Карпенко О. КН20-1

**ЛАБОРАТОРНА РОБОТА 2**

Мета: Розробити програму для емуляції дисплейного модуля.

Етапи виконання лабораторної роботи:

1. Розробити архітектуру і реалізувати програму з графічним інтерфейсом здатну виконувати команди, наведені в лабораторній роботі №1. Обмеження на тип даних і параметри дивися в описі команд в лабораторній роботі №1.

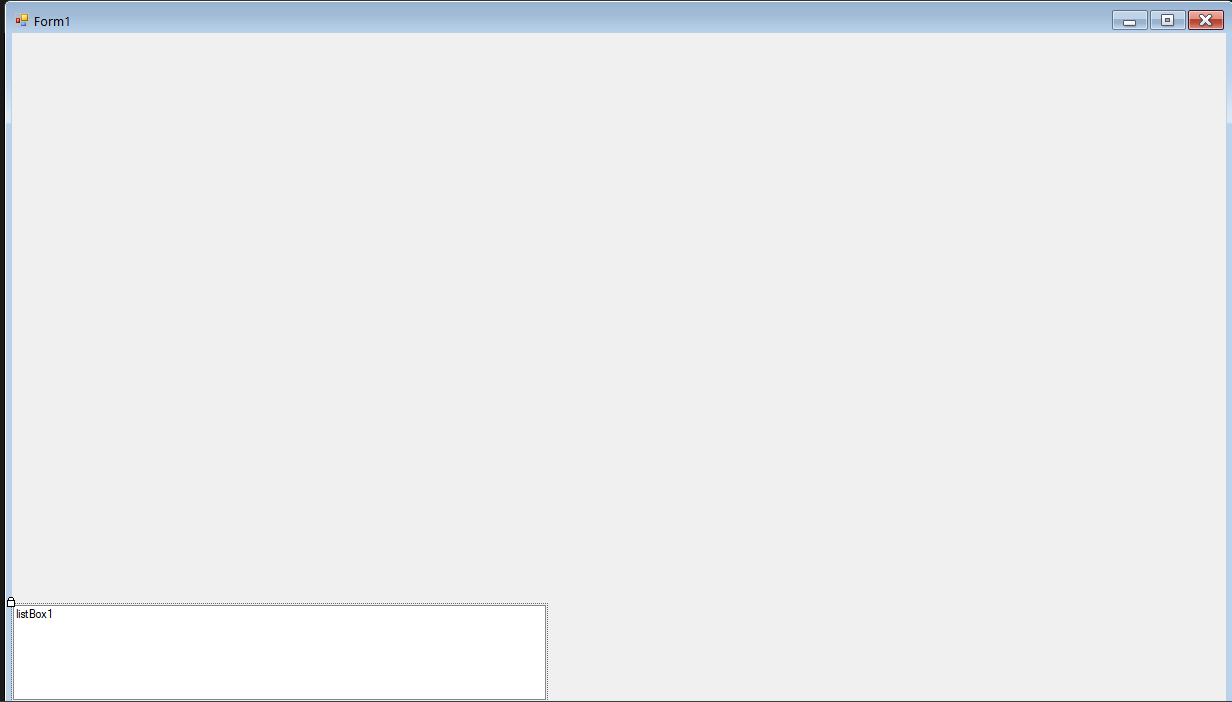
2. Додати в програму, розроблену в п.1 код для реалізації UDP сервера з лабораторної роботи №1. При спільному використанні як графічного інтерфейсу так і роботи з мережею можливо Вам знадобиться використання додаткових потоків виконання (threads).

3. Після проведення інтеграції(п.2) програма повинна мати можливість приймати команди, описані в лабораторній роботі №1, по протоколу UDP і відображати їх в графічному інтерфейсі.

Хід Роботи:

Сервер:

Дизайн:



Код:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net.Sockets;

using System.Net;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Threading;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

using System.Runtime.CompilerServices;

using System.Drawing.Drawing2D;

using static System.Net.Mime.MediaTypeNames;

using System.IO;

namespace ServerDrawer

{

public partial class Form1 : Form

{

public static int rotation;

public class Figure

{

public string Name, text;

public short x1, y1, x2, y2, rad, size;

public byte[] RGB;

public System.Drawing.Image Image;

public Figure(string \_name, short \_x1, short \_y1, byte[] \_RGB)

{

Name = \_name;

this.x1 = \_x1;

this.y1 = \_y1;

RGB = \_RGB;

}

public Figure(string \_name, short \_x1, short \_y1, short \_x2, short \_y2, byte[] \_RGB)

: this(\_name, \_x1, \_y1, \_RGB)

{

this.x2 = \_x2;

this.y2 = \_y2;

}

public Figure(string \_name, short \_x1, short \_y1, short \_x2, byte[] \_RGB)

: this(\_name, \_x1, \_y1, \_RGB)

{

this.x2 = \_x2;

}

public Figure(string \_name, short \_x1, short \_y1, short \_x2, short \_y2, short \_rad, byte[] \_RGB)

: this(\_name, \_x1, \_y1, \_RGB)

{

this.x2 = \_x2;

this.y2 = \_y2;

this.rad = \_rad;

}

public Figure(string \_name, short \_x1, short \_y1, short \_size, string \_text, byte[] \_RGB )

: this(\_name, \_x1, \_y1, \_RGB)

{

this.text = \_text;

this.size = \_size;

}

public Figure(string \_name, short \_x1, short \_y1, System.Drawing.Image \_image)

{

Name = \_name;

this.x1 = \_x1;

this.y1 = \_y1;

this.Image = \_image;

}

}

public static List<Figure> Figures = new List<Figure>();

public void DrawPixel(byte command, short x1, short y1, byte[] RGB)

{

var Figure = new Figure("Pixel", x1, y1, RGB);

Figures.Add(Figure);

Invalidate();

}

public void DrawLine(byte command, short x1, short y1, short x2, short y2, byte[] RGB)

{

var Figure = new Figure("Line", x1, y1, x2, y2, RGB);

Figures.Add(Figure);

Invalidate();

}

public void DrawRectangle (byte command, short x1, short y1, short x2, short y2, byte[] RGB)

{

var Figure = new Figure("RectangleOutline", x1, y1, x2, y2, RGB);

Figures.Add(Figure);

Invalidate();

}

public void FillRectangle(byte command, short x1, short y1, short x2, short y2, byte[] RGB)

{

var Figure = new Figure("RectangleFilled", x1, y1, x2, y2, RGB);

Figures.Add(Figure);

Invalidate();

}

public void DrawEllipse(byte command, short x1, short y1, short x2, short y2, byte[] RGB)

{

var Figure = new Figure("EllipseOutline", x1, y1, x2, y2, RGB);

Figures.Add(Figure);

Invalidate();

}

public void FillEllipse(byte command, short x1, short y1, short x2, short y2, byte[] RGB)

{

var Figure = new Figure("EllipseFilled", x1, y1, x2, y2, RGB);

Figures.Add(Figure);

Invalidate();

}

public void DrawCircle(byte command, short x1, short y1, short x2, byte[] RGB)

{

var Figure = new Figure("CircleOutline", x1, y1, x2, RGB);

Figures.Add(Figure);

Invalidate();

}

public void FillCircle(byte command, short x1, short y1, short x2, byte[] RGB)

{

var Figure = new Figure("CircleFilled", x1, y1, x2, RGB);

Figures.Add(Figure);

Invalidate();

}

public void DrawRoundedRectangle(byte command, short x1, short y1, short x2, short y2, short rad, byte[] RGB)

{

var Figure = new Figure("RoundedRectangleOutline", x1, y1, x2, y2, rad, RGB);

Figures.Add(Figure);

Invalidate();

}

public void FillRoundedRectangle(byte command, short x1, short y1, short x2, short y2, short rad, byte[] RGB)

{

var Figure = new Figure("RoundedRectangleFilled", x1, y1, x2, y2, rad, RGB);

Figures.Add(Figure);

Invalidate();

}

public void DrawText(byte command, short x1, short y1, short size, string text, byte[] RGB)

{

var Figure = new Figure("Text", x1, y1, size, text, RGB);

Figures.Add(Figure);

Invalidate();

}

public void DrawImage(byte command, short x1, short y1, System.Drawing.Image img)

{

var Figure = new Figure("Image", x1, y1, img);

Figures.Add(Figure);

Invalidate();

}

public static GraphicsPath RoundedRect(Rectangle bounds, int radius)

{

int diameter = radius \* 2;

Size size = new Size(diameter, diameter);

Rectangle arc = new Rectangle(bounds.Location, size);

GraphicsPath path = new GraphicsPath();

if (radius == 0)

{

path.AddRectangle(bounds);

return path;

