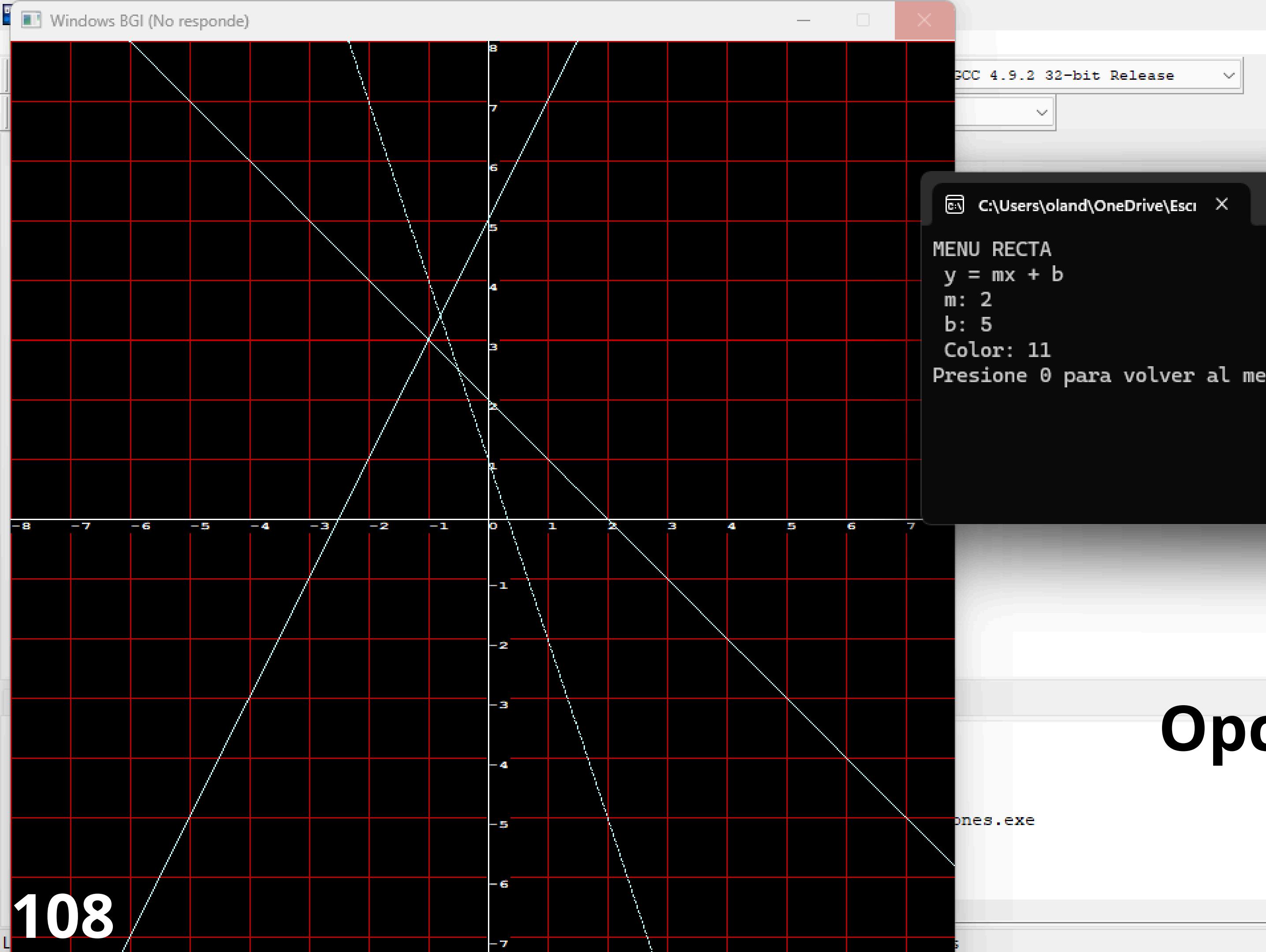




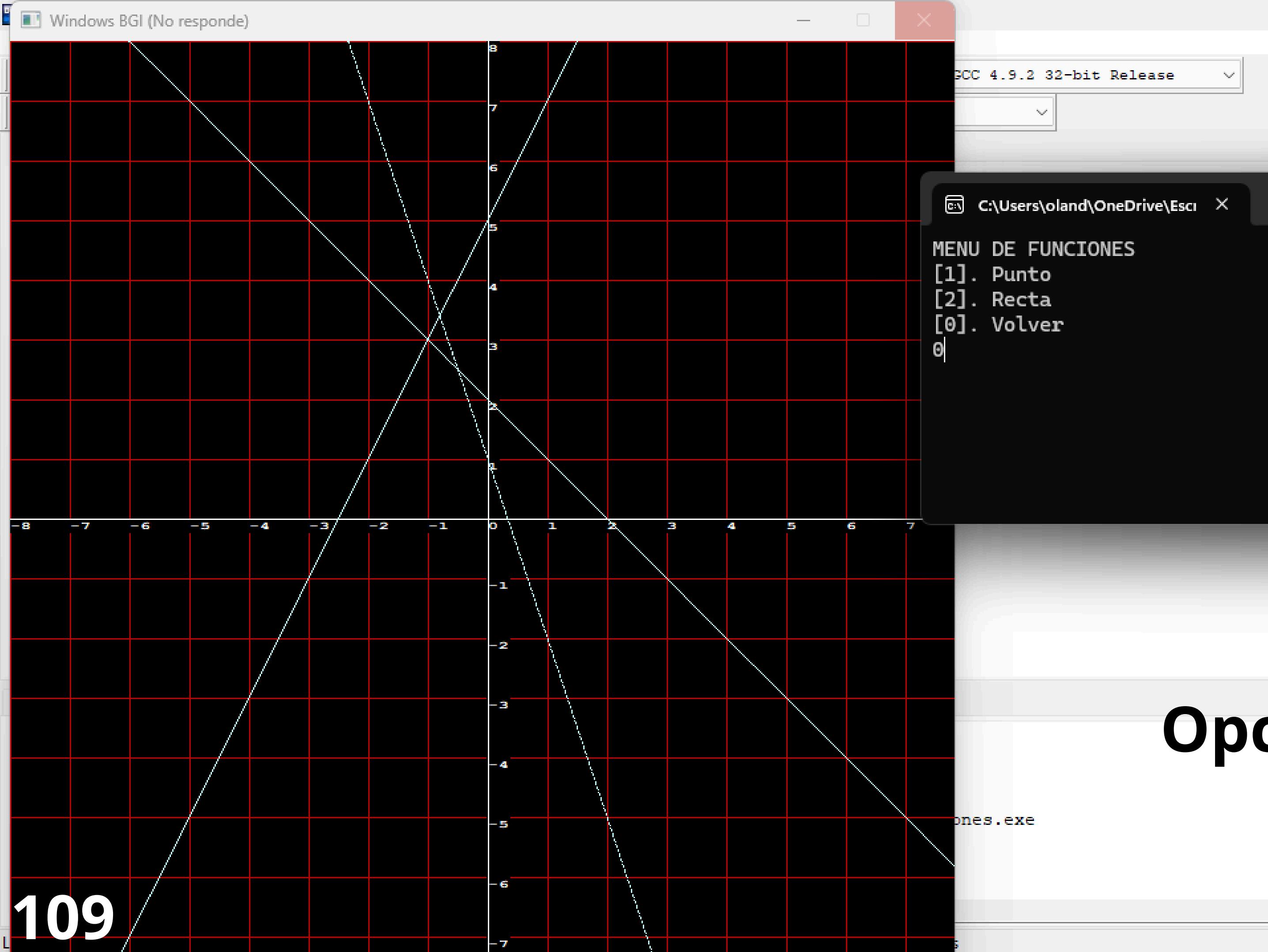
Opción [2]



