**Лабораторна робота №1**

Виконав студент групи КН-21-1

Ільїчов Д. С.

**Мета роботи**: Розробити програму для реалізації протоколу управління дисплейним модулем.

**Етапи виконання лабораторної роботи:**

1. Розробити специфікацію для протоколу управління дисплейним модулем, що підтримує команди описані нижче.

2. Розробити алгоритм і виконати реалізацію модуля для розбору масиву байтів в команди згідно специфікації п.1.

3. Реалізувати тестування модуля розбору команд із масиву байтів. Для реалізації тестів використати бібліотеку UnitTest. Тести повинні покривати як коректні так і помилкові варіанти команд протоколу.

4. Реалізувати програму сервера, що приймає команди по протоколу UDP. Сервер повинен прийняти пакет (масив байтів), провести розбір пакета на команди. На підставі результату розбору пакету програма сервера повинна вивести в консоль ім'я прийнятої команди і параметри або повідомлення про помилку. Працездатність програми сервера повинна зберігатися як при отриманні правильної так і при отриманні не коректної команди.

Команди для реалізації.

Мінімальний набір:

* clear display: color
* draw pixel: x0, y0, color
* draw line: x0, y0, x1, y1, color
* draw rectangle: x0, y0, w, h, color
* fill rectangle: x0, y0, w, h, color
* draw ellipse: x0, y0, radius\_x, radius\_y, color
* fill ellipse: x0, y0, radius\_x, radius\_y, color

Розширений набір:

* draw circle: x0, x0, radius, color
* fill circle: x0, y0, radius, color
* draw rounded rectangle: x0, y0, w, h, radius, color
* fill rounded rectangle: x0, y0, w, h, radius, color
* draw text: x0, y0, color, font\_number, length, text
* draw image: x0, y0, w, h, data //(data\_length = w \* h \* sizeof(color))

*Хід роботи*

**Лістинг коду:**

Код для роботи серверу:

Program.cs

using System;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Drawing;

using System.Runtime.InteropServices;

namespace CSCS1

{

class ReceivingProgram

{

static void Main(string[] args)

{

ReceiveMessage();

}

private static void ReceiveMessage()

{

IntPtr handle = GetStdHandle(-11);

GetConsoleMode(handle, out int mode);

SetConsoleMode(handle, mode | 0x4);

int port = 1001;

Commands commands = new 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;

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}; ");

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}; ");

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}; ");

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}; ");

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}; ");

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}; ");

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}; ");

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}; ");

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}; ");

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}; ");

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}; ");

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}; ");

break;

case 13:

Color[,] colors;

commands.ImageDecode(data, out command, out x0, out y0, out x1, out y1, out colors);

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

for (int col = 0; col < y1; col++)

{

for (int row = 0; row < x1; row++)

{

Console.Write("\x1b[38;2;" + colors[row, col].R + ";" + colors[row, col].G + ";" + colors[row, col].B + "m");

Console.Write("██");

}

Console.WriteLine("");

}

Console.WriteLine("");

Console.ResetColor();

break;

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

receiver.Close();

}

}

[DllImport("kernel32.dll", SetLastError = true)]

public static extern bool SetConsoleMode(IntPtr hConsoleHandle, int mode);

[DllImport("kernel32.dll", SetLastError = true)]

public static extern bool GetConsoleMode(IntPtr handle, out int mode);

[DllImport("kernel32.dll", SetLastError = true)]

public static extern IntPtr GetStdHandle(int handle);

}

}

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;

}

}

}

Код для роботи клієнту:

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 = 1001;

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

{

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"://clear display

commandbyte[0] = 1;

hexcolor = ReadHexColor();

result = commands.ClearDisplayEncode(commandbyte[0], hexcolor);

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

break;

case "2"://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"://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"://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"://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"://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"://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"://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"://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"://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"://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"://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"://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;

default:

Console.ForegroundColor = ConsoleColor.Yellow; 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.Yellow;

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.Yellow;

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

}

else { break; }

}

else { break; }

}

catch

{

Console.ForegroundColor = ConsoleColor.Yellow;

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.Yellow;

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;

}

}

}

Код для тестування:

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);

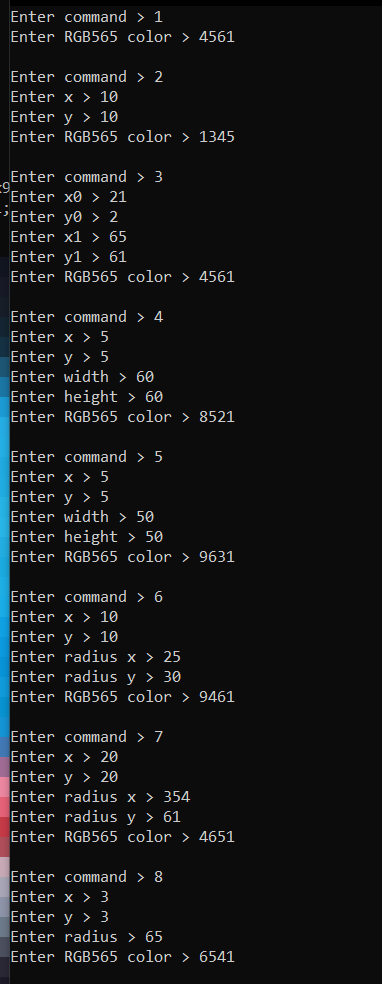