}

path.AddArc(arc, 180, 90);

arc.X = bounds.Right - diameter;

path.AddArc(arc, 270, 90);

arc.Y = bounds.Bottom - diameter;

path.AddArc(arc, 0, 90);

arc.X = bounds.Left;

path.AddArc(arc, 90, 90);

path.CloseFigure();

return path;

}

const int port = 1984;

public void ClearDisplay(byte[] RecievedData, out byte[] RGB)

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

Array.Copy(RecievedData, val1place, RGB, 0, RGB.Length);

Figures.Clear();

this.BackColor = Color.FromArgb(RGB[0], RGB[1], RGB[2]);

Invalidate();

}

public void RotateImage(byte[] RecievedData, out Int16 Orientation)

{

Orientation = RecievedData[1];

rotation = BitConverter.ToInt16(RecievedData, 1);

Invalidate();

}

private void Form1\_Paint(object sender, PaintEventArgs e)

{

Graphics g = e.Graphics;

g.TranslateTransform(this.Width/2, this.Height/2);

g.RotateTransform(rotation);

g.TranslateTransform(-this.Width / 2, -this.Height / 2);

g.InterpolationMode = InterpolationMode.HighQualityBicubic;

if (Figures != null)

{

foreach(var Figure in Figures)

{

if (Figure.Name == "Pixel")

{

Brush brush = new SolidBrush(Color.FromArgb(Figure.RGB[0], Figure.RGB[1], Figure.RGB[2]));

g.FillRectangle(brush, Figure.x1+this.Width/2, Figure.y1+this.Height/2, 1, 1);

}

else if (Figure.Name == "Line")

{

Pen Pen = new Pen(Color.FromArgb(Figure.RGB[0], Figure.RGB[1], Figure.RGB[2]),5);

g.DrawLine(Pen, Figure.x1 + this.Width / 2 - Figure.x1 / 2, Figure.y1 + this.Height / 2 - Figure.y1 / 2, Figure.x2 + this.Width / 2 - Figure.x2 / 2, Figure.y2 + this.Height / 2 - Figure.y2/2);

}

else if (Figure.Name == "RectangleOutline")

{

Pen Pen = new Pen(Color.FromArgb(Figure.RGB[0], Figure.RGB[1], Figure.RGB[2]), 1);

g.DrawRectangle(Pen, Figure.x1 + this.Width / 2 - Figure.x2 / 2, Figure.y1 + this.Height / 2 - Figure.y2 / 2, Figure.x2, Figure.y2);

}

else if (Figure.Name == "RectangleFilled")

{

Brush brush = new SolidBrush(Color.FromArgb(Figure.RGB[0], Figure.RGB[1], Figure.RGB[2]));

g.FillRectangle(brush, Figure.x1 + this.Width / 2 - Figure.x2 / 2, Figure.y1 + this.Height / 2 - Figure.y2 / 2, Figure.x2, Figure.y2);

}

else if (Figure.Name == "EllipseOutline")

{

Pen Pen = new Pen(Color.FromArgb(Figure.RGB[0], Figure.RGB[1], Figure.RGB[2]), 1);

g.DrawEllipse(Pen, Figure.x1 + this.Width / 2 - Figure.x2 / 2, Figure.y1 + this.Height / 2 - Figure.y2 / 2, Figure.x2, Figure.y2);

}

else if (Figure.Name == "EllipseFilled")

{

Brush brush = new SolidBrush(Color.FromArgb(Figure.RGB[0], Figure.RGB[1], Figure.RGB[2]));

g.FillEllipse(brush, Figure.x1 + this.Width / 2 - Figure.x2 / 2, Figure.y1 + this.Height / 2 - Figure.y2 / 2, Figure.x2, Figure.y2);

}

else if (Figure.Name == "CircleOutline")

{

Pen Pen = new Pen(Color.FromArgb(Figure.RGB[0], Figure.RGB[1], Figure.RGB[2]), 1);

g.DrawEllipse(Pen, Figure.x1 + this.Width / 2 - Figure.x2 / 2, Figure.y1 + this.Height / 2 - Figure.x2 / 2, Figure.x2, Figure.x2);

}

else if (Figure.Name == "CircleFilled")

{

Brush brush = new SolidBrush(Color.FromArgb(Figure.RGB[0], Figure.RGB[1], Figure.RGB[2]));

g.FillEllipse(brush, Figure.x1 + this.Width / 2 - Figure.x2 / 2, Figure.y1 + this.Height / 2 - Figure.x2 / 2, Figure.x2, Figure.x2);

}

else if (Figure.Name == "RoundedRectangleOutline")

{

Pen Pen = new Pen(Color.FromArgb(Figure.RGB[0], Figure.RGB[1], Figure.RGB[2]), 1);

Rectangle rectangle = new Rectangle(Figure.x1 + this.Width / 2 - Figure.x2 / 2, Figure.y1 + this.Height / 2 - Figure.y2 / 2, Figure.x2, Figure.y2);

g.DrawPath(Pen, RoundedRect(rectangle,Figure.rad));

}

else if (Figure.Name == "RoundedRectangleFilled")

{

Brush brush = new SolidBrush(Color.FromArgb(Figure.RGB[0], Figure.RGB[1], Figure.RGB[2]));

Rectangle rectangle = new Rectangle(Figure.x1 + this.Width / 2 - Figure.x2 / 2, Figure.y1 + this.Height / 2 - Figure.x2 / 2, Figure.x2, Figure.y2);

g.FillPath(brush, RoundedRect(rectangle, Figure.rad));

}

else if (Figure.Name == "Text")

{

Brush brush = new SolidBrush(Color.FromArgb(Figure.RGB[0], Figure.RGB[1], Figure.RGB[2]));

Font font = new Font("Arial", Figure.size);

g.DrawString(Figure.text, font, brush, Figure.x1 + this.Width / 2, Figure.y1 + this.Height / 2);

}

else if(Figure.Name == "Image")

{

g.DrawImage(Figure.Image, Figure.x1 + this.Width / 2, Figure.y1 + this.Height / 2);

}

}

}

}

public static void ThreeVarsDecode(byte[] RecievedData, out Int16 val1, out Int16 val2, out byte[] RGB)

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, RGB, 0, RGB.Length);

}

public static void FiveVarsDecode(byte[] RecievedData, out Int16 val1, out Int16 val2, out Int16 val3, out Int16 val4, out byte[] RGB)

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val3 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val4 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, RGB, 0, RGB.Length);

}

public static void CircleDecoder(byte[] RecievedData, out Int16 val1, out Int16 val2, out Int16 val3, out byte[] RGB)

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val3 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, RGB, 0, RGB.Length);

}

public static void RoundedRectangleDecoder(byte[] RecievedData, out Int16 val1, out Int16 val2, out Int16 val3, out Int16 val4, out Int16 val5, out byte[] RGB)

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val3 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val4 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val5 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, RGB, 0, RGB.Length);

}

public static void TextDecoder(byte[] RecievedData, out Int16 val1, out Int16 val2, out Int16 val3, out Int16 val4, out byte[] RGB, out string text)

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, RGB, 0, RGB.Length);

val1place += 3;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val3 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val4 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

transfer = new byte[val4];

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

text = Encoding.ASCII.GetString(transfer);

}

public static void ImageDecoder(byte[] RecievedData, out Int16 val1, out Int16 val2, out Int16 width, out Int16 height, out System.Drawing.Image pic)

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

Bitmap bitmap;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

width = BitConverter.ToInt16(transfer, 0);

val1place += 2;

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

height = BitConverter.ToInt16(transfer, 0);

val1place += 2;

bitmap = new Bitmap(width, height);

for (int i = 0; i < width; i++)

{

for (int j = 0; j < height; j++)

{

transfer = new byte[3];

Array.Copy(RecievedData, val1place, transfer, 0, transfer.Length);

bitmap.SetPixel(i,j, Color.FromArgb(transfer[0], transfer[1], transfer[2]));

val1place += 3;

}

}

pic = bitmap;

}

private void StartServer()

{

Int16 val1, val2, val3, val4, val5; byte[] RGB, sendMessage; string text; byte command; System.Drawing.Image img;

Console.WriteLine("Server Begin");

UdpClient server = new UdpClient(port);

IPEndPoint localEP = new IPEndPoint(IPAddress.Any, 0);

IPEndPoint remoteEP;

try

{

while (true)

{

Console.WriteLine("Waiting for message");

byte[] RecievedData = server.Receive(ref localEP);

command = RecievedData[0];

Console.WriteLine($"Received broadcast from {localEP} :");

switch (command)

{

case 1:

ClearDisplay(RecievedData, out RGB);

text = $"command:Clear Display; color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]};";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text ); });

sendMessage = Encoding.ASCII.GetBytes(text);

remoteEP = new IPEndPoint(localEP.Address, localEP.Port);

server.Send(sendMessage, sendMessage.Length, remoteEP);

break;

case 2:

ThreeVarsDecode(RecievedData, out val1, out val2, out RGB);

DrawPixel(command, val1, val2, RGB);

text = $"command:Draw Pixel; Coordinates: x = {val1}, y = {val2}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]} ";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

remoteEP = new IPEndPoint(localEP.Address, localEP.Port);

server.Send(sendMessage, sendMessage.Length, remoteEP);

break;

case 3:

FiveVarsDecode(RecievedData, out val1, out val2, out val3, out val4, out RGB);

DrawLine(command,val1,val2,val3,val4,RGB);

text = $"command:Draw Line; Coordinates: x1 = {val1}, y1 = {val2}, x2 = {val3}, y2 = {val4}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

remoteEP = new IPEndPoint(localEP.Address, localEP.Port);

server.Send(sendMessage, sendMessage.Length, remoteEP);

break;

case 4:

FiveVarsDecode(RecievedData, out val1, out val2, out val3, out val4, out RGB);

DrawRectangle(command, val1, val2, val3, val4, RGB);

text = $"command:Draw Rectangle; Coordinates: x1 = {val1}, y1 = {val2}, width = {val3}, height = {val4}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

remoteEP = new IPEndPoint(localEP.Address, localEP.Port);

server.Send(sendMessage, sendMessage.Length, remoteEP);

break;

case 5:

FiveVarsDecode(RecievedData, out val1, out val2, out val3, out val4, out RGB);

FillRectangle(command, val1, val2, val3, val4, RGB);

text = $"command:Fill Rectangle; Coordinates: x1 = {val1}, y1 = {val2}, width = {val3}, height = {val4}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

server.Send(sendMessage, sendMessage.Length, localEP);

break;

case 6:

FiveVarsDecode(RecievedData, out val1, out val2, out val3, out val4, out RGB);

DrawEllipse(command, val1, val2, val3, val4, RGB);

text = $"command:Draw Ellipse; Coordinates: x1 = {val1}, y1 = {val2}, radius x = {val3}, radius y = {val4}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

server.Send(sendMessage, sendMessage.Length, localEP);

break;

case 7:

FiveVarsDecode(RecievedData, out val1, out val2, out val3, out val4, out RGB);

FillEllipse(command, val1, val2, val3, val4, RGB);

text = $"command:Fill Ellipse; Coordinates: x1 = {val1}, y1 = {val2}, radius x = {val3}, radius y = {val4}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

server.Send(sendMessage, sendMessage.Length, localEP);

break;

case 8:

CircleDecoder(RecievedData, out val1, out val2, out val3, out RGB);

DrawCircle(command, val1, val2, val3, RGB);

text = $"command:Draw Circle; Coordinates: x1 = {val1}, y1 = {val2}, radius = {val3}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

server.Send(sendMessage, sendMessage.Length, localEP);

break;

case 9:

CircleDecoder(RecievedData, out val1, out val2, out val3, out RGB);

FillCircle(command, val1, val2, val3, RGB);

text = $"command:Fill Circle; Coordinates: x1 = {val1}, y1 = {val2}, radius = {val3}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

server.Send(sendMessage, sendMessage.Length, localEP);

break;

case 10:

RoundedRectangleDecoder(RecievedData, out val1, out val2, out val3, out val4, out val5, out RGB);

DrawRoundedRectangle(command, val1, val2, val3, val4, val5, RGB);

text = $"command:Draw Rounded Rectangle; Rounded Rectangle Drawn: Coordinates: x1 = {val1}, y1 = {val2}, width = {val3}, height = {val4}, radius = {val5}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

server.Send(sendMessage, sendMessage.Length, localEP);

break;

case 11:

RoundedRectangleDecoder(RecievedData, out val1, out val2, out val3, out val4, out val5, out RGB);

FillRoundedRectangle(command, val1, val2, val3, val4, val5, RGB);

text = $"command:Fill Rounded Rectangle. Rounded Rectangle Filled: Coordinates: x1 = {val1}, y1 = {val2}, width = {val3}, height = {val4}, radius = {val5}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

server.Send(sendMessage, sendMessage.Length, localEP);

break;

case 12:

TextDecoder(RecievedData, out val1, out val2, out val3, out val4, out RGB, out text);

DrawText(command,val1,val2,val3,text,RGB);

text = $"command:Draw Text. Coordinates: x1 = {val1}, y1 = {val2}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}, font size = {val3}, text = \b {text} ";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

server.Send(sendMessage, sendMessage.Length, localEP);

break;

case 13:

ImageDecoder(RecievedData, out val1, out val2, out val3, out val4, out img);

DrawImage(command, val1, val2, img);

text = $"command:Draw Image; Coordinates: x1 = {val1}, y1 = {val2}, image width = {val3}, image height = {val4}";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

server.Send(sendMessage, sendMessage.Length, localEP);

break;

case 14:

RotateImage(RecievedData, out val1);

text = $"Command: Set Orientation; {val1} degrees";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

server.Send(sendMessage, sendMessage.Length, localEP);

break;

case 15:

val1 = Convert.ToInt16(this.Width);

text = $"Command: Get Width; Width: {val1} px";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

server.Send(sendMessage, sendMessage.Length, localEP);

break;

case 16:

val1 = Convert.ToInt16(this.Height);

Console.WriteLine($"Height: {val1} px");

text = $"Command: Get Height; Height: {val1} px";

Invoke((MethodInvoker)delegate { listBox1.Items.Add(Text = text); });

sendMessage = Encoding.ASCII.GetBytes(text);

server.Send(sendMessage, sendMessage.Length, localEP);

break;

}

}

}

catch (SocketException e)

{

Console.WriteLine(e);

}

finally

{

server.Close();

}

}

public Form1()

{

int middleHeight = this.ClientSize.Height/2;

int middleWidth = this.ClientSize.Width/2;

InitializeComponent();

try

{

Thread receiveThread = new Thread(new ThreadStart(StartServer));

receiveThread.Start();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

private void Form1\_Resize(object sender, EventArgs e)

{

Invalidate();

}

}

}

Клієнт:

Код:

using System.Net.Sockets;

using System.Net;

using System.Text;

using System.Globalization;

using System.Runtime.ConstrainedExecution;

using System.Drawing;

using static System.Net.Mime.MediaTypeNames;

using System.Collections.Generic;

using System.IO;

using System;

using System.Drawing.Imaging;

namespace Client

{

public static class ImageExtensions

{

public static byte[] ToByteArray(this System.Drawing.Image image, ImageFormat format)

{

using (MemoryStream ms = new MemoryStream())

{

image.Save(ms, format);

return ms.ToArray();

}

}

}

public class ClientProgram

{

//Sends packet to destination

private static void SendMessage(UdpClient udpClient, Byte[] sendBytes, IPEndPoint SendEP)

{

udpClient.Send(sendBytes, sendBytes.Length, SendEP);

}

private static void RecieveMessage(UdpClient udpClient, IPEndPoint RecieveEP)

{

byte[] RecievedData = udpClient.Receive(ref RecieveEP);

Console.WriteLine($"Received broadcast from {RecieveEP} :");

Console.WriteLine(Encoding.ASCII.GetString(RecievedData));

}

public static byte[] Color()

{

byte[] RGB = new byte[3];

string[] ColorArray = new string[] { "red", "blue", "green", "black", "white" };

byte[,] ColorValues = new byte[,] { { 255, 0, 0 },

{ 0, 0, 255 },

{ 0, 255, 0 },

{0, 0, 0 },

{ 255, 255, 255 } };

string ccode = "";

while (true)

{

Console.WriteLine("Input color, for available colors input colors: ");

string input = Console.ReadLine();

if (input == "colors")

{

for (int i = 0; i < 5; i++)

{

ccode = ccode + ColorArray[i] + " ";

}

Console.WriteLine("Available colors are: " + ccode);

continue;

}

for (int i = 0; i < ColorArray.Length; i++)

{

if (ColorArray[i] == input)

{

RGB[0] = ColorValues[i, 0];

RGB[1] = ColorValues[i, 1];

RGB[2] = ColorValues[i, 2];

return RGB;

}

}

Console.WriteLine("Error, such color does not exist!");

}

}

//Command Methods

public static byte[] ClearDisplay(byte command, byte[] RGB)

{

byte[] output = null; Int16 clr; List<byte> sendBytesList;

sendBytesList = new List<byte>();

sendBytesList.Add(command);

sendBytesList.AddRange(RGB);

output = sendBytesList.ToArray();

return output;

}

public static byte[] ThreeVarsConverter(byte command, Int16 x1, Int16 y1, byte[] RGB)

{

byte[] output = null; Int16 clr; List<byte> sendBytesList;

sendBytesList = new List<byte>();

sendBytesList.Add(command);

sendBytesList.AddRange(BitConverter.GetBytes(x1));

sendBytesList.AddRange(BitConverter.GetBytes(y1));

sendBytesList.AddRange(RGB);

output = sendBytesList.ToArray();

return output;

}

public static byte[] FiveVarsConverter(byte command, Int16 var1, Int16 var2, Int16 var3, Int16 var4, byte[] RGB)

{

byte[] output = null; Int16 clr; List<byte> sendBytesList;

sendBytesList = new List<byte>();

sendBytesList.Add(command);

sendBytesList.AddRange(BitConverter.GetBytes(var1));

sendBytesList.AddRange(BitConverter.GetBytes(var2));

sendBytesList.AddRange(BitConverter.GetBytes(var3));

sendBytesList.AddRange(BitConverter.GetBytes(var4));

sendBytesList.AddRange(RGB);

output = sendBytesList.ToArray();

return output;

}

public static byte[] CircleConverter(byte command, Int16 val1, Int16 val2, Int16 val3, byte[] RGB)

{

byte[] output = null; Int16 clr; List<byte> sendBytesList;

sendBytesList = new List<byte>();

sendBytesList.Add(command);

sendBytesList.AddRange(BitConverter.GetBytes(val1));

sendBytesList.AddRange(BitConverter.GetBytes(val2));

sendBytesList.AddRange(BitConverter.GetBytes(val3));

sendBytesList.AddRange(RGB);

output = sendBytesList.ToArray();

return output;

}

public static byte[] TextConverter(byte command, Int16 val1, Int16 val2, Int16 val4, string text, byte[] RGB)

{

List<byte> sendBytesList; byte[] output;

sendBytesList = new List<byte>();

sendBytesList.Add(command);

sendBytesList.AddRange(BitConverter.GetBytes(val1));

sendBytesList.AddRange(BitConverter.GetBytes(val2));

sendBytesList.AddRange(RGB);

sendBytesList.AddRange(BitConverter.GetBytes(val4));

sendBytesList.AddRange(BitConverter.GetBytes(Convert.ToInt16(Encoding.ASCII.GetBytes(text).Length)));

sendBytesList.AddRange(Encoding.ASCII.GetBytes(text));

output = sendBytesList.ToArray();

return output;

}

public static byte[] RoundedRectangleConverter(byte command, Int16 val1, Int16 val2, Int16 val3, Int16 val4, Int16 val5, byte[] RGB)

{

byte[] output = null; List<byte> sendBytesList;

sendBytesList = new List<byte>();

sendBytesList.Add(command);

sendBytesList.AddRange(BitConverter.GetBytes(val1));

sendBytesList.AddRange(BitConverter.GetBytes(val2));

sendBytesList.AddRange(BitConverter.GetBytes(val3));

sendBytesList.AddRange(BitConverter.GetBytes(val4));

sendBytesList.AddRange(BitConverter.GetBytes(val5));

sendBytesList.AddRange(RGB);

output = sendBytesList.ToArray();

return output;

}

public static byte[] ImageConverter(byte command, Int16 val1, Int16 val2, string path)

{

byte[] output = null; List<byte> sendBytesList; Bitmap map; Color color;

map = new Bitmap(path);

sendBytesList = new List<byte>();

sendBytesList.Add(command);

sendBytesList.AddRange(BitConverter.GetBytes(val1));

sendBytesList.AddRange(BitConverter.GetBytes(val2));

sendBytesList.AddRange(BitConverter.GetBytes(Convert.ToInt16(map.Width)));

sendBytesList.AddRange(BitConverter.GetBytes(Convert.ToInt16(map.Height)));

for (int i = 0; i < map.Width; i++)

{

for (int j = 0; j < map.Height; j++)

{

color = map.GetPixel(i, j);

sendBytesList.Add(color.R);

sendBytesList.Add(color.G);

sendBytesList.Add(color.B);

}

}

output = sendBytesList.ToArray();

return output;

}

public static byte[] OrientationConverter(byte command, Int16 rotation)

{

byte[] output = null; List<byte> sendBytesList;

sendBytesList = new List<byte>();

sendBytesList.Add(command);

sendBytesList.AddRange(BitConverter.GetBytes(rotation));

output = sendBytesList.ToArray();

return output;

}

public static Int16 NumberInputCheckAny()

{

int input;

Int16 output;

while (true)

{

Console.WriteLine("Min Number: -32767, Max Number: 32767");

try

{

input = Convert.ToInt32(Console.ReadLine());

}

catch

{

Console.WriteLine("Error, not a number!");

continue;

}

if (input > 32767)

{

Console.WriteLine("Error, number is too big");

continue;

}

else if (input < -32767)

{

Console.WriteLine("Error, number is too small");

continue;

}

else

{

output = Convert.ToInt16(input);

return output;

}

}

}

public static Int16 NumberInputCheckOnlyPositive()

{

int input;

Int16 output;

while (true)

{

Console.WriteLine("Only positive numbers, Max Number: 32767");

try

{

input = Convert.ToInt32(Console.ReadLine());

}

catch

{

Console.WriteLine("Error, not a number!");

continue;

}

if (input > 32767)

{

Console.WriteLine("Error, number is too big");

continue;

}

else if (input < 1)

{

Console.WriteLine("Error, number is not positive");

continue;

}

else

{

output = Convert.ToInt16(input);

return output;

}

}

}

static void Main(string[] args)

{

//Arguments

byte command; Int16 val1, val2, val3, val4, val5, port; string imgname, text; byte[] sendBytes, RGB; string IpAddr; List<Byte> sendBytesList;

Console.WriteLine("Client Begin");

UdpClient udpClient = new UdpClient(0);

Console.WriteLine("Enter Ip Address of the server (Format is x.x.x.x): ");

IpAddr = Console.ReadLine();

Console.WriteLine("Enter the port of the server: ");

port = Convert.ToInt16(Console.ReadLine());

IPAddress serverAddr = IPAddress.Parse(IpAddr);

IPEndPoint SendEP = new IPEndPoint(serverAddr, port);

while (true)

{

Console.WriteLine("Input the command you wish to send to the server, to view the commands enter help, to exit enter exit");

string input = Console.ReadLine();

switch (input)

{

case "help":

Console.WriteLine("Available commands:");

Console.WriteLine("clear display; draw pixel; draw line; draw rectangle; fill rectangle; draw ellipse; fill ellipse; draw circle; fill circle;" +

"draw rounded rectangle; fill rounded rectangle; draw text; draw image; set orientation; get width; get height ");

break;

case "exit":

Environment.Exit(0);

break;

case "clear display":

command = 1;

Console.WriteLine("Clearing display");

RGB = Color();

sendBytes = ClearDisplay(command, RGB);

SendMessage(udpClient, sendBytes, SendEP);//send command;

RecieveMessage(udpClient, SendEP);

break;

case "draw pixel":

command = 2;

Console.WriteLine("Input x1: ");

val1 = NumberInputCheckAny();

Console.WriteLine("Input y1: ");

val2 = NumberInputCheckAny();

RGB = Color();

sendBytes = ThreeVarsConverter(command, val1, val2, RGB);

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

case "draw line":

command = 3;

Console.WriteLine("Input x1: ");

val1 = NumberInputCheckAny();

Console.WriteLine("Input y1: ");

val2 = NumberInputCheckAny();

Console.WriteLine("Input x2: ");

val3 = NumberInputCheckAny();

Console.WriteLine("Input y2: ");

val4 = NumberInputCheckAny();

RGB = Color();

sendBytes = FiveVarsConverter(command, val1, val2, val3, val4, RGB);

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

case "draw rectangle":

command = 4;

Console.WriteLine("Input x1: ");

val1 = NumberInputCheckAny();

Console.WriteLine("Input y1: ");

val2 = NumberInputCheckAny();

Console.WriteLine("Input width: ");

val3 = NumberInputCheckOnlyPositive();

Console.WriteLine("Input height: ");

val4 = NumberInputCheckOnlyPositive();

RGB = Color();

sendBytes = FiveVarsConverter(command, val1, val2, val3, val4, RGB);

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

case "fill rectangle":

command = 5;

Console.WriteLine("Input x1: ");

val1 = NumberInputCheckAny();

Console.WriteLine("Input y1: ");

val2 = NumberInputCheckAny();

Console.WriteLine("Input width: ");

val3 = NumberInputCheckOnlyPositive();

Console.WriteLine("Input height: ");

val4 = NumberInputCheckOnlyPositive();

RGB = Color();

sendBytes = FiveVarsConverter(command, val1, val2, val3, val4, RGB);

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

case "draw ellipse":

command = 6;

Console.WriteLine("Input x1: ");

val1 = NumberInputCheckAny();

Console.WriteLine("Input y1: ");

val2 = NumberInputCheckAny();

Console.WriteLine("Input radius x: ");

val3 = NumberInputCheckOnlyPositive();

Console.WriteLine("Input radius y: ");

val4 = NumberInputCheckOnlyPositive();

RGB = Color();

sendBytes = FiveVarsConverter(command, val1, val2, val3, val4, RGB);

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

case "fill ellipse":

command = 7;

Console.WriteLine("Input x1: ");

val1 = NumberInputCheckAny();

Console.WriteLine("Input y1: ");

val2 = NumberInputCheckAny();

Console.WriteLine("Input radius x: ");

val3 = NumberInputCheckOnlyPositive();

Console.WriteLine("Input radius y: ");

val4 = NumberInputCheckOnlyPositive();

RGB = Color();

sendBytes = FiveVarsConverter(command, val1, val2, val3, val4, RGB);

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

case "draw circle":

command = 8;

Console.WriteLine("Input x1: ");

val1 = NumberInputCheckAny();

Console.WriteLine("Input y1: ");

val2 = NumberInputCheckAny();

Console.WriteLine("Input radius: ");

val3 = NumberInputCheckOnlyPositive();

RGB = Color();

sendBytes = CircleConverter(command, val1, val2, val3, RGB);

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

case "fill circle":

command = 9;

Console.WriteLine("Input x1: ");

val1 = NumberInputCheckAny();

Console.WriteLine("Input y1: ");

val2 = NumberInputCheckAny();

Console.WriteLine("Input radius: ");

val3 = NumberInputCheckOnlyPositive();

RGB = Color();

sendBytes = CircleConverter(command, val1, val2, val3, RGB);

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

case "draw rounded rectangle":

command = 10;

Console.WriteLine("Input x1: ");

val1 = NumberInputCheckAny();

Console.WriteLine("Input y1: ");

val2 = NumberInputCheckAny();

Console.WriteLine("Input width: ");

val3 = NumberInputCheckOnlyPositive();

Console.WriteLine("Input height: ");

val4 = NumberInputCheckOnlyPositive();

Console.WriteLine("Input radius: ");

val5 = NumberInputCheckOnlyPositive();

RGB = Color();

sendBytes = RoundedRectangleConverter(command, val1, val2, val3, val4, val5, RGB);

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

case "fill rounded rectangle":

command = 11;

Console.WriteLine("Input x1: ");

val1 = NumberInputCheckAny();

Console.WriteLine("Input y1: ");

val2 = NumberInputCheckAny();

Console.WriteLine("Input width: ");

val3 = NumberInputCheckOnlyPositive();

Console.WriteLine("Input height: ");

val4 = NumberInputCheckOnlyPositive();

Console.WriteLine("Input radius: ");

val5 = NumberInputCheckOnlyPositive();

RGB = Color();

sendBytes = RoundedRectangleConverter(command, val1, val2, val3, val4, val5, RGB);

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

case "draw text":

command = 12;

Console.WriteLine("Input x1: ");

val1 = NumberInputCheckAny();

Console.WriteLine("Input y1: ");

val2 = NumberInputCheckAny();

Console.WriteLine("Input font size: ");

val3 = NumberInputCheckOnlyPositive(); ;

Console.WriteLine("Input text:\b ");

text = Console.ReadLine();

RGB = Color();

sendBytes = TextConverter(command, val1, val2, val3, text, RGB);

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

case "draw image":

string imgpath;

command = 13;

Console.WriteLine("Input image path with file extension(img must be in desktop)");

imgname = Console.ReadLine();

Console.WriteLine("Input x1: ");

val1 = NumberInputCheckAny();

Console.WriteLine("Input y1: ");

val2 = NumberInputCheckAny();

imgpath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Desktop), imgname);

sendBytes = ImageConverter(command, val1, val2, imgpath);

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

case "set orientation":

command = 14;

while(true)

{

try

{

Console.WriteLine("Choose orientation(In degrees):");

Int16 choice = Convert.ToInt16(Console.ReadLine());

sendBytes = OrientationConverter(command, choice);

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

}

catch

{

Console.WriteLine("Incorrect input, try again");

continue;

}

}

break;

case "get width":

command = 15;

sendBytes = new byte[] { 15 };

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

case "get height":

command = 16 ;

sendBytes = new byte[] { 16 };

SendMessage(udpClient, sendBytes, SendEP);

RecieveMessage(udpClient, SendEP);

break;

default:

Console.WriteLine("Error, command unrecognized, try again");

break;

}

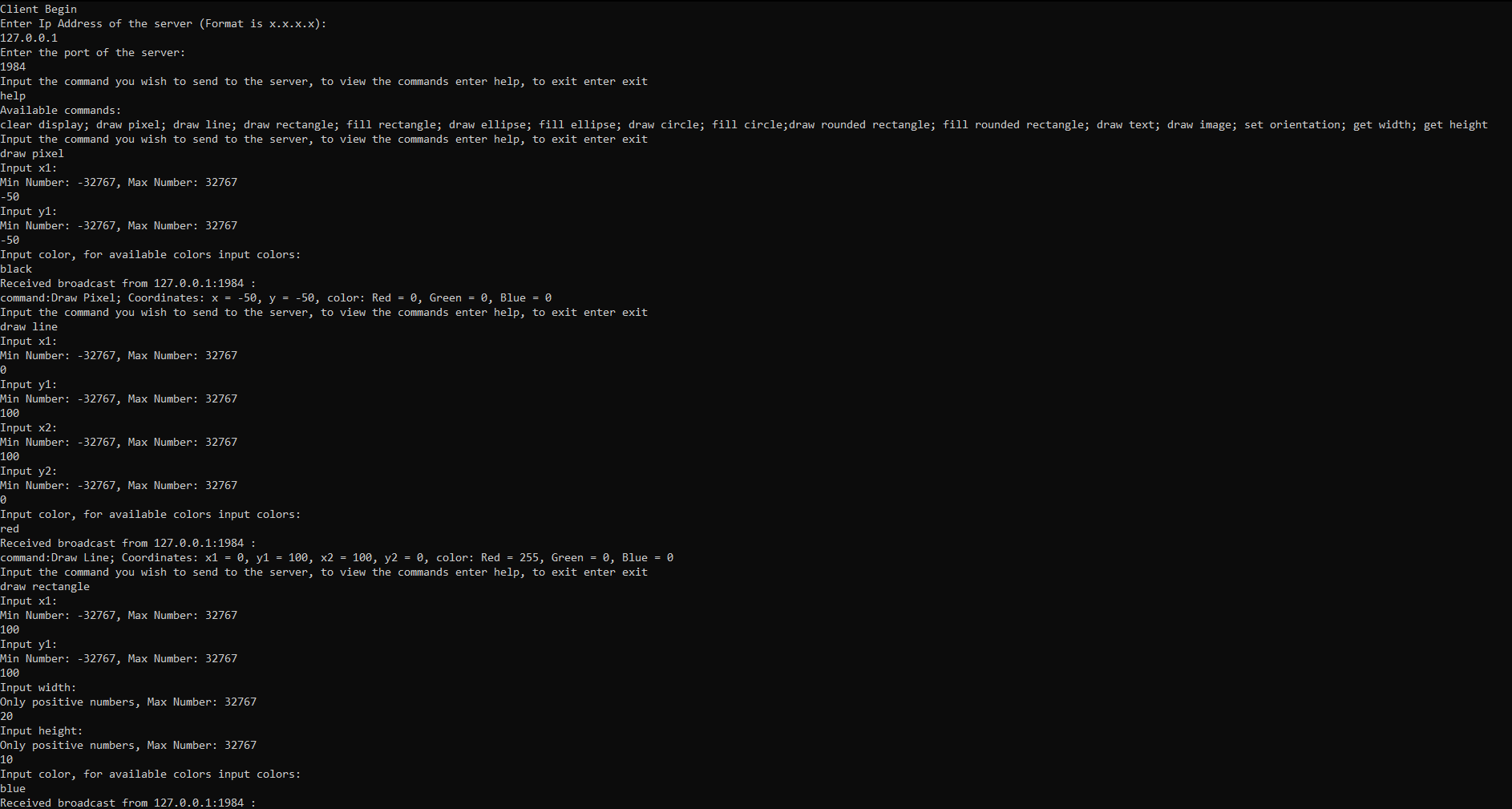
}

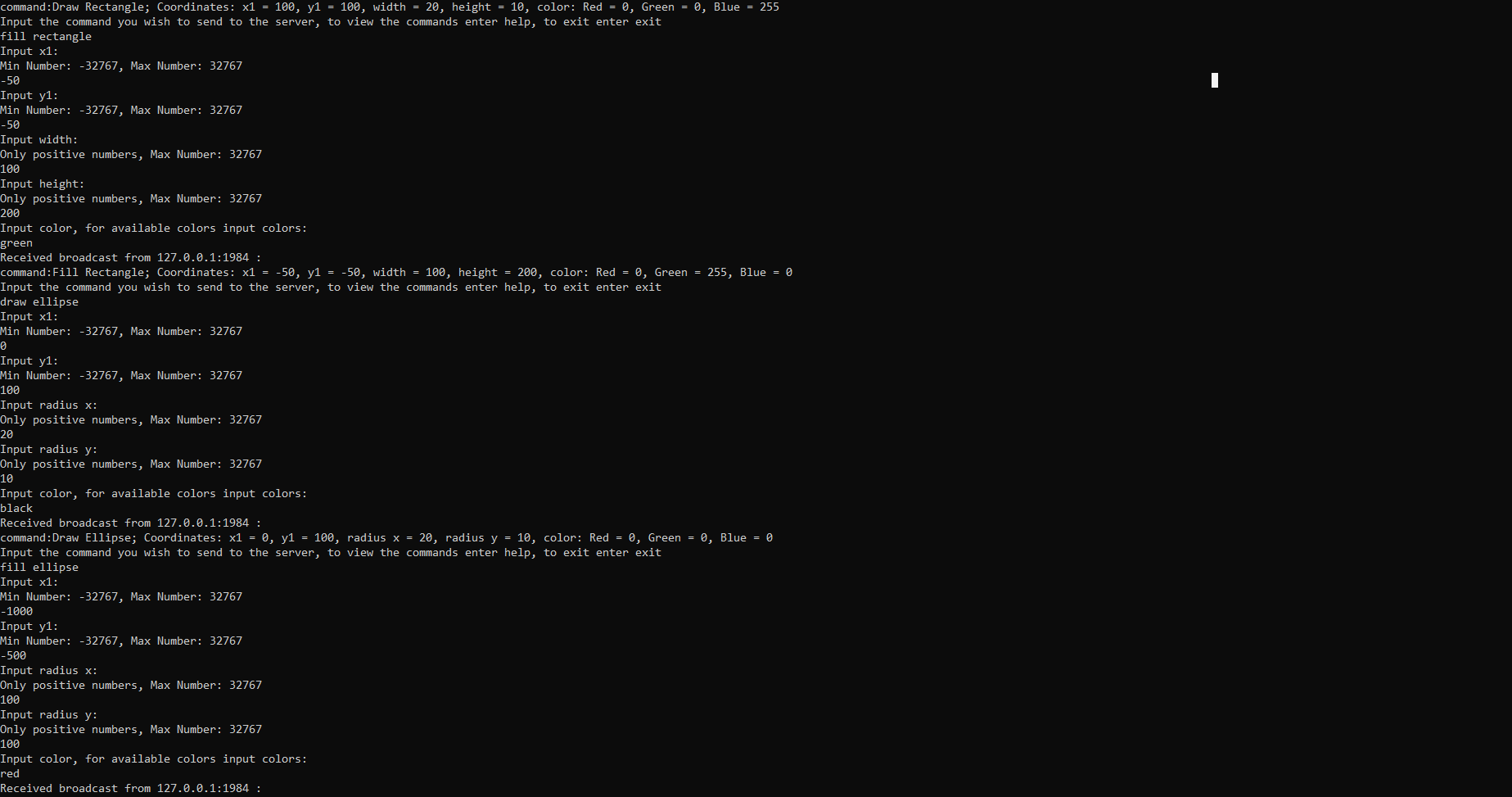
}

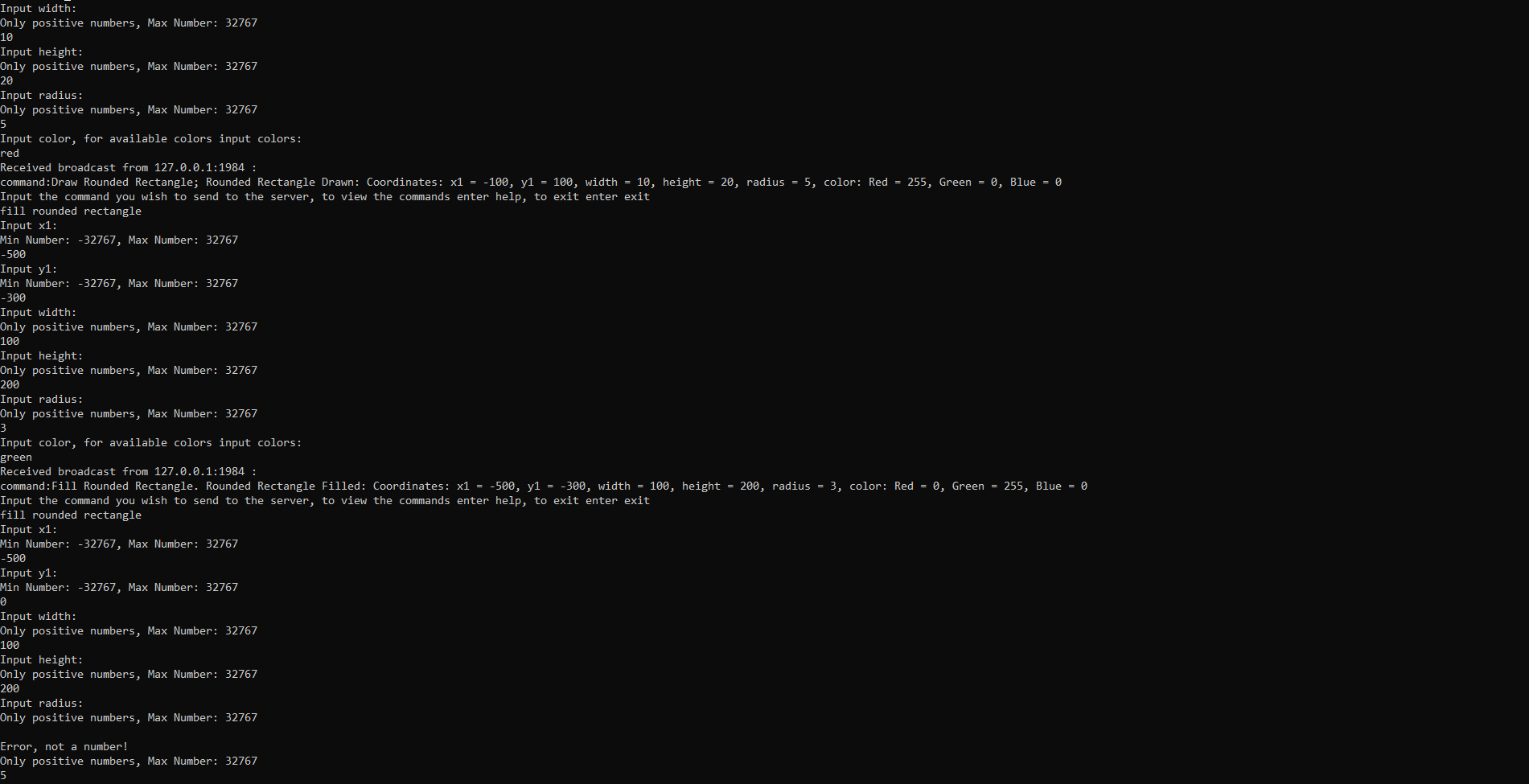
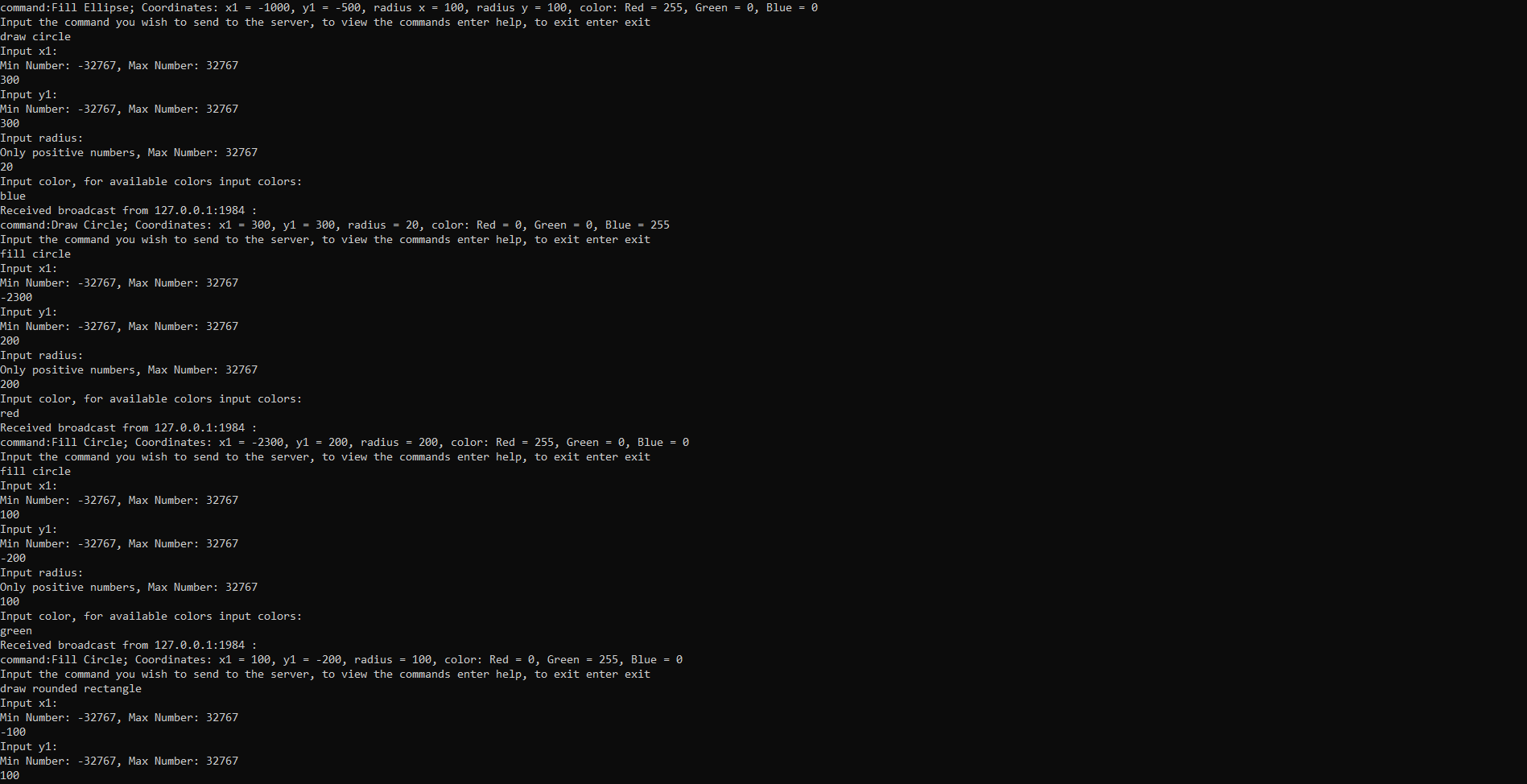
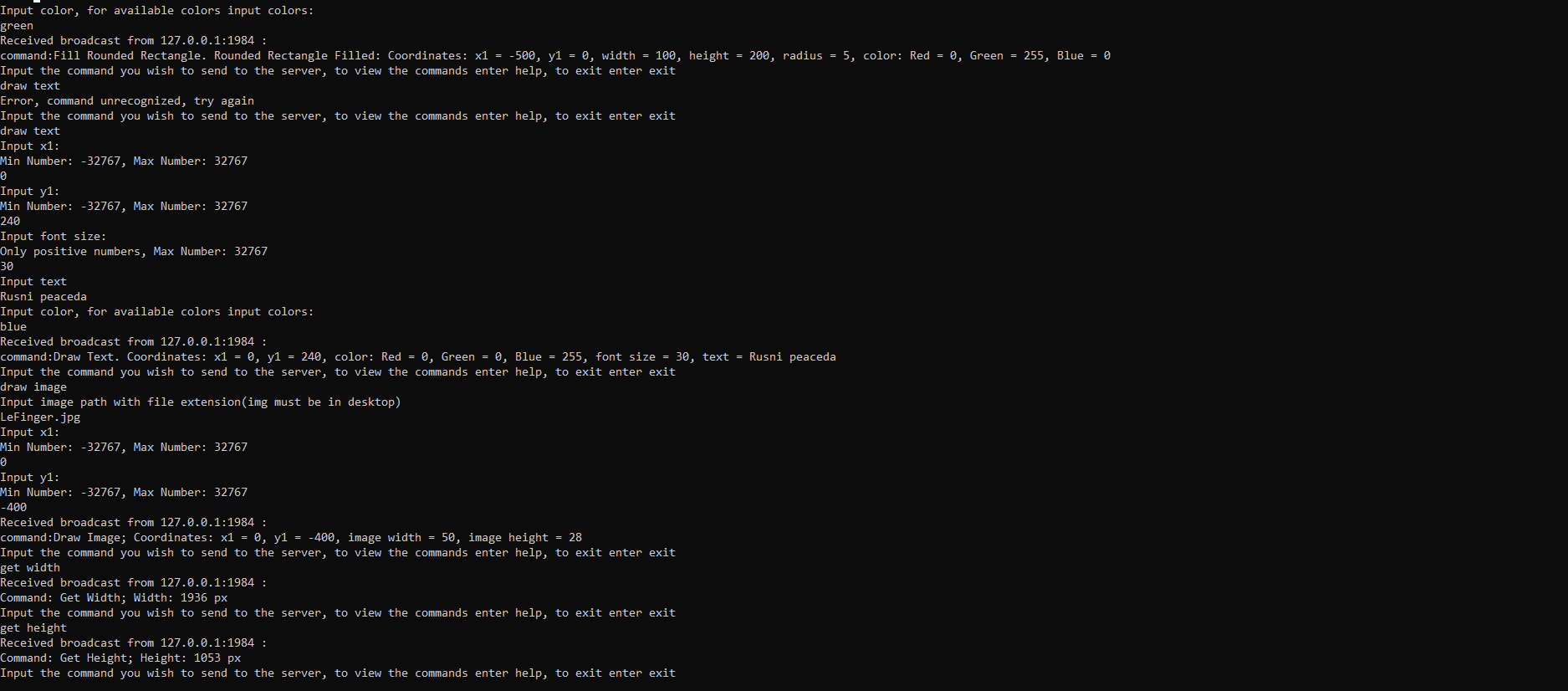
}

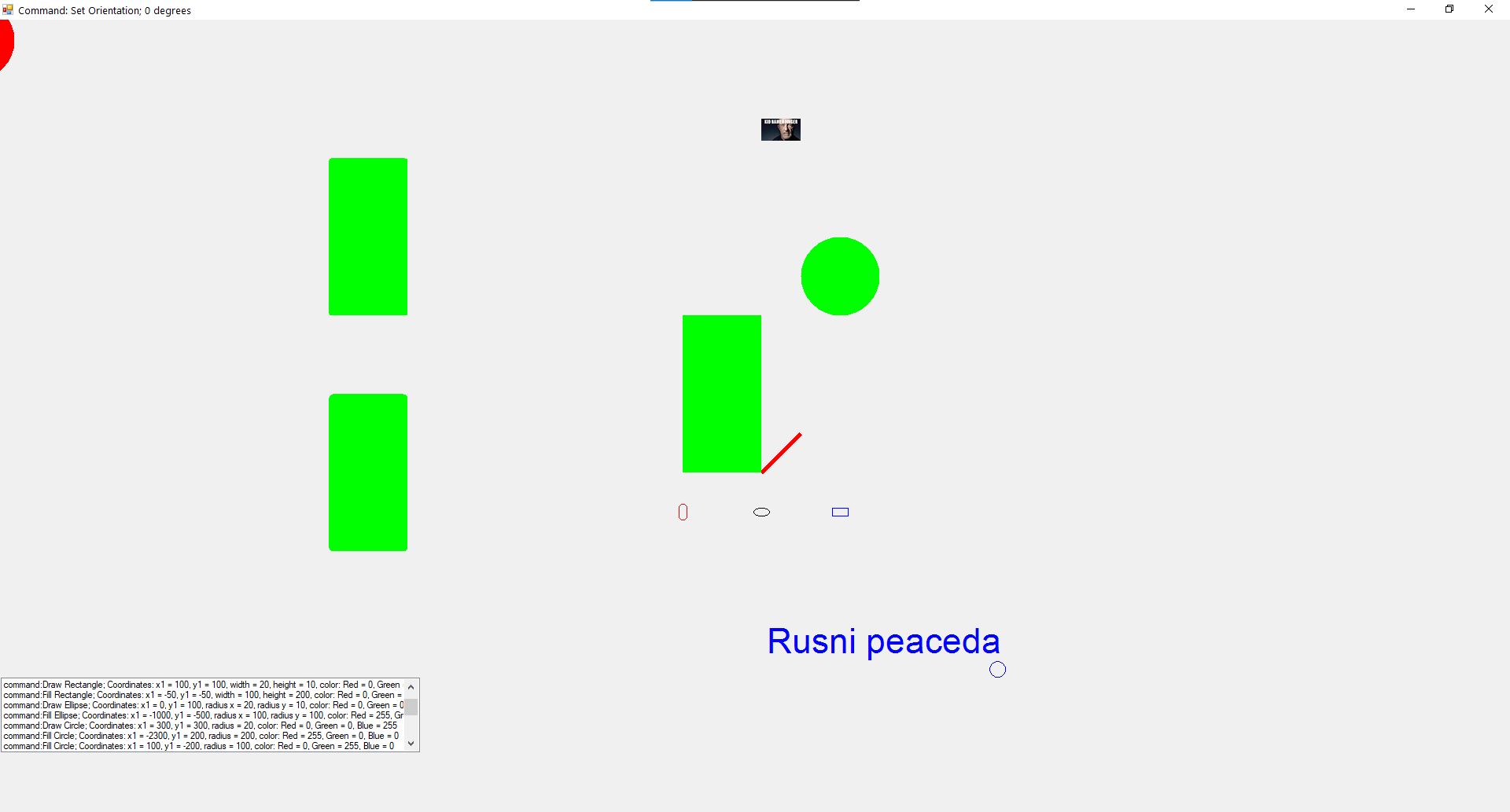
}

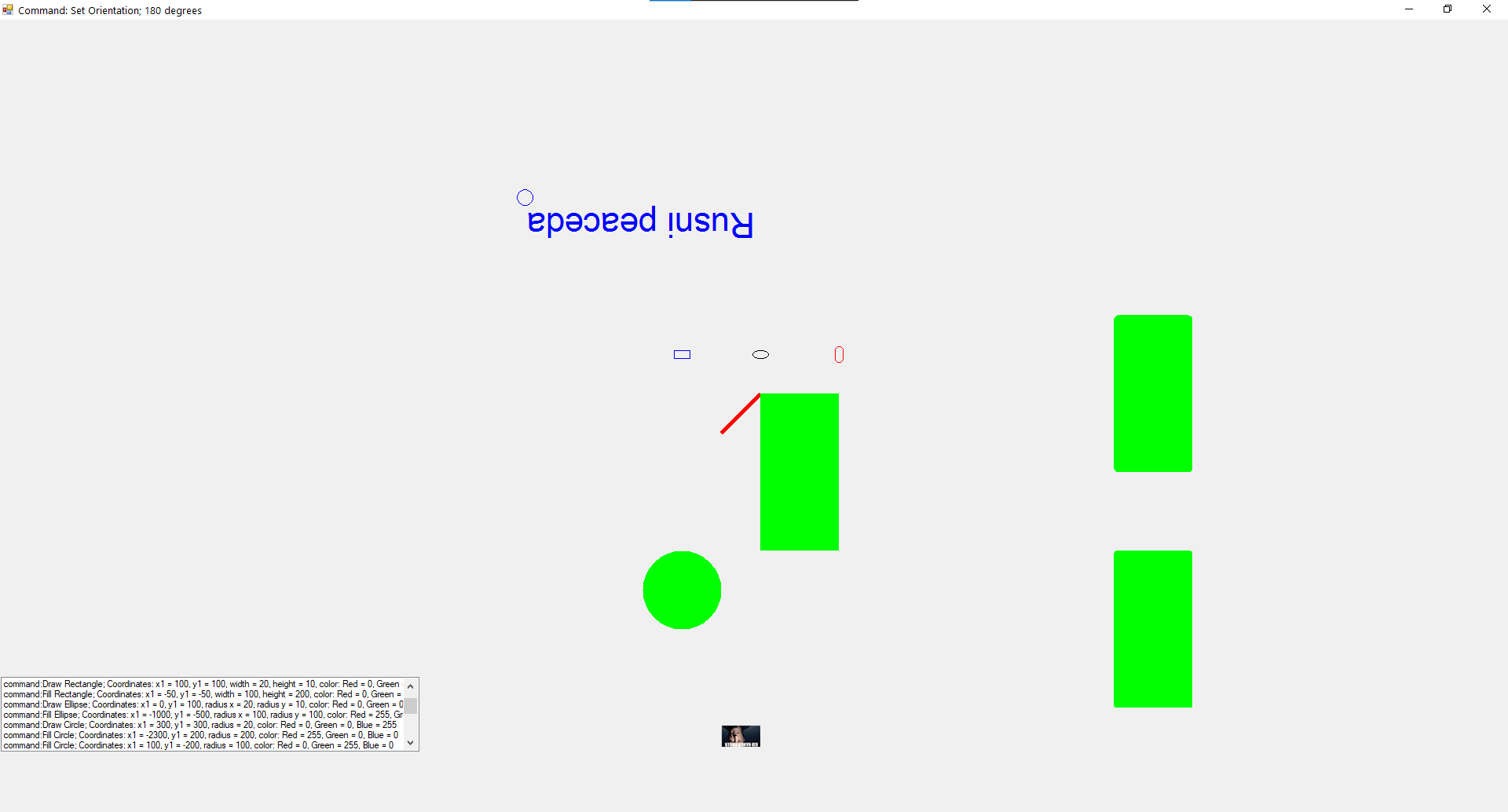
Команди:

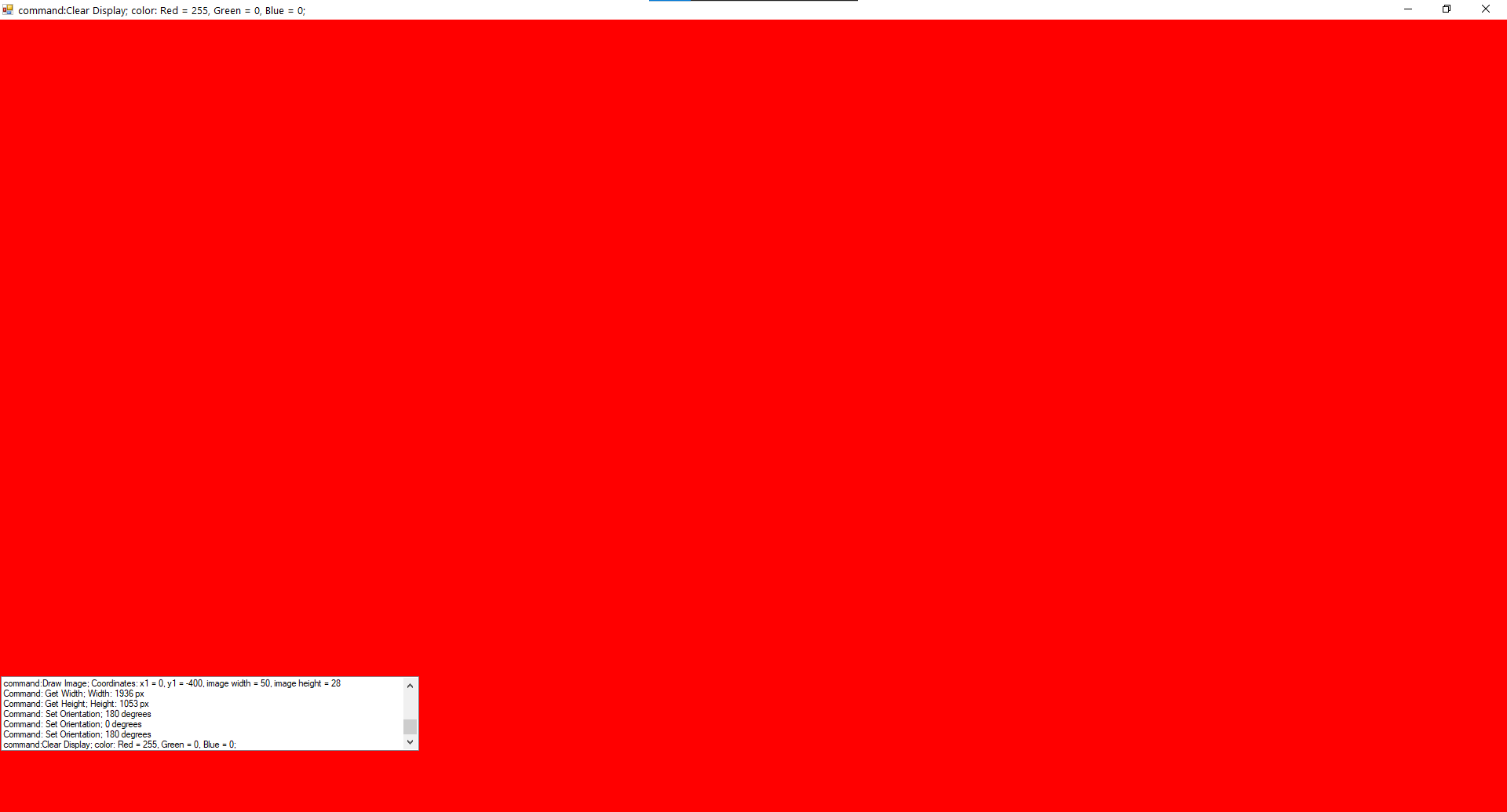
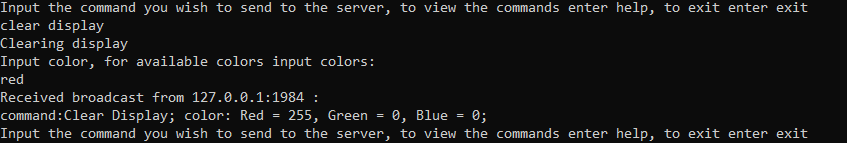












Висновок: Протягом лабораторної роботи, я розробив програму для емуляції дисплейного модуля.