**Лабораторна робота № 2**

Виконав студент

Групи кн21-1

Кончич Даніїл

Варіант 14

**Мета роботи:** 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**

**Лістинг клієнта:**

**Program.cs:**

using System;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.IO;

namespace CSCS1

{

class SendProgram

{

static void Main(string[] args)

{

SendMessage();

}

private static void SendMessage()

{

string remoteAddress = "127.0.0.1";

int port = 8080;

Commands commands = new Commands();

UdpClient sender = new UdpClient(0);

IPEndPoint endPoint = new IPEndPoint(IPAddress.Parse(remoteAddress),

port);

Int16 x0, y0;

Int16 x1, y1;

Int16 radius;

string text;

string hexcolor;

try

{

Console.WriteLine("Type 'help' or '?' for command list");

while (true)

{

Console.Write("Enter command > ");

string commandText = Console.ReadLine();

byte[] commandbyte = new byte[1];

byte[] result = new byte[1] { 0 };

switch (commandText)

{

case "1":

case "clear display":

commandbyte[0] = 1;

hexcolor = ReadHexColor();

result = commands.ClearDisplayEncode(commandbyte[0],

hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "2":

case "draw pixel":

commandbyte[0] = 2;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

hexcolor = ReadHexColor();

result = commands.PixelEncode(commandbyte[0], x0, y0,

hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "3":

case "draw line":

commandbyte[0] = 3;

x0 = ReadNumber("x0", false);

y0 = ReadNumber("y0", false);

x1 = ReadNumber("x1", false);

y1 = ReadNumber("y1", false);

hexcolor = ReadHexColor();

result = commands.FourNumbersEncode(commandbyte[0],

x0, y0, x1, y1, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "4":

case "draw rectangle":

commandbyte[0] = 4; x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

x1 = ReadNumber("width", true);

y1 = ReadNumber("height", true);

hexcolor = ReadHexColor();

result = commands.FourNumbersEncode(commandbyte[0],

x0, y0, x1, y1, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "5":

case "fill rectangle":

commandbyte[0] = 5;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

x1 = ReadNumber("width", true);

y1 = ReadNumber("height", true);

hexcolor = ReadHexColor();

result = commands.FourNumbersEncode(commandbyte[0],

x0, y0, x1, y1, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "6":

case "draw ellipse":

commandbyte[0] = 6;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

x1 = ReadNumber("radius x", true);

y1 = ReadNumber("radius y", true);

hexcolor = ReadHexColor();

result = commands.FourNumbersEncode(commandbyte[0],

x0, y0, x1, y1, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "7":

case "fill ellipse":

commandbyte[0] = 7;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

x1 = ReadNumber("radius x", true);

y1 = ReadNumber("radius y", true);

hexcolor = ReadHexColor();

result = commands.FourNumbersEncode(commandbyte[0],

x0, y0, x1, y1, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "8":

case "draw circle":

commandbyte[0] = 8;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

radius = ReadNumber("radius", true);

hexcolor = ReadHexColor();

result = commands.CircleEncode(commandbyte[0], x0, y0,

radius, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "9":

case "fill circle":

commandbyte[0] = 9;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

radius = ReadNumber("radius", true);

hexcolor = ReadHexColor();

result = commands.CircleEncode(commandbyte[0], x0, y0,

radius, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "10":

case "draw rounded rectangle":

commandbyte[0] = 10;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

x1 = ReadNumber("width", true);

y1 = ReadNumber("height", true);

radius = ReadNumber("radius", true);

hexcolor = ReadHexColor();

result = commands.RoundedRectEncode(commandbyte[0],

x0, y0, x1, y1, radius, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "11":

case "fill rounded rectangle":

commandbyte[0] = 11;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

x1 = ReadNumber("width", true);

y1 = ReadNumber("height", true);

radius = ReadNumber("radius", true);

hexcolor = ReadHexColor();

result = commands.RoundedRectEncode(commandbyte[0],

x0, y0, x1, y1, radius, hexcolor);

sender.Send(result, result.Length, endPoint);

break;

case "12":

case "draw text":

commandbyte[0] = 12;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

hexcolor = ReadHexColor();

x1 = ReadNumber("font number", true);

Console.Write("Enter text > ");

text = Console.ReadLine();

y1 = Convert.ToInt16(text.Length);

result = commands.TextEncode(commandbyte[0], x0, y0,

hexcolor, x1, y1, text);

sender.Send(result, result.Length, endPoint);

break;

case "13":

case "draw image":

commandbyte[0] = 13;

x0 = ReadNumber("x", false);

y0 = ReadNumber("y", false);

x1 = ReadNumber("width", true);

y1 = ReadNumber("height", true);

text = ReadPath(); result = commands.ImageEncode(commandbyte[0], x0, y0,

x1, y1, text);

sender.Send(result, result.Length, endPoint);

break;

case "14":

case "set orientation":

commandbyte[0] = 14;

x0 = ReadNumber("rotation angle", false);

result =

commandbyte.Concat(BitConverter.GetBytes(x0)).ToArray();

sender.Send(result, result.Length, endPoint);

break;

case "15":

case "get width":

commandbyte[0] = 15;

sender.Send(commandbyte, commandbyte.Length,

endPoint);

RecieveMessage(sender, endPoint);

break;

case "16":

case "get height":

commandbyte[0] = 16;

sender.Send(commandbyte, commandbyte.Length,

endPoint);

RecieveMessage(sender, endPoint);

break;

case "17":

case "set pen width":

commandbyte[0] = 17;

x0 = ReadNumber("width", true);

result =

commandbyte.Concat(BitConverter.GetBytes(x0)).ToArray();

sender.Send(result, result.Length, endPoint);

break;

case "help":

case "?":

Console.WriteLine("\nCommands:");

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine(" 1. clear display");

Console.WriteLine(" 2. draw pixel");

Console.WriteLine(" 3. draw line");

Console.WriteLine(" 4. draw rectangle");

Console.WriteLine(" 5. fill rectangle");

Console.WriteLine(" 6. draw ellipse");

Console.WriteLine(" 7. fill ellipse");

Console.WriteLine(" 8. draw circle");

Console.WriteLine(" 9. fill circle");

Console.WriteLine(" 10. draw rounded rectangle");

Console.WriteLine(" 11. fill rounded rectangle");

Console.WriteLine(" 12. draw text");

Console.WriteLine(" 13. draw image");

Console.WriteLine(" 14. set orientation");

Console.WriteLine(" 15. get width");

Console.WriteLine(" 16. get height");

Console.ResetColor();

break;

default:

Console.ForegroundColor = ConsoleColor.Red; Console.WriteLine("Error! Unknown operation! Tryagain.");

Console.ResetColor();

break;

}

Console.WriteLine();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sender.Close();

}

}

public static bool IsStringInHex(string text)

{

return System.Text.RegularExpressions.Regex.IsMatch(text, @"\A\b[0-9afA-F]+\b\Z");

}

private static string ReadHexColor()

{

string str;

while (true)

{

Console.Write("Enter RGB565 color > ");

str = Console.ReadLine();

if (IsStringInHex(str) && str.Length <= 4)

{

break;

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Error! Data is not hexadecimal! Try again.");

Console.ResetColor();

}

}

return str;

}

private static Int16 ReadNumber(string text, bool onlyPositive = false)

{

string str;

Int16 number;

while (true)

{

Console.Write($"Enter {text} > ");

str = Console.ReadLine();

try

{

number = Int16.Parse(str);

if (onlyPositive)

{

if (number < 0)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Error! Bad data! (range 0 to 32767) Try again.");Console.ResetColor();

}

else { break; }

}

else { break; }

}

catch

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Error! Bad data! (range -32768 to 32767) Try again.");

Console.ResetColor();

}

}

return Convert.ToInt16(str);

}

private static string ReadPath()

{

string str;

while (true)

{

Console.Write("Enter path > ");

str = Console.ReadLine();

if (File.Exists(str) && IsImage(str))

{

break;

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Error! File does not exist! Try again.");

Console.ResetColor();

}

}

return @"" + str;

}

public static bool IsImage(string path)

{

return System.Text.RegularExpressions.Regex.IsMatch(path,

@"^.\*\.(jpg|JPG|gif|GIF|png|PNG)$");

}

public static void RecieveMessage(UdpClient sender, IPEndPoint endPoint)

{

byte[] data = sender.Receive(ref endPoint);

Console.WriteLine($"Recieved value: {BitConverter.ToInt16(data, 0)}");

}

}

}

**MyCommands.cs:**

using System;

using System.Drawing;

using System.Linq;

using System.Text;

namespace CSCS1

{

public class Commands

{

//\*\*\*\*\*\*\*\*\*\*\*\*ClearDisplay\*\*\*\*\*\*\*\*\*\*\*\*

public byte[] ClearDisplayEncode(byte command, string hexcolor)

{

byte[] commandbyte = { command };

Int16 color = Convert.ToInt16(hexcolor, 16);

return commandbyte.Concat(BitConverter.GetBytes(color)).ToArray();

}

public void ClearDisplayDecode(byte[] result, out byte command, out string

hexcolor)

{

command = result[0];

hexcolor = ByteToHexColor(result, 1);

}

//\*\*\*\*\*\*\*\*\*\*\*\*Pixel\*\*\*\*\*\*\*\*\*\*\*\*

public byte[] PixelEncode(byte command, Int16 x0, Int16 y0, string

hexcolor)

{

byte[] commandbyte = { command };

Int16 color = Convert.ToInt16(hexcolor, 16);

return

commandbyte.Concat(BitConverter.GetBytes(x0)).Concat(BitConverter.GetBytes(y0)).Concat(BitConverter.GetBytes(color)).ToArray();

}

public void PixelDecode(byte[] result, out byte command, out Int16 x0, out

Int16 y0, out string hexcolor)

{

command = result[0];

x0 = BitConverter.ToInt16(result, 1);

y0 = BitConverter.ToInt16(result, 3);

hexcolor = ByteToHexColor(result, 5);

}

//\*\*\*\*\*\*\*\*\*\*\*\*FourNumbers\*\*\*\*\*\*\*\*\*\*\*\*

public byte[] FourNumbersEncode(byte command, Int16 x0, Int16 y0, Int16

x1, Int16 y1, string hexcolor)

{

byte[] commandbyte = { command };

Int16 color = Convert.ToInt16(hexcolor, 16);

return

commandbyte.Concat(BitConverter.GetBytes(x0)).Concat(BitConverter.GetBytes(y0)).Concat(BitConverter.GetBytes(x1)).Concat(BitConverter.GetBytes(y1)).Concat(BitConverter.GetBytes(color)).ToArray();

}

public void FourNumbersDecode(byte[] result, out byte command, out Int16

x0, out Int16 y0, out Int16 x1, out Int16 y1, out string hexcolor)

{

command = result[0];

x0 = BitConverter.ToInt16(result, 1);

y0 = BitConverter.ToInt16(result, 3);

x1 = BitConverter.ToInt16(result, 5);

y1 = BitConverter.ToInt16(result, 7);

hexcolor = ByteToHexColor(result, 9);

}

//\*\*\*\*\*\*\*\*\*\*\*\*Circle\*\*\*\*\*\*\*\*\*\*\*\*

public byte[] CircleEncode(byte command, Int16 x0, Int16 y0, Int16 radius,

string hexcolor)

{

byte[] commandbyte = { command };

Int16 color = Convert.ToInt16(hexcolor, 16);

return

commandbyte.Concat(BitConverter.GetBytes(x0)).Concat(BitConverter.GetBytes(y0)).Concat(BitConverter.GetBytes(radius)).Concat(BitConverter.GetBytes(color)).ToArray()

;

}

public void CircleDecode(byte[] result, out byte command, out Int16 x0,

out Int16 y0, out Int16 radius, out string hexcolor)

{

command = result[0];

x0 = BitConverter.ToInt16(result, 1);

y0 = BitConverter.ToInt16(result, 3);

radius = BitConverter.ToInt16(result, 5);

hexcolor = ByteToHexColor(result, 7);

}

//\*\*\*\*\*\*\*\*\*\*\*\*RoundedRect\*\*\*\*\*\*\*\*\*\*\*\*

public byte[] RoundedRectEncode(byte command, Int16 x0, Int16 y0, Int16

x1, Int16 y1, Int16 radius, string hexcolor)

{

byte[] commandbyte = { command };

Int16 color = Convert.ToInt16(hexcolor, 16);

return

commandbyte.Concat(BitConverter.GetBytes(x0)).Concat(BitConverter.GetBytes(y0)).Concat(BitConverter.GetBytes(x1)).Concat(BitConverter.GetBytes(y1)).Concat(BitConverter.GetBytes(radius)).Concat(BitConverter.GetBytes(color)).ToArray();

}

public void RoundedRectDecode(byte[] result, out byte command, out Int16

x0, out Int16 y0, out Int16 x1, out Int16 y1, out Int16 radius, out string

hexcolor)

{

command = result[0];

x0 = BitConverter.ToInt16(result, 1);

y0 = BitConverter.ToInt16(result, 3);

x1 = BitConverter.ToInt16(result, 5);

y1 = BitConverter.ToInt16(result, 7);

radius = BitConverter.ToInt16(result, 9);

hexcolor = ByteToHexColor(result, 11);

}

//\*\*\*\*\*\*\*\*\*\*\*\*Text\*\*\*\*\*\*\*\*\*\*\*\*

public byte[] TextEncode(byte command, Int16 x0, Int16 y0, string

hexcolor, Int16 x1, Int16 y1, string text)

{

byte[] commandbyte = { command };

Int16 color = Convert.ToInt16(hexcolor, 16);

return

commandbyte.Concat(BitConverter.GetBytes(x0)).Concat(BitConverter.GetBytes(y0)).Concat(BitConverter.GetBytes(color)).Concat(BitConverter.GetBytes(x1)).Concat(BitConverter.GetBytes(y1)).Concat(Encoding.Unicode.GetBytes(text)).ToArray();

}

public void TextDecode(byte[] result, out byte command, out Int16 x0, out

Int16 y0, out string hexcolor, out Int16 x1, out Int16 y1, out string text)

{

command = result[0];

x0 = BitConverter.ToInt16(result, 1);

y0 = BitConverter.ToInt16(result, 3);

hexcolor = ByteToHexColor(result, 5);

x1 = BitConverter.ToInt16(result, 7);

y1 = BitConverter.ToInt16(result, 9);

text = Encoding.Unicode.GetString(result.Skip(11).Take(y1 \*

2).ToArray());

}

//\*\*\*\*\*\*\*\*\*\*\*\*Image\*\*\*\*\*\*\*\*\*\*\*\*

public byte[] ImageEncode(byte command, Int16 x0, Int16 y0, Int16 x1,

Int16 y1, string data)

{

byte[] commandbyte = { command };

Color[] colors = ColorsEncode(new Bitmap(data, true), x1, y1);

byte[] byteColors = ColorsToByte(colors);

return

commandbyte.Concat(BitConverter.GetBytes(x0)).Concat(BitConverter.GetBytes(y0)).Concat(BitConverter.GetBytes(x1)).Concat(BitConverter.GetBytes(y1)).Concat(byteColors).ToArray();

}

public void ImageDecode(byte[] result, out byte command, out Int16 x0, out

Int16 y0, out Int16 x1, out Int16 y1, out Color[,] colors)

{

command = result[0];

x0 = BitConverter.ToInt16(result, 1);

y0 = BitConverter.ToInt16(result, 3);

x1 = BitConverter.ToInt16(result, 5);

y1 = BitConverter.ToInt16(result, 7);

colors = ByteToColors(result.Skip(9).Take(x1 \* y1 \* 4).ToArray(), x1,

y1);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SECONDARY FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public static string ByteToHexColor(byte[] value, int startIndex)

{

Int16 color = BitConverter.ToInt16(value, startIndex);

return color.ToString("X");

}

public static Color[] ColorsEncode(Bitmap source, Int16 w, Int16 h)

{

Bitmap bmp = new Bitmap(source, w, h); Color[] result = new Color[w \* h];

int counter = 0;

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

{

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

{

result[counter] = bmp.GetPixel(j, i);

counter++;

}

}

return result;

}

public static byte[] ColorsToByte(Color[] colors)

{

int length = colors.Length;

byte[] result = new byte[0];

byte[] Combine(byte[] first, byte[] second)

{

byte[] ret = new byte[first.Length + second.Length];

Buffer.BlockCopy(first, 0, ret, 0, first.Length);

Buffer.BlockCopy(second, 0, ret, first.Length, second.Length);

return ret;

}

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

{

result = Combine(result,

BitConverter.GetBytes(colors[i].ToArgb()));

}

return result;

}

public static Color[,] ByteToColors(byte[] byteColors, Int16 w, Int16 h)

{

Color[,] result = new Color[w, h];

int counter = 0;

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

{

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

{

result[j, i] = Color.FromArgb(BitConverter.ToInt32(byteColors,

4 \* counter));

counter++;

}

}

return result;

}

}

}

**Лістинг сервера:**

**Form1.cs:**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using System.Net;

using System.Net.Sockets;

using System.Threading;

using System.Drawing.Drawing2D;

namespace CSCS2\_Forms

{

public partial class Form1 : Form

{

static Int16 rotation = 0;

static Int16 penWidth = 2;

static List<Lines> lines = new List<Lines>();

static List<Pixels> pixels = new List<Pixels>();

static List<Rectangles> rectangles = new List<Rectangles>();

static List<Ellipses> ellipses = new List<Ellipses>();

static List<RoundedRectangle> roundedRectangles = new

List<RoundedRectangle>();

static List<Texts> texts = new List<Texts>();

static List<Pictures> pictures = new List<Pictures>(); public Form1()

{

InitializeComponent();

try

{

new Thread(new ThreadStart(ReceiveMessage)).Start();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CLASSES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public class Pixels

{

public Int16 x0;

public Int16 y0;

public Color argb;

public Pixels(Int16 \_x0, Int16 \_y0, Color \_argb)

{

this.x0 = \_x0;

this.y0 = \_y0;

this.argb = \_argb;

}

}

public class Lines

{

public Int16 x0;

public Int16 y0;

public Int16 x1;

public Int16 y1;

public Color argb;

public Lines(Int16 \_x0, Int16 \_y0, Int16 \_x1, Int16 \_y1, Color \_argb)

{

this.x0 = \_x0;

this.y0 = \_y0;

this.x1 = \_x1;

this.y1 = \_y1;

this.argb = \_argb;

}

}

public class Rectangles

{

public Int16 x0;

public Int16 y0;

public Int16 w;

public Int16 h;

public Color argb;

public bool isfilled;

public Rectangles(Int16 \_x0, Int16 \_y0, Int16 \_w, Int16 \_h, Color

\_argb, bool \_isfilled)

{

this.x0 = \_x0;

this.y0 = \_y0;

this.w = \_w;

this.h = \_h;

this.argb = \_argb;

this.isfilled = \_isfilled;

}

}

public class Ellipses

{

public Int16 x0;

public Int16 y0;

public Int16 radius\_x; public Int16 radius\_y;

public Color argb;

public bool isfilled;

public Ellipses(Int16 \_x0, Int16 \_y0, Int16 \_radius\_x, Int16

\_radius\_y, Color \_argb, bool \_isfilled)

{

this.x0 = \_x0;

this.y0 = \_y0;

this.radius\_x = \_radius\_x;

this.radius\_y = \_radius\_y;

this.argb = \_argb;

this.isfilled = \_isfilled;

}

}

public class RoundedRectangle

{

public Int16 x0;

public Int16 y0;

public Int16 w;

public Int16 h;

public Int16 radius;

public Color argb;

public bool isfilled;

public RoundedRectangle(Int16 \_x0, Int16 \_y0, Int16 \_w, Int16 \_h,

Int16 \_radius, Color \_argb, bool \_isfilled)

{

this.x0 = \_x0;

this.y0 = \_y0;

this.w = \_w;

this.h = \_h;

this.radius = \_radius;

this.argb = \_argb;

this.isfilled = \_isfilled;

}

}

public class Texts

{

public Int16 x0;

public Int16 y0;

public Color argb;

public Int16 fontSize;

public string text;

public Texts(Int16 \_x0, Int16 \_y0, Color \_argb, Int16 \_fontSize,

string \_text)

{

this.x0 = \_x0;

this.y0 = \_y0;

this.argb = \_argb;

this.fontSize = \_fontSize;

this.text = \_text;

}

}

/\*public class TextLines : Texts

{

public List<int[,]> symbols = new List<int[,]>();

public TextLines(Int16 \_x0, Int16 \_y0, Color \_argb, Int16 \_fontSize,

string \_text)

: base(\_x0, \_y0, \_argb, \_fontSize, \_text)

{

Chars s = new Chars();

short x = x0;

foreach (var symbol in text)

{

symbols.Add(s.GetCharCoords(symbol, x, y0, fontSize));

x = Convert.ToInt16(x + fontSize \* 0.8);

}

}

}\*/

public class Pictures

{

public Int16 x0;

public Int16 y0;

public Int16 w;

public Int16 h;

public Color[,] argb;

public Pictures(Int16 \_x0, Int16 \_y0, Int16 \_w, Int16 \_h, Color[,]

\_argb)

{

this.x0 = \_x0;

this.y0 = \_y0;

this.w = \_w;

this.h = \_h;

this.argb = \_argb;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RENDERING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

protected override void OnPaint(PaintEventArgs e)

{

Graphics graphics = e.Graphics;

graphics.SmoothingMode = SmoothingMode.HighQuality;

graphics.InterpolationMode = InterpolationMode.HighQualityBicubic;

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

graphics.RotateTransform(rotation);

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

foreach (var pixel in pixels.ToArray())

{

graphics.FillRectangle(new SolidBrush(pixel.argb), pixel.x0 +

this.Width / 2, pixel.y0 + this.Height / 2, 1, 1);

}

foreach (var line in lines.ToList())

{

graphics.DrawLine(new Pen(line.argb, penWidth), line.x0 +

this.Width / 2, line.y0 + this.Height / 2, line.x1 + this.Width / 2, line.y1 +

this.Height / 2);

}

foreach (var rectangle in rectangles.ToList())

{

if (rectangle.isfilled)

{

graphics.FillRectangle(new SolidBrush(rectangle.argb),

rectangle.x0 + this.Width / 2 - rectangle.w / 2, rectangle.y0 + this.Height / 2 -

rectangle.h / 2, rectangle.w, rectangle.h);

}

else

{

graphics.DrawRectangle(new Pen(rectangle.argb, penWidth),

rectangle.x0 + this.Width / 2 - rectangle.w / 2, rectangle.y0 + this.Height / 2 -

rectangle.h / 2, rectangle.w, rectangle.h);

}

}

foreach (var ellipse in ellipses.ToList())

{

if (ellipse.isfilled)

{

graphics.FillEllipse(new SolidBrush(ellipse.argb), ellipse.x0

+ this.Width / 2 - ellipse.radius\_x / 2, ellipse.y0 + this.Height / 2 -

ellipse.radius\_y / 2, ellipse.radius\_x, ellipse.radius\_y);

}

else

{

graphics.DrawEllipse(new Pen(ellipse.argb, penWidth),

ellipse.x0 + this.Width / 2 - ellipse.radius\_x / 2, ellipse.y0 + this.Height / 2 -

ellipse.radius\_y / 2, ellipse.radius\_x, ellipse.radius\_y);

}

}

foreach (var roundedRectangle in roundedRectangles.ToList())

{

if (roundedRectangle.isfilled)

{

graphics.FillPath(new SolidBrush(roundedRectangle.argb),

RoundedRect(new Rectangle(roundedRectangle.x0 + this.Width / 2 -

roundedRectangle.w / 2, roundedRectangle.y0 + this.Height / 2 - roundedRectangle.h

/ 2, roundedRectangle.w, roundedRectangle.h), roundedRectangle.radius));

}

else

{

graphics.DrawPath(new Pen(roundedRectangle.argb, penWidth),

RoundedRect(new Rectangle(roundedRectangle.x0 + this.Width / 2 -

roundedRectangle.w / 2, roundedRectangle.y0 + this.Height / 2 - roundedRectangle.h

/ 2, roundedRectangle.w, roundedRectangle.h), roundedRectangle.radius));

}

}

foreach (var text in texts.ToList())

{

graphics.DrawString(text.text, new Font("Arial", text.fontSize),

new SolidBrush(text.argb), text.x0 + this.Width / 2, text.y0 + this.Height / 2,

new StringFormat());

}

foreach (var picture in pictures.ToList())

{

graphics.SmoothingMode = SmoothingMode.Default;

Int16 x = picture.x0;

Int16 y = picture.y0;

for (int i = 0; i < picture.h; i++)

{

x = picture.x0;

for (int j = 0; j < picture.w; j++)

{

graphics.FillRectangle(new SolidBrush(picture.argb[j, i]),

x + this.Width / 2, y + this.Height / 2, 3, 3);

x += 3;

}

y += 3;

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RECEIVE MESSAGE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

private void ReceiveMessage()

{

int port = 8080;

CSCS1.Commands commands = new CSCS1.Commands();

UdpClient receiver = new UdpClient(port);

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

IPEndPoint iPEndPoint;

byte commandNum;

byte command;

Int16 x0, y0;

Int16 x1, y1;

Int16 radius;

string text;

string hexcolor;

Color argb;

try

{

while (true)

{

byte[] data = receiver.Receive(ref remoteIp);

commandNum = data[0];

switch (commandNum)

{

case 1:

commands.ClearDisplayDecode(data, out command, out

hexcolor);

Console.WriteLine($"Recieved command: clear display; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

DeleteAllGraphics();

this.BackColor = argb;

Invalidate();

break;

case 2:

commands.PixelDecode(data, out command, out x0, out

y0, out hexcolor);

Console.WriteLine($"Recieved command: draw pixel; x: { x0}; y: { y0}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

pixels.Add(new Pixels(x0, y0, argb));

Invalidate();

break;

case 3:

commands.FourNumbersDecode(data, out command, out x0,

out y0, out x1, out y1, out hexcolor);

Console.WriteLine($"Recieved command: draw line; x0: { x0}; y0: { y0}; x1: { x1}; y1: { y1}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

lines.Add(new Lines(x0, y0, x1, y1, argb));

Invalidate();

break;

case 4:

commands.FourNumbersDecode(data, out command, out x0,

out y0, out x1, out y1, out hexcolor);

Console.WriteLine($"Recieved command: draw rectangle; x: { x0}; y: { y0}; width: { x1}; height: { y1}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

rectangles.Add(new Rectangles(x0, y0, x1, y1, argb,

false));

Invalidate();

break;

case 5:

commands.FourNumbersDecode(data, out command, out x0,

out y0, out x1, out y1, out hexcolor);

Console.WriteLine($"Recieved command: fill rectangle; x: { x0}; y: { y0}; width: { x1}; height: { y1}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

rectangles.Add(new Rectangles(x0, y0, x1, y1, argb,

true));

Invalidate();

break;

case 6:

commands.FourNumbersDecode(data, out command, out x0,

out y0, out x1, out y1, out hexcolor);

Console.WriteLine($"Recieved command: draw ellipse; x: { x0}; y: { y0}; radius x: { x1}; radius y: { y1}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

ellipses.Add(new Ellipses(x0, y0, x1, y1, argb,

false));

Invalidate();

break;

case 7:

commands.FourNumbersDecode(data, out command, out x0,

out y0, out x1, out y1, out hexcolor);

Console.WriteLine($"Recieved command: fill ellipse; x: { x0}; y: { y0}; radius x: { x1}; radius y: { y1}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

ellipses.Add(new Ellipses(x0, y0, x1, y1, argb,

true));

Invalidate();

break;

case 8:

commands.CircleDecode(data, out command, out x0, out

y0, out radius, out hexcolor);

Console.WriteLine($"Recieved command: draw circle; x: { x0}; y: { y0}; radius: { radius}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

ellipses.Add(new Ellipses(x0, y0, radius, radius,

argb, false));

Invalidate();

break;

case 9:

commands.CircleDecode(data, out command, out x0, out

y0, out radius, out hexcolor);

Console.WriteLine($"Recieved command: fill circle; x: { x0}; y: { y0}; radius: { radius}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

ellipses.Add(new Ellipses(x0, y0, radius, radius,

argb, true));

Invalidate();

break;

case 10:

commands.RoundedRectDecode(data, out command, out x0,

out y0, out x1, out y1, out radius, out hexcolor);

Console.WriteLine($"Recieved command: draw rounded rectangle; x: { x0}; y: { y0}; width: { x1}; height: { y1}; radius: { radius}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

roundedRectangles.Add(new RoundedRectangle(x0, y0, x1,

y1, radius, argb, false));

Invalidate();

break;

case 11:

commands.RoundedRectDecode(data, out command, out x0,

out y0, out x1, out y1, out radius, out hexcolor);

Console.WriteLine($"Recieved command: fill rounded rectangle; x: { x0}; y: { y0}; width: { x1}; height: { y1}; radius: { radius}; color: 0x{ hexcolor}; ");

argb = ColorConvert(hexcolor);

roundedRectangles.Add(new RoundedRectangle(x0, y0, x1,

y1, radius, argb, true));

Invalidate();

break;

case 12:

commands.TextDecode(data, out command, out x0, out y0,

out hexcolor, out x1, out y1, out text);

Console.WriteLine($"Recieved command: draw text; x: { x0}; y: { y0}; color: 0x{ hexcolor}; font number: { x1}; length: { y1}; text:{ text}; ");

argb = ColorConvert(hexcolor);

texts.Add(new Texts(x0, y0, argb, x1, text));

Invalidate();

break;

case 13:

commands.ImageDecode(data, out command, out x0, out

y0, out x1, out y1, out Color[,] colors);

Console.WriteLine($"Recieved command: draw image; x: { x0}; y: { y0}; width: { x1}; height: { y1}; colors: ");

pictures.Add(new Pictures(x0, y0, x1, y1, colors)); Invalidate();

break;

case 14:

rotation =

BitConverter.ToInt16(data.Skip(1).Take(2).ToArray(), 0);

Console.WriteLine($"Recieved command: set orientation; rotation angle: { rotation}; ");

Invalidate();

break;

case 15:

data =

BitConverter.GetBytes(Convert.ToInt16(this.Width));

Console.WriteLine($"Recieved command: get width;");

iPEndPoint = new IPEndPoint(remoteIp.Address,

remoteIp.Port);

receiver.Send(data, data.Length, iPEndPoint);

break;

case 16:

data =

BitConverter.GetBytes(Convert.ToInt16(this.Height));

Console.WriteLine($"Recieved command: get height;");

iPEndPoint = new IPEndPoint(remoteIp.Address,

remoteIp.Port);

receiver.Send(data, data.Length, iPEndPoint);

break;

case 17:

penWidth =

BitConverter.ToInt16(data.Skip(1).Take(2).ToArray(), 0);

Console.WriteLine($"Recieved command: set pen width; width: { penWidth}; ");

Invalidate();

break;

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

receiver.Close();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SECONDARY FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static void DeleteAllGraphics()

{

pixels.Clear();

lines.Clear();

rectangles.Clear();

ellipses.Clear();

roundedRectangles.Clear();

texts.Clear();

pictures.Clear();

}

static public Color ColorConvert(string hexcolor)

{

Int16 color = Convert.ToInt16(hexcolor, 16);

string bits = Convert.ToString(color, 2).PadLeft(16, '0');

int R = Convert.ToInt32(bits.Substring(0, 5).PadRight(8, '0'), 2);

int G = Convert.ToInt32(bits.Substring(5, 6).PadRight(8, '0'), 2);

int B = Convert.ToInt32(bits.Substring(11, 5).PadRight(8, '0'), 2);

return Color.FromArgb(R, G, B);

}

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;

}

// top left arc

path.AddArc(arc, 180, 90);

// top right arc

arc.X = bounds.Right - diameter;

path.AddArc(arc, 270, 90);

// bottom right arc

arc.Y = bounds.Bottom - diameter;

path.AddArc(arc, 0, 90);

// bottom left arc

arc.X = bounds.Left;

path.AddArc(arc, 90, 90);

path.CloseFigure();

return path;

}

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

{

Invalidate();

}

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

{ }

}

}

**Form1.Designer.cs:**

namespace CSCS2\_Forms

{

partial class Form1

{

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

/// <param name="disposing">истинно, если управляемый ресурс должен быть удален; иначе ложно.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Код, автоматически созданный конструктором форм Windows

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

private void InitializeComponent()

{

this.components = new System.ComponentModel.Container();

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(800, 450);

this.Text = "Form1";

}

#endregion

}

}

**Program.cs:**

using CSCS2\_Forms;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ServerApp

{

internal static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

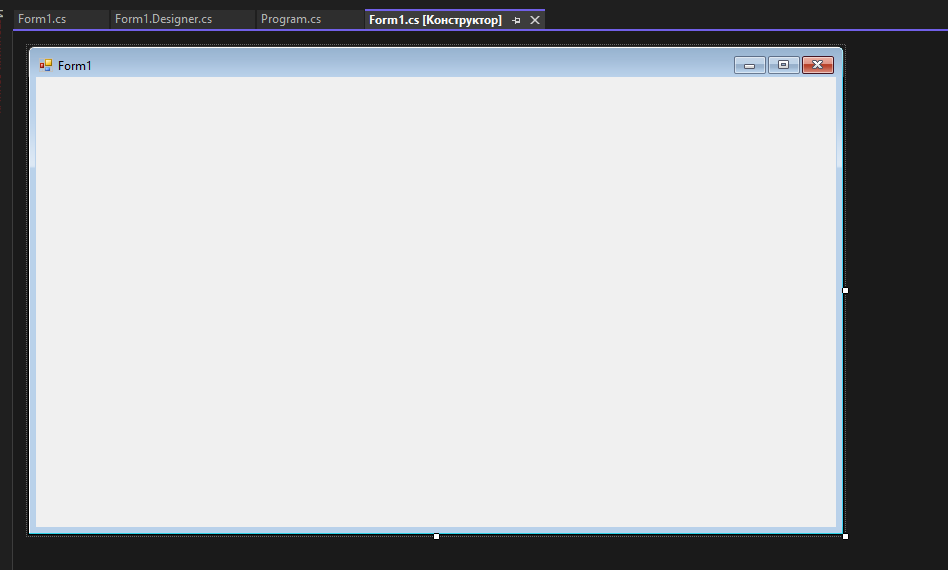
Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}



**Unit Test:**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

namespace CSCS1

{

[TestClass]

public class UnitTest1

{

Commands command = new Commands();

[TestMethod]

public void Command1Test1()

{

/\*\*\*\*\* Encode \*\*\*\*\*/

// Arrange

byte commandNum = 1;

string hexcolor = "59FF";

byte[] resultExpect = { 1, 255, 89 };

// Act

byte[] result = command.ClearDisplayEncode(commandNum, hexcolor);

// Assert

CollectionAssert.AreEqual(resultExpect, result);

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] message = { 1, 68, 236 };

string hexcolorExpect = "EC44";

byte commandExpect = 1;

// Act

command.ClearDisplayDecode(message, out byte commandResult, out string

hexcolorResult);

// Assert

Assert.AreEqual(commandExpect, commandResult);

Assert.AreEqual(hexcolorExpect, hexcolorResult);

}

[TestMethod]

public void Command1Test2()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageEmpty = { };

string hexcolorExpect = "EC44";

byte commandExpect = 1;

// Act

command.ClearDisplayDecode(messageEmpty, out byte commandResultEmpty,

out string hexcolorResultEmpty);

// Assert

Assert.AreEqual(commandExpect, commandResultEmpty);

Assert.AreEqual(hexcolorExpect, hexcolorResultEmpty);

}

[TestMethod]

public void Command1Test3()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageShort = { 1, 68 };

string hexcolorExpect = "EC44";

byte commandExpect = 1;

// Act

command.ClearDisplayDecode(messageShort, out byte commandResultShort,

out string hexcolorResultShort);

// Assert

Assert.AreEqual(commandExpect, commandResultShort);

Assert.AreEqual(hexcolorExpect, hexcolorResultShort);

}

[TestMethod]

public void Command1Test4()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageLong = { 1, 68, 236, 20, 8 };

string hexcolorExpect = "EC44";

byte commandExpect = 1;

// Act

command.ClearDisplayDecode(messageLong, out byte commandResultLong,

out string hexcolorResultLong);

// Assert

Assert.AreEqual(commandExpect, commandResultLong);

Assert.AreEqual(hexcolorExpect, hexcolorResultLong);

}

[TestMethod]

public void Command2Test1()

{

/\*\*\*\*\* Encode \*\*\*\*\*/

// Arrange

byte commandNum = 2;

Int16 x0 = 50;

Int16 y0 = 35;

string hexcolor = "1D6C";

byte[] resultExpect = { 2, 50, 0, 35, 0, 108, 29 };

// Act

byte[] result = command.PixelEncode(commandNum, x0, y0, hexcolor);

// Assert

CollectionAssert.AreEqual(resultExpect, result);

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] message = { 2, 12, 0, 20, 0, 233, 215 };

byte commandExpect = 2;

Int16 x0Expect = 12;

Int16 y0Expect = 20;

string hexcolorExpect = "D7E9";

// Act

command.PixelDecode(message, out byte commandResult, out Int16

x0Result, out Int16 y0Result, out string hexcolorResult);

// Assert

Assert.AreEqual(commandExpect, commandResult); Assert.AreEqual(x0Expect, x0Result);

Assert.AreEqual(y0Expect, y0Result);

Assert.AreEqual(hexcolorExpect, hexcolorResult);

}

[TestMethod]

public void Command2Test2()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageEmpty = { };

byte commandExpect = 2;

Int16 x0Expect = 12;

Int16 y0Expect = 20;

string hexcolorExpect = "D7E9";

// Act

command.PixelDecode(messageEmpty, out byte commandResultEmpty, out

Int16 x0ResultEmpty, out Int16 y0ResultEmpty, out string hexcolorResultEmpty);

// Assert

Assert.AreEqual(commandExpect, commandResultEmpty);

Assert.AreEqual(x0Expect, x0ResultEmpty);

Assert.AreEqual(y0Expect, y0ResultEmpty);

Assert.AreEqual(hexcolorExpect, hexcolorResultEmpty);

}

[TestMethod]

public void Command2Test3()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageShort = { 2, 12, 0 };

byte commandExpect = 2;

Int16 x0Expect = 12;

Int16 y0Expect = 20;

string hexcolorExpect = "D7E9";

// Act

command.PixelDecode(messageShort, out byte commandResultShort, out

Int16 x0ResultShort, out Int16 y0ResultShort, out string hexcolorResultShort);

// Assert

Assert.AreEqual(commandExpect, commandResultShort);

Assert.AreEqual(x0Expect, x0ResultShort);

Assert.AreEqual(y0Expect, y0ResultShort);

Assert.AreEqual(hexcolorExpect, hexcolorResultShort);

}

[TestMethod]

public void Command2Test4()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageLong = { 2, 12, 0, 20, 0, 233, 215, 24, 45 };

byte commandExpect = 2;

Int16 x0Expect = 12;

Int16 y0Expect = 20;

string hexcolorExpect = "D7E9";

// Act

command.PixelDecode(messageLong, out byte commandResultLong, out Int16

x0ResultLong, out Int16 y0ResultLong, out string hexcolorResultLong);

// Assert

Assert.AreEqual(commandExpect, commandResultLong);

Assert.AreEqual(x0Expect, x0ResultLong); Assert.AreEqual(y0Expect, y0ResultLong);

Assert.AreEqual(hexcolorExpect, hexcolorResultLong);

}

[TestMethod]

public void Command3Test1()

{

/\*\*\*\*\* Encode \*\*\*\*\*/

// Arrange

byte commandNum = 3;

Int16 x0 = 40;

Int16 y0 = 31;

Int16 x1 = 62;

Int16 y1 = 39;

string hexcolor = "2A28";

byte[] resultExpect = { 3, 40, 0, 31, 0, 62, 0, 39, 0, 40, 42 };

// Act

byte[] result = command.FourNumbersEncode(commandNum, x0, y0, x1, y1,

hexcolor);

// Assert

CollectionAssert.AreEqual(resultExpect, result);

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] message = { 3, 42, 0, 55, 0, 34, 0, 75, 0, 232, 40 };

byte commandExpect = 3;

Int16 x0Expect = 42;

Int16 y0Expect = 55;

Int16 x1Expect = 34;

Int16 y1Expect = 75;

string hexcolorExpect = "28E8";

// Act

command.FourNumbersDecode(message, out byte commandResult, out Int16

x0Result, out Int16 y0Result, out Int16 x1Result, out Int16 y1Result, out string

hexcolorResult);

// Assert

Assert.AreEqual(commandExpect, commandResult);

Assert.AreEqual(x0Expect, x0Result);

Assert.AreEqual(y0Expect, y0Result);

Assert.AreEqual(x1Expect, x1Result);

Assert.AreEqual(y1Expect, y1Result);

Assert.AreEqual(hexcolorExpect, hexcolorResult);

}

[TestMethod]

public void Command3Test2()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageEmpty = { };

byte commandExpect = 3;

Int16 x0Expect = 42;

Int16 y0Expect = 55;

Int16 x1Expect = 34;

Int16 y1Expect = 75;

string hexcolorExpect = "28E8";

// Act

command.FourNumbersDecode(messageEmpty, out byte commandResultEmpty,

out Int16 x0ResultEmpty, out Int16 y0ResultEmpty, out Int16 x1ResultEmpty, out

Int16 y1ResultEmpty, out string hexcolorResultEmpty);// Assert

Assert.AreEqual(commandExpect, commandResultEmpty);

Assert.AreEqual(x0Expect, x0ResultEmpty);

Assert.AreEqual(y0Expect, y0ResultEmpty);

Assert.AreEqual(x1Expect, x1ResultEmpty);

Assert.AreEqual(y1Expect, y1ResultEmpty);

Assert.AreEqual(hexcolorExpect, hexcolorResultEmpty);

}

[TestMethod]

public void Command3Test3()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageShort = { 3, 42, 0, 55, 0, 34 };

byte commandExpect = 3;

Int16 x0Expect = 42;

Int16 y0Expect = 55;

Int16 x1Expect = 34;

Int16 y1Expect = 75;

string hexcolorExpect = "28E8";

// Act

command.FourNumbersDecode(messageShort, out byte commandResultShort,

out Int16 x0ResultShort, out Int16 y0ResultShort, out Int16 x1ResultShort, out

Int16 y1ResultShort, out string hexcolorResultShort);

// Assert

Assert.AreEqual(commandExpect, commandResultShort);

Assert.AreEqual(x0Expect, x0ResultShort);

Assert.AreEqual(y0Expect, y0ResultShort);

Assert.AreEqual(x1Expect, x1ResultShort);

Assert.AreEqual(y1Expect, y1ResultShort);

Assert.AreEqual(hexcolorExpect, hexcolorResultShort);

}

[TestMethod]

public void Command3Test4()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageLong = { 3, 42, 0, 55, 0, 34, 0, 75, 0, 232, 40, 84, 22

};

byte commandExpect = 3;

Int16 x0Expect = 42;

Int16 y0Expect = 55;

Int16 x1Expect = 34;

Int16 y1Expect = 75;

string hexcolorExpect = "28E8";

// Act

command.FourNumbersDecode(messageLong, out byte commandResultLong, out

Int16 x0ResultLong, out Int16 y0ResultLong, out Int16 x1ResultLong, out Int16

y1ResultLong, out string hexcolorResultLong);

// Assert

Assert.AreEqual(commandExpect, commandResultLong);

Assert.AreEqual(x0Expect, x0ResultLong);

Assert.AreEqual(y0Expect, y0ResultLong);

Assert.AreEqual(x1Expect, x1ResultLong);

Assert.AreEqual(y1Expect, y1ResultLong);

Assert.AreEqual(hexcolorExpect, hexcolorResultLong);

}

[TestMethod]

public void Command4Test1()

{

/\*\*\*\*\* Encode \*\*\*\*\*/

// Arrange

byte commandNum = 4;

Int16 x0 = 32;

Int16 y0 = 54;

Int16 radius = 10;

string hexcolor = "4240";

byte[] resultExpect = { 4, 32, 0, 54, 0, 10, 0, 64, 66 };

// Act

byte[] result = command.CircleEncode(commandNum, x0, y0, radius,

hexcolor);

// Assert

CollectionAssert.AreEqual(resultExpect, result);

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] message = { 4, 67, 0, 95, 0, 18, 0, 255, 255 };

byte commandExpect = 4;

Int16 x0Expect = 67;

Int16 y0Expect = 95;

Int16 radiusExpect = 18;

string hexcolorExpect = "FFFF";

// Act

command.CircleDecode(message, out byte commandResult, out Int16

x0Result, out Int16 y0Result, out Int16 radiusResult, out string hexcolorResult);

// Assert

Assert.AreEqual(commandExpect, commandResult);

Assert.AreEqual(x0Expect, x0Result);

Assert.AreEqual(y0Expect, y0Result);

Assert.AreEqual(radiusExpect, radiusResult);

Assert.AreEqual(hexcolorExpect, hexcolorResult);

}

[TestMethod]

public void Command4Test2()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageEmpty = { };

byte commandExpect = 4;

Int16 x0Expect = 67;

Int16 y0Expect = 95;

Int16 radiusExpect = 18;

string hexcolorExpect = "FFFF";

// Act

command.CircleDecode(messageEmpty, out byte commandResultEmpty, out

Int16 x0ResultEmpty, out Int16 y0ResultEmpty, out Int16 radiusResultEmpty, out

string hexcolorResultEmpty);

// Assert

Assert.AreEqual(commandExpect, commandResultEmpty);

Assert.AreEqual(x0Expect, x0ResultEmpty);

Assert.AreEqual(y0Expect, y0ResultEmpty);

Assert.AreEqual(radiusExpect, radiusResultEmpty);

Assert.AreEqual(hexcolorExpect, hexcolorResultEmpty);

}

[TestMethod]

public void Command4Test3()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageShort = { 4, 67, 0, 95, 0, 18 };

byte commandExpect = 4;

Int16 x0Expect = 67;

Int16 y0Expect = 95;

Int16 radiusExpect = 18;

string hexcolorExpect = "FFFF";

// Act

command.CircleDecode(messageShort, out byte commandResultShort, out

Int16 x0ResultShort, out Int16 y0ResultShort, out Int16 radiusResultShort, out

string hexcolorResultShort);

// Assert

Assert.AreEqual(commandExpect, commandResultShort);

Assert.AreEqual(x0Expect, x0ResultShort);

Assert.AreEqual(y0Expect, y0ResultShort);

Assert.AreEqual(radiusExpect, radiusResultShort);

Assert.AreEqual(hexcolorExpect, hexcolorResultShort);

}

[TestMethod]

public void Command4Test4()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageLong = { 4, 67, 0, 95, 0, 18, 0, 255, 255, 95, 0 };

byte commandExpect = 4;

Int16 x0Expect = 67;

Int16 y0Expect = 95;

Int16 radiusExpect = 18;

string hexcolorExpect = "FFFF";

// Act

command.CircleDecode(messageLong, out byte commandResultLong, out

Int16 x0ResultLong, out Int16 y0ResultLong, out Int16 radiusResultLong, out string

hexcolorResultLong);

// Assert

Assert.AreEqual(commandExpect, commandResultLong);

Assert.AreEqual(x0Expect, x0ResultLong);

Assert.AreEqual(y0Expect, y0ResultLong);

Assert.AreEqual(radiusExpect, radiusResultLong);

Assert.AreEqual(hexcolorExpect, hexcolorResultLong);

}

[TestMethod]

public void Command5Test1()

{

/\*\*\*\*\* Encode \*\*\*\*\*/

// Arrange

byte commandNum = 5;

Int16 x0 = 3;

Int16 y0 = 6;

Int16 x1 = 2;

Int16 y1 = 11;

Int16 radius = 10;

string hexcolor = "34E7";

byte[] resultExpect = { 5, 3, 0, 6, 0, 2, 0, 11, 0, 10, 0, 231, 52 };

// Act

byte[] result = command.RoundedRectEncode(commandNum, x0, y0, x1, y1,

radius, hexcolor);// Assert

CollectionAssert.AreEqual(resultExpect, result);

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] message = { 5, 44, 0, 12, 0, 34, 0, 56, 0, 18, 0, 225, 154 };

byte commandExpect = 5;

Int16 x0Expect = 44;

Int16 y0Expect = 12;

Int16 x1Expect = 34;

Int16 y1Expect = 56;

Int16 radiusExpect = 18;

string hexcolorExpect = "9AE1";

// Act

command.RoundedRectDecode(message, out byte commandResult, out Int16

x0Result, out Int16 y0Result, out Int16 x1Result, out Int16 y1Result, out Int16

radiusResult, out string hexcolorResult);

// Assert

Assert.AreEqual(commandExpect, commandResult);

Assert.AreEqual(x0Expect, x0Result);

Assert.AreEqual(y0Expect, y0Result);

Assert.AreEqual(x1Expect, x1Result);

Assert.AreEqual(y1Expect, y1Result);

Assert.AreEqual(radiusExpect, radiusResult);

Assert.AreEqual(hexcolorExpect, hexcolorResult);

}

[TestMethod]

public void Command5Test2()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageEmpty = { };

byte commandExpect = 5;

Int16 x0Expect = 44;

Int16 y0Expect = 12;

Int16 x1Expect = 34;

Int16 y1Expect = 56;

Int16 radiusExpect = 18;

string hexcolorExpect = "9AE1";

// Act

command.RoundedRectDecode(messageEmpty, out byte commandResultEmpty,

out Int16 x0ResultEmpty, out Int16 y0ResultEmpty, out Int16 x1ResultEmpty, out

Int16 y1ResultEmpty, out Int16 radiusResultEmpty, out string hexcolorResultEmpty);

// Assert

Assert.AreEqual(commandExpect, commandResultEmpty);

Assert.AreEqual(x0Expect, x0ResultEmpty);

Assert.AreEqual(y0Expect, y0ResultEmpty);

Assert.AreEqual(x1Expect, x1ResultEmpty);

Assert.AreEqual(y1Expect, y1ResultEmpty);

Assert.AreEqual(radiusExpect, radiusResultEmpty);

Assert.AreEqual(hexcolorExpect, hexcolorResultEmpty);

}

[TestMethod]

public void Command5Test3()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageShort = { 5, 44, 0, 12, 0, 34, 0 };

byte commandExpect = 5; Int16 x0Expect = 44;

Int16 y0Expect = 12;

Int16 x1Expect = 34;

Int16 y1Expect = 56;

Int16 radiusExpect = 18;

string hexcolorExpect = "9AE1";

// Act

command.RoundedRectDecode(messageShort, out byte commandResultShort,

out Int16 x0ResultShort, out Int16 y0ResultShort, out Int16 x1ResultShort, out

Int16 y1ResultShort, out Int16 radiusResultShort, out string hexcolorResultShort);

// Assert

Assert.AreEqual(commandExpect, commandResultShort);

Assert.AreEqual(x0Expect, x0ResultShort);

Assert.AreEqual(y0Expect, y0ResultShort);

Assert.AreEqual(x1Expect, x1ResultShort);

Assert.AreEqual(y1Expect, y1ResultShort);

Assert.AreEqual(radiusExpect, radiusResultShort);

Assert.AreEqual(hexcolorExpect, hexcolorResultShort);

}

[TestMethod]

public void Command5Test4()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageLong = { 5, 44, 0, 12, 0, 34, 0, 56, 0, 18, 0, 225, 154,

19, 57 };

byte commandExpect = 5;

Int16 x0Expect = 44;

Int16 y0Expect = 12;

Int16 x1Expect = 34;

Int16 y1Expect = 56;

Int16 radiusExpect = 18;

string hexcolorExpect = "9AE1";

// Act

command.RoundedRectDecode(messageLong, out byte commandResultLong, out

Int16 x0ResultLong, out Int16 y0ResultLong, out Int16 x1ResultLong, out Int16

y1ResultLong, out Int16 radiusResultLong, out string hexcolorResultLong);

// Assert

Assert.AreEqual(commandExpect, commandResultLong);

Assert.AreEqual(x0Expect, x0ResultLong);

Assert.AreEqual(y0Expect, y0ResultLong);

Assert.AreEqual(x1Expect, x1ResultLong);

Assert.AreEqual(y1Expect, y1ResultLong);

Assert.AreEqual(radiusExpect, radiusResultLong);

Assert.AreEqual(hexcolorExpect, hexcolorResultLong);

}

[TestMethod]

public void Command6Test1()

{

/\*\*\*\*\* Encode \*\*\*\*\*/

// Arrange

byte commandNum = 6;

Int16 x0 = 43;

Int16 y0 = 12;

string hexcolor = "04E0";

Int16 x1 = 14;

string text = "Hello, World!";

Int16 y1 = Convert.ToInt16(text.Length); byte[] resultExpect = { 6, 43, 0, 12, 0, 224, 4, 14, 0, 13, 0, 72, 0,

101, 0, 108, 0, 108, 0, 111, 0, 44, 0, 32, 0, 87, 0, 111, 0, 114, 0, 108, 0, 100,

0, 33, 0 };

// Act

byte[] result = command.TextEncode(commandNum, x0, y0, hexcolor, x1,

y1, text);

// Assert

CollectionAssert.AreEqual(resultExpect, result);

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] message = { 6, 21, 0, 45, 0, 240, 153, 12, 0, 13, 0, 71, 0,

111, 0, 111, 0, 100, 0, 32, 0, 77, 0, 111, 0, 114, 0, 110, 0, 105, 0, 110, 0, 103,

0, 33, 0 };

byte commandExpect = 6;

Int16 x0Expect = 21;

Int16 y0Expect = 45;

string hexcolorExpect = "99F0";

Int16 x1Expect = 12;

Int16 y1Expect = 13;

string textExpect = "Good Morning!";

// Act

command.TextDecode(message, out byte commandResult, out Int16

x0Result, out Int16 y0Result, out string hexcolorResult, out Int16 x1Result, out

Int16 y1Result, out string textResult);

// Assert

Assert.AreEqual(commandExpect, commandResult);

Assert.AreEqual(x0Expect, x0Result);

Assert.AreEqual(y0Expect, y0Result);

Assert.AreEqual(hexcolorExpect, hexcolorResult);

Assert.AreEqual(x1Expect, x1Result);

Assert.AreEqual(y1Expect, y1Result);

Assert.AreEqual(textExpect, textResult);

}

[TestMethod]

public void Command6Test2()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageEmpty = { };

byte commandExpect = 6;

Int16 x0Expect = 21;

Int16 y0Expect = 45;

string hexcolorExpect = "99F0";

Int16 x1Expect = 12;

Int16 y1Expect = 13;

string textExpect = "Good Morning!";

// Act

command.TextDecode(messageEmpty, out byte commandResultEmpty, out

Int16 x0ResultEmpty, out Int16 y0ResultEmpty, out string hexcolorResultEmpty, out

Int16 x1ResultEmpty, out Int16 y1ResultEmpty, out string textResultEmpty);

// Assert

Assert.AreEqual(commandExpect, commandResultEmpty);

Assert.AreEqual(x0Expect, x0ResultEmpty);

Assert.AreEqual(y0Expect, y0ResultEmpty);

Assert.AreEqual(hexcolorExpect, hexcolorResultEmpty);

Assert.AreEqual(x1Expect, x1ResultEmpty); Assert.AreEqual(y1Expect, y1ResultEmpty);

Assert.AreEqual(textExpect, textResultEmpty);

}

[TestMethod]

public void Command6Test3()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageShort = { 6, 21, 0, 45, 0, 240, 153, 12, 0, 13, 0, 71,

0, 111, 0, 111, 0, 100, 0, 32, 0, 77, 0, 111, 0, 114, 0, 110, 0, 105 };

byte commandExpect = 6;

Int16 x0Expect = 21;

Int16 y0Expect = 45;

string hexcolorExpect = "99F0";

Int16 x1Expect = 12;

Int16 y1Expect = 13;

string textExpect = "Good Morning!";

// Act

command.TextDecode(messageShort, out byte commandResultShort, out

Int16 x0ResultShort, out Int16 y0ResultShort, out string hexcolorResultShort, out

Int16 x1ResultShort, out Int16 y1ResultShort, out string textResultShort);

// Assert

Assert.AreEqual(commandExpect, commandResultShort);

Assert.AreEqual(x0Expect, x0ResultShort);

Assert.AreEqual(y0Expect, y0ResultShort);

Assert.AreEqual(hexcolorExpect, hexcolorResultShort);

Assert.AreEqual(x1Expect, x1ResultShort);

Assert.AreEqual(y1Expect, y1ResultShort);

Assert.AreEqual(textExpect, textResultShort);

}

[TestMethod]

public void Command6Test4()

{

/\*\*\*\*\* Decode \*\*\*\*\*/

// Arrange

byte[] messageLong = { 6, 21, 0, 45, 0, 240, 153, 12, 0, 13, 0, 71, 0,

111, 0, 111, 0, 100, 0, 32, 0, 77, 0, 111, 0, 114, 0, 110, 0, 105, 0, 110, 0, 103,

0, 33, 0, 110, 0, 105, 0, 110, 0, 103, 0, 33, 0 };

byte commandExpect = 6;

Int16 x0Expect = 21;

Int16 y0Expect = 45;

string hexcolorExpect = "99F0";

Int16 x1Expect = 12;

Int16 y1Expect = 13;

string textExpect = "Good Morning!";

// Act

command.TextDecode(messageLong, out byte commandResultLong, out Int16

x0ResultLong, out Int16 y0ResultLong, out string hexcolorResultLong, out Int16

x1ResultLong, out Int16 y1ResultLong, out string textResultLong);

// Assert

Assert.AreEqual(commandExpect, commandResultLong);

Assert.AreEqual(x0Expect, x0ResultLong);

Assert.AreEqual(y0Expect, y0ResultLong);

Assert.AreEqual(hexcolorExpect, hexcolorResultLong);

Assert.AreEqual(x1Expect, x1ResultLong);

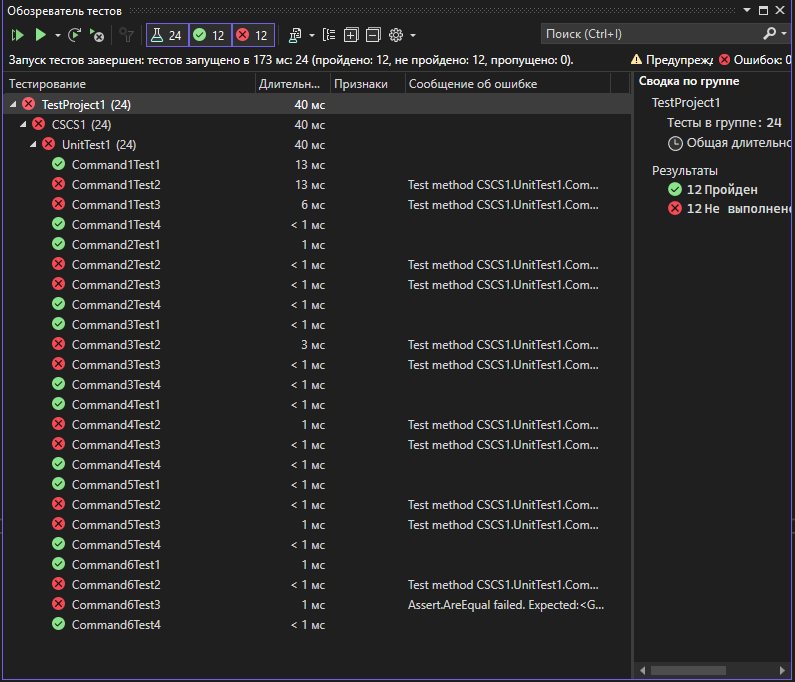
Assert.AreEqual(y1Expect, y1ResultLong);

Assert.AreEqual(textExpect, textResultLong);

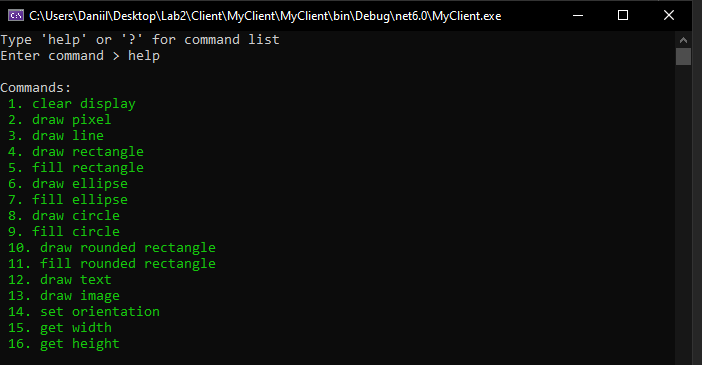
}

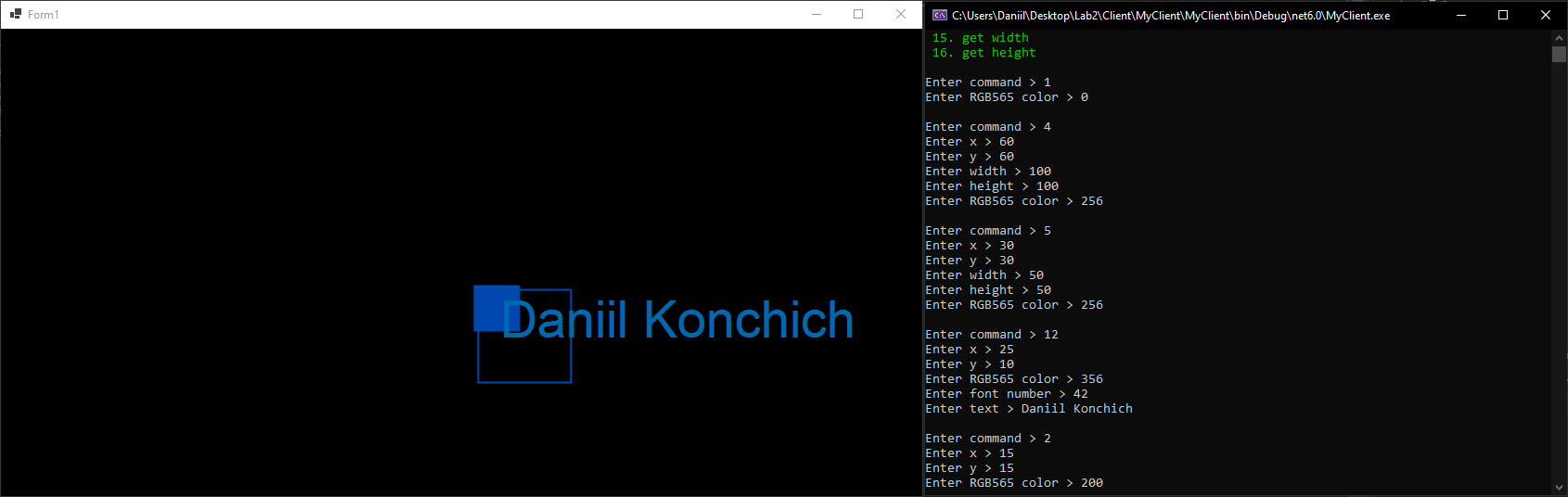
}

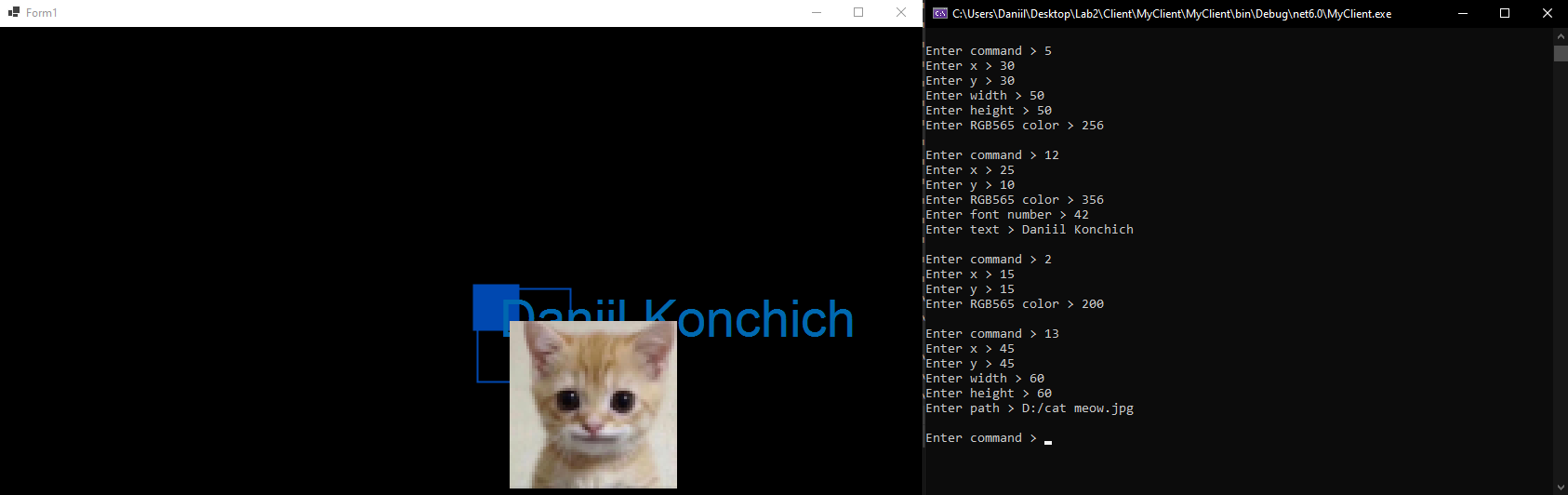
}



Перевіримо усі команди, що реалізували в лр1.







**Висновок:** розробив програму для емуляції дисплейного модуля.