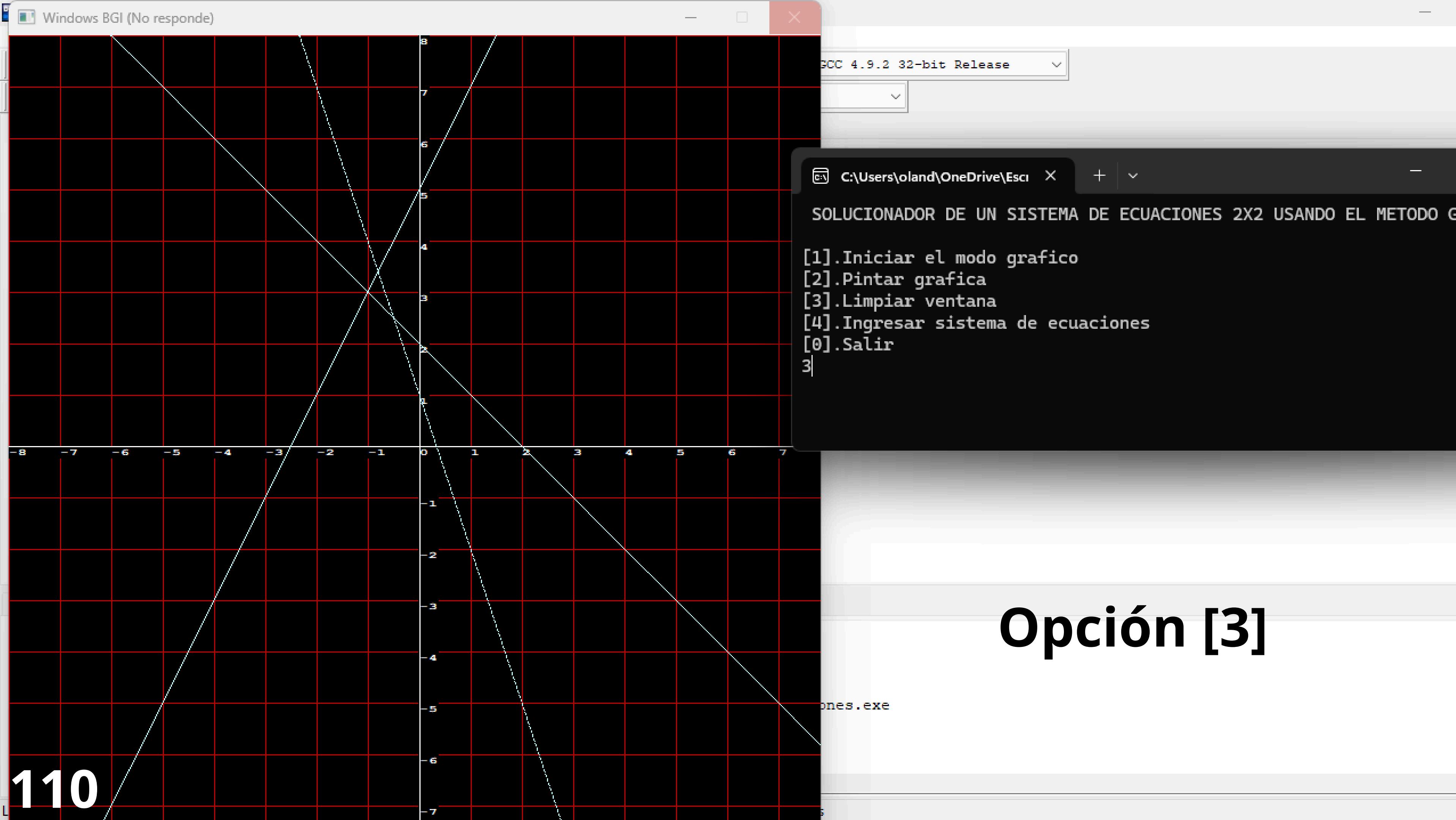
Ingresamos valores
de la segunda recta

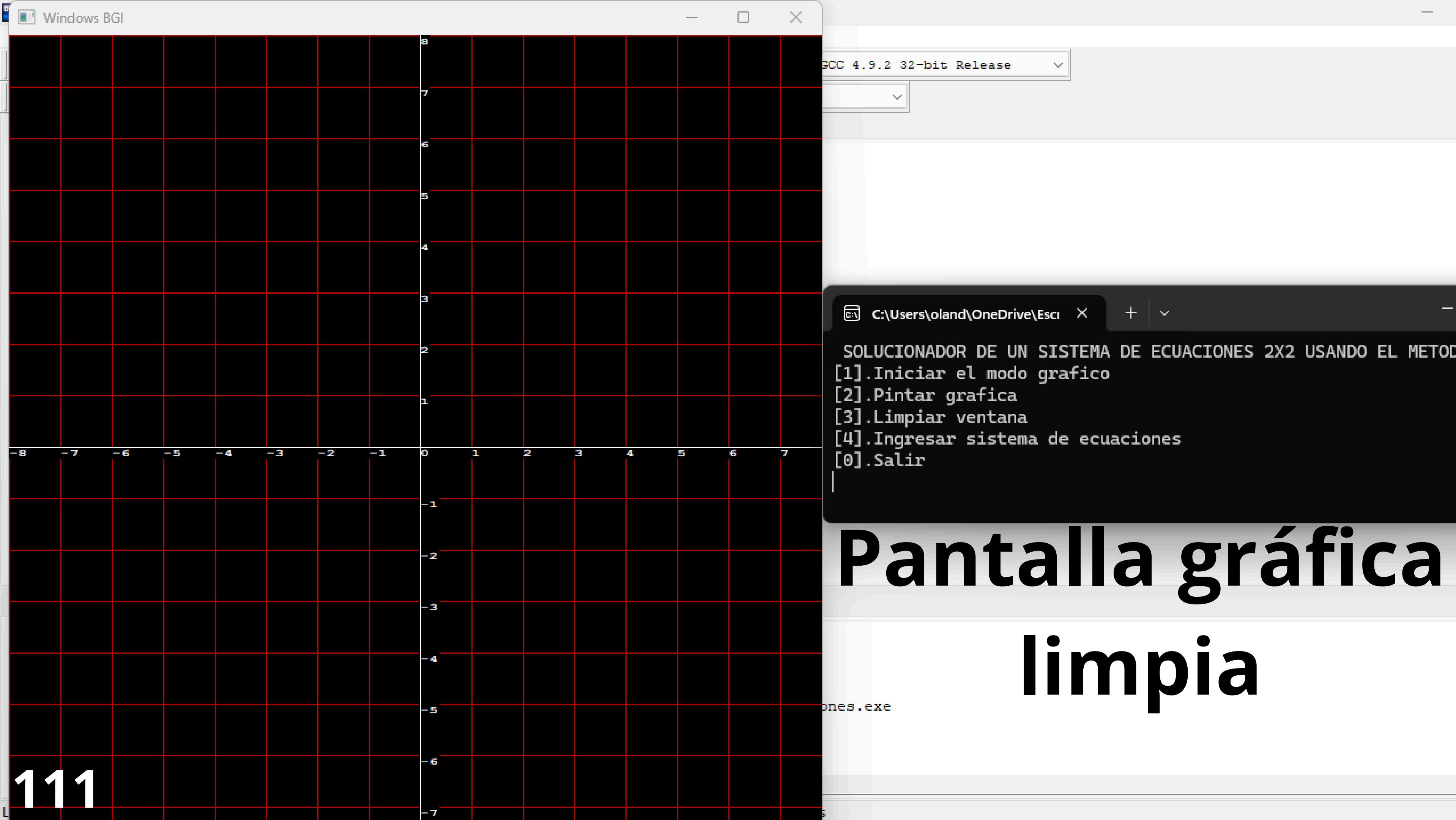


Opción [0]



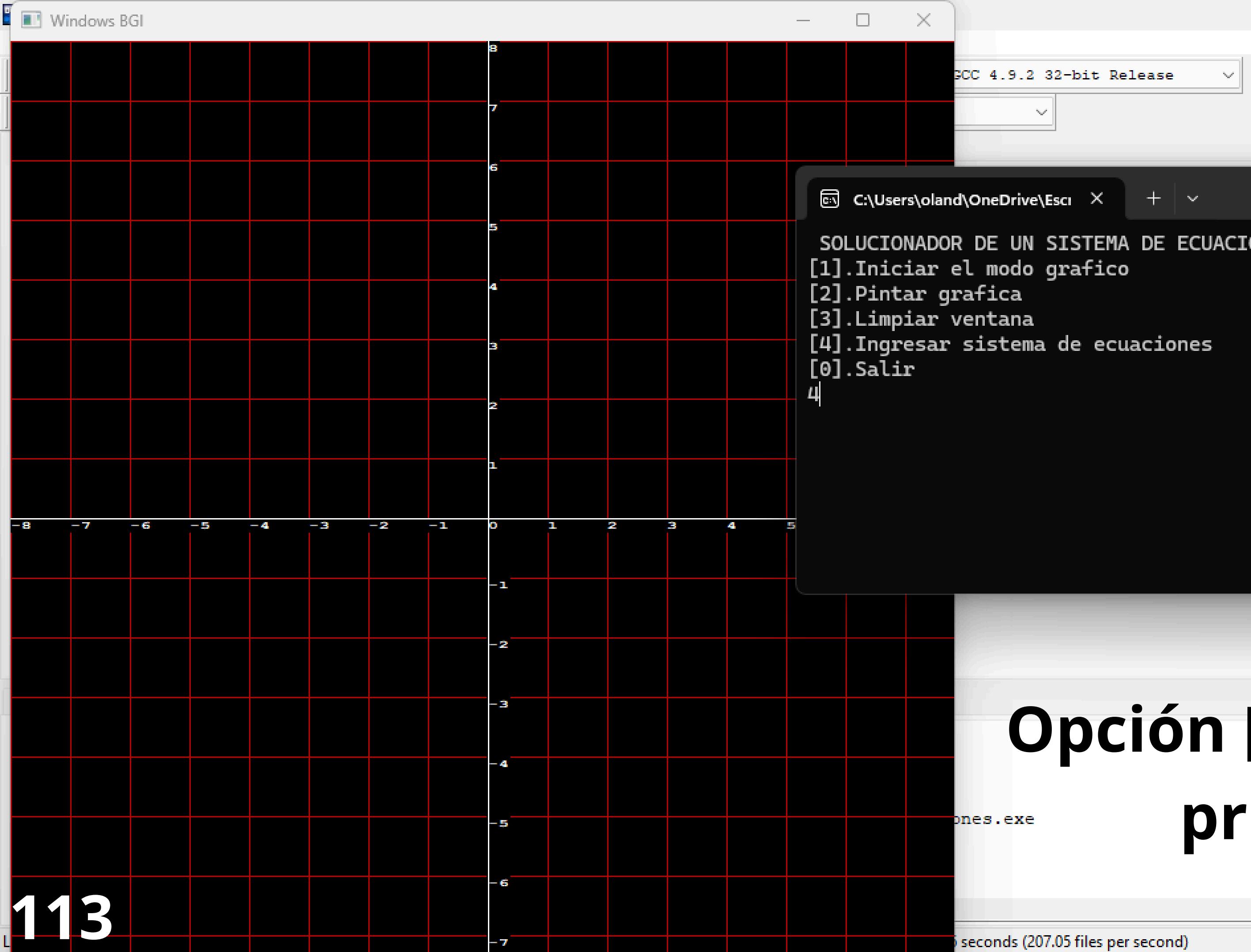
Opción [0]



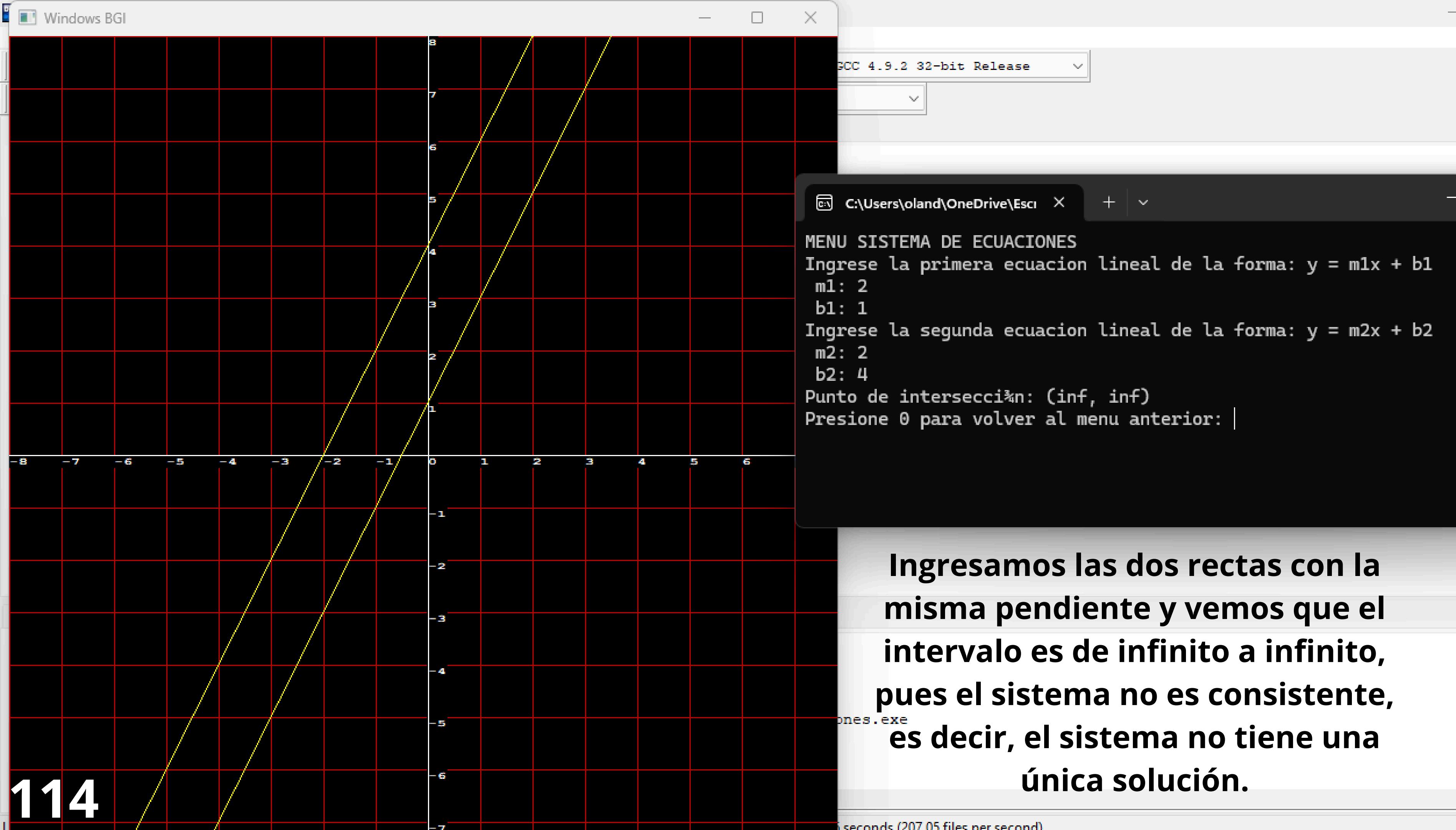


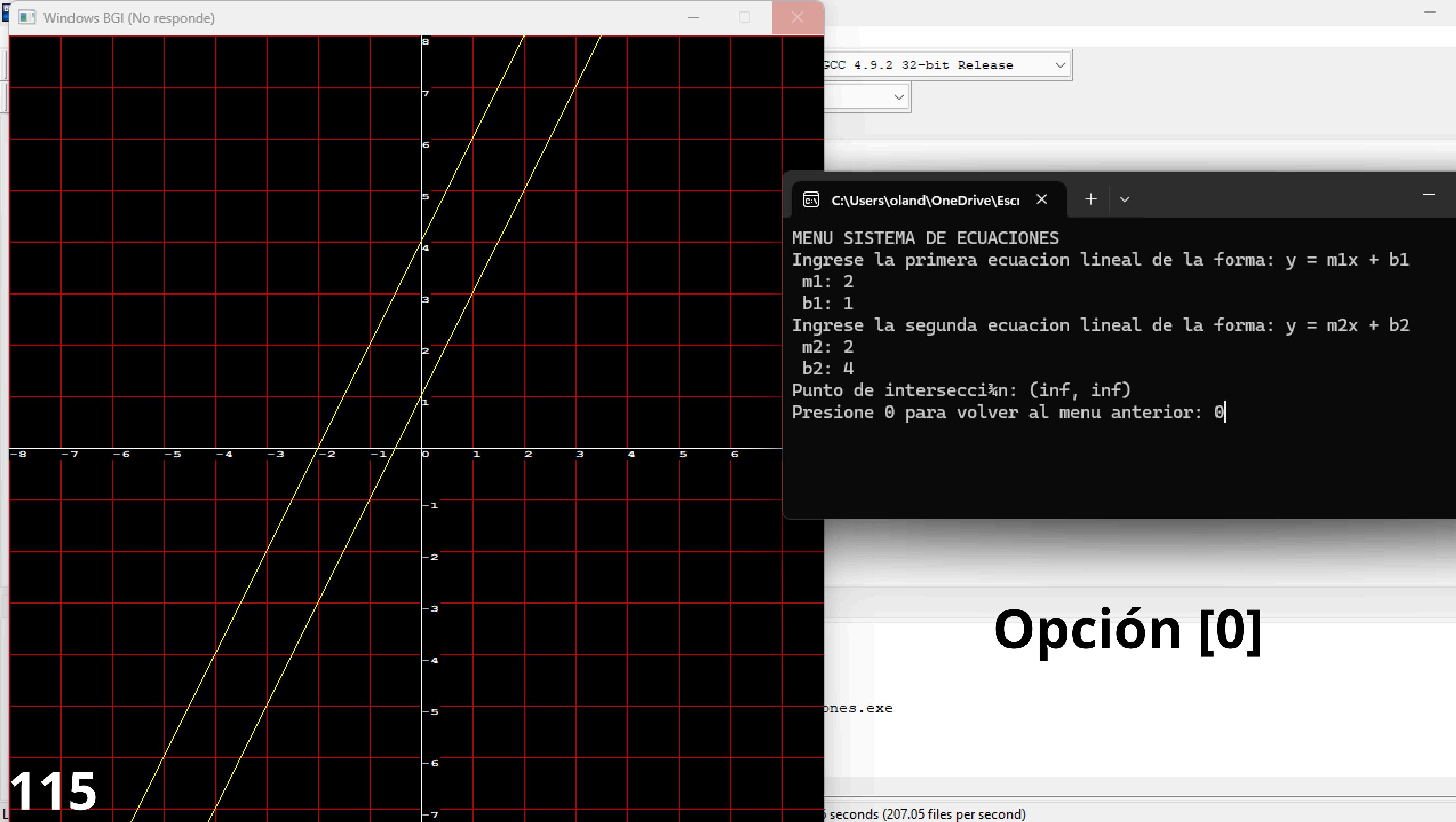
Caso de rectas paralelas
(misma pendiente) m_1

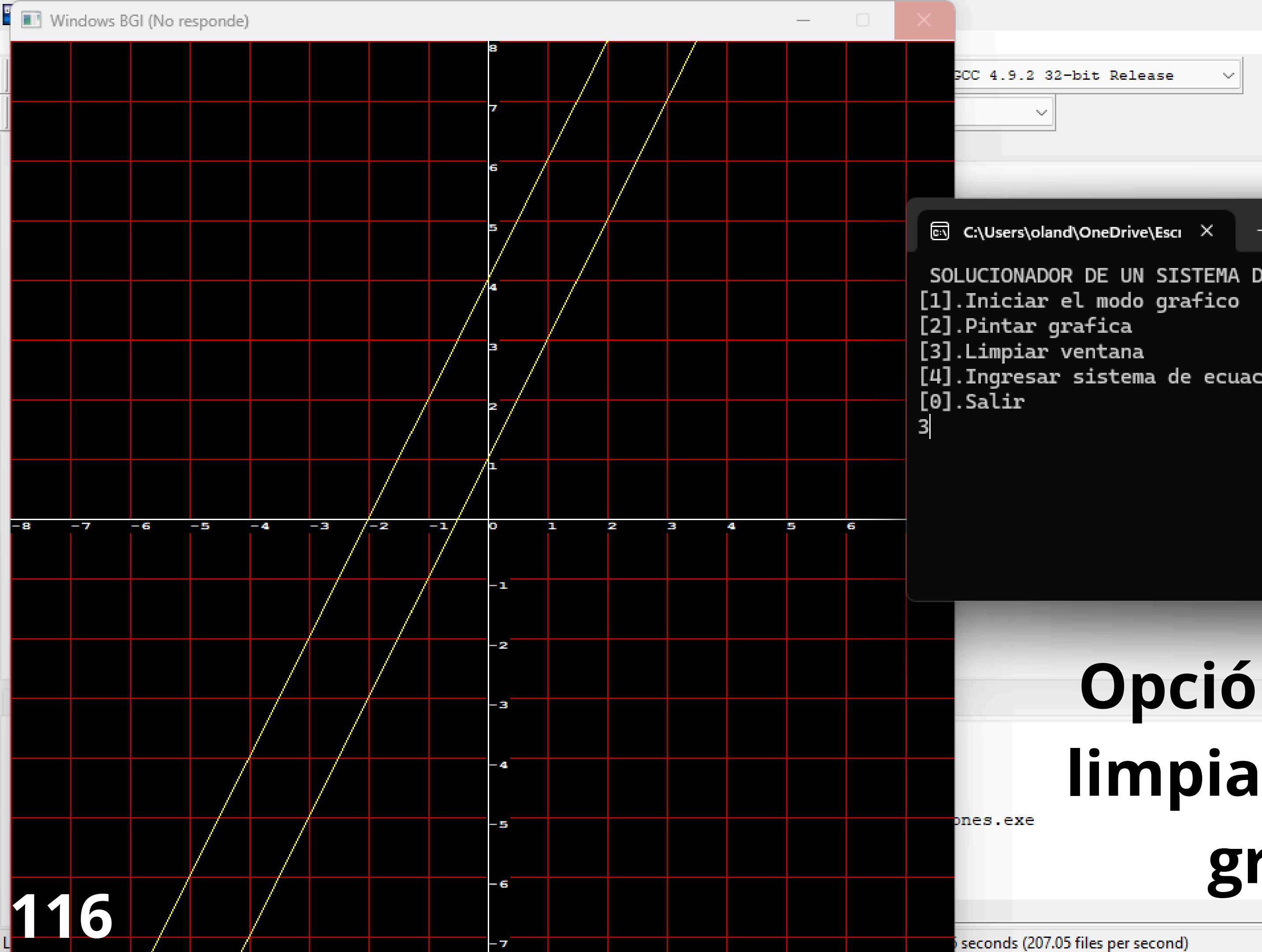
$$=m_2$$



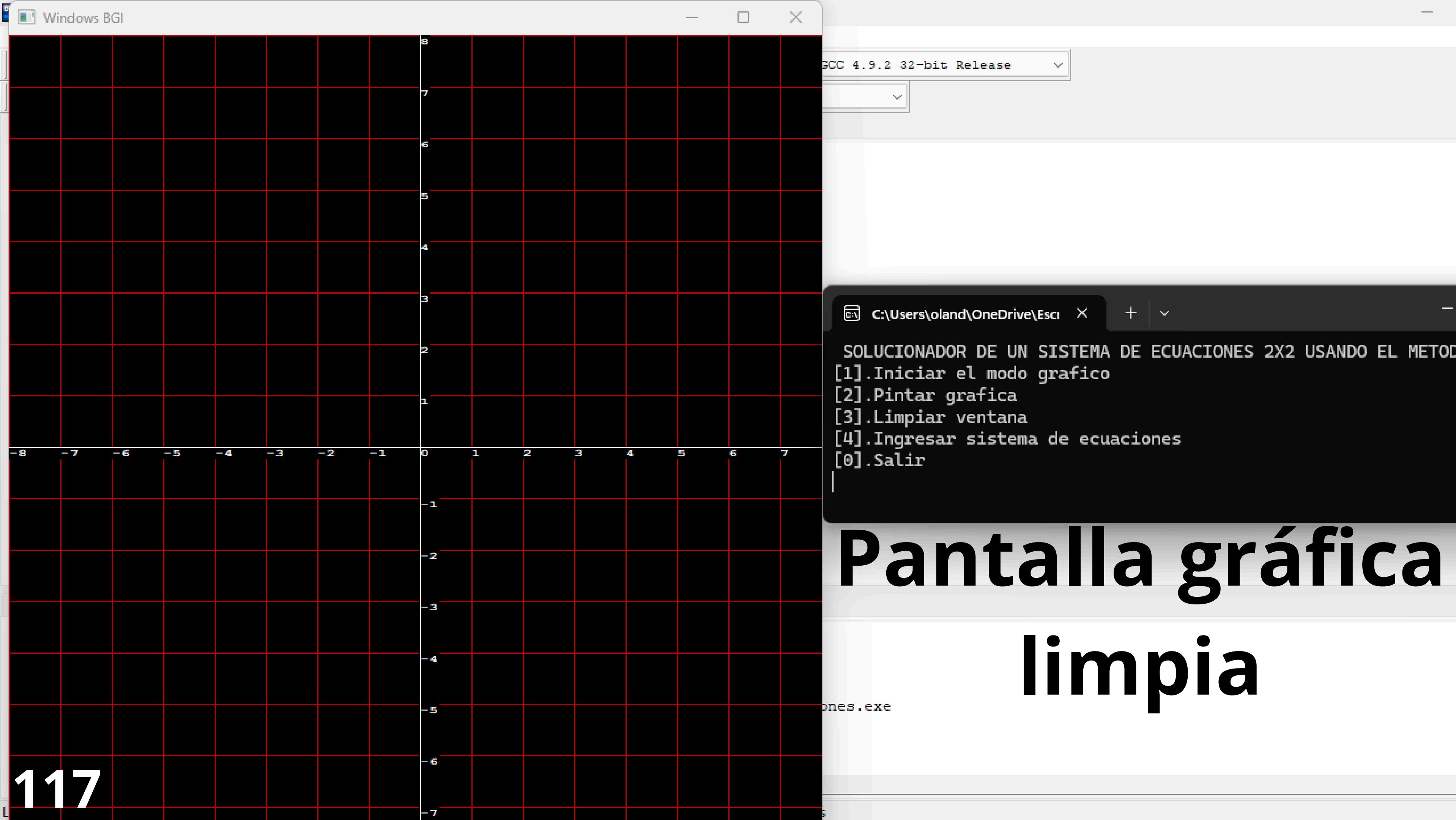
Opción [4] del menú principal

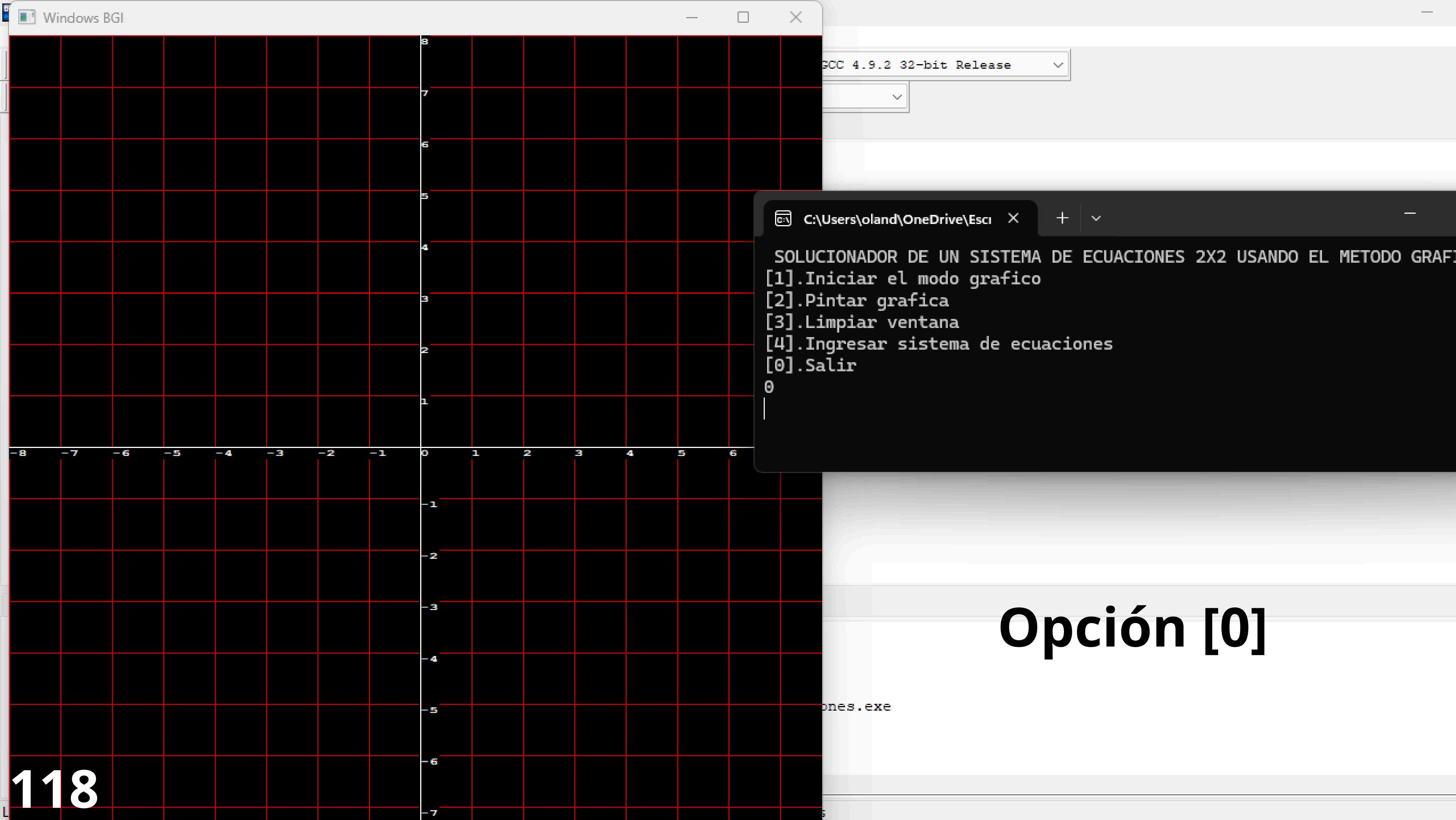






Opción [3] para
limpiar ventana
gráfica

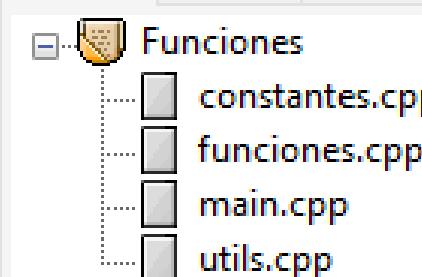




File Edit Search View Project Execute Tools AStyle Window Help



Project Classes Debug constantes.cpp main.cpp utils.cpp funciones.cpp



```
1 // Constantes
2 const int ancho = 720; // A
3 const int alto = 720; // B
4 const int k = 45;
5
6 // ATLITEC SARABIA DIEGO ALEJANDRO
```

```
C:\Users\oland\OneDrive\Escritorio\Funciones.exe
[1].Iniciar el modo grafico
[2].Pintar grafica
[3].Limpiar ventana
[4].Ingresar sistema de ecuaciones
[0].Salir
0

-----
Process exited after 43.3 seconds with return value 0
Presione una tecla para continuar . . . |
```

Compiler Resources Compile Log Debug Find Results Close

Abort Compilation

 Shorten compiler paths

Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\oland\OneDrive\Escritorio\Funciones\Funciones.exe
- Output Size: 1.36651515960693 MiB
- Compilation Time: 0.06s

Salimos del programa



Conclusiones

El proyecto desarrollado presenta una implementación eficaz de un sistema de graficación en C++ que permite visualizar y analizar la intersección de rectas en un plano cartesiano. A través del uso de bibliotecas gráficas y matemáticas, se logró crear una herramienta interactiva que facilita la comprensión y resolución de sistemas de ecuaciones lineales.

Durante el desarrollo del proyecto, se implementaron diversas funciones para manejar la entrada y salida de datos, así como para graficar puntos y líneas en un entorno gráfico. Esto incluyó la inicialización de una ventana gráfica, la representación de puntos y líneas, y la visualización de las intersecciones de las rectas ingresadas por el usuario.

El sistema no solo demuestra la correcta intersección de rectas, sino que también permite una interacción dinámica, donde el usuario puede ingresar diferentes valores y observar los resultados visualmente. Esto contribuye significativamente al aprendizaje y la comprensión de conceptos matemáticos fundamentales como la pendiente, la intersección y la solución de sistemas de ecuaciones lineales.

En resumen, el proyecto cumple con los objetivos propuestos, proporcionando una herramienta educativa valiosa que combina la teoría matemática con la práctica de programación en C++. Esta integración no solo refuerza los conocimientos adquiridos en ambas áreas, sino que también abre la puerta a futuras mejoras y ampliaciones del sistema, tales como la incorporación de más figuras geométricas y la posibilidad de resolver sistemas de ecuaciones más complejos.

MUCHAS GRACIAS

Finalmente, deseo expresar mi más sincero agradecimiento al Dr. Felipe Rolando Menchaca García por su esmero y dedicación en la enseñanza de la materia de Programación Orientada a Objetos. Su compromiso y pasión por la docencia han sido fundamentales para mi comprensión y apreciación de esta disciplina, lo que ha permitido la realización exitosa de este proyecto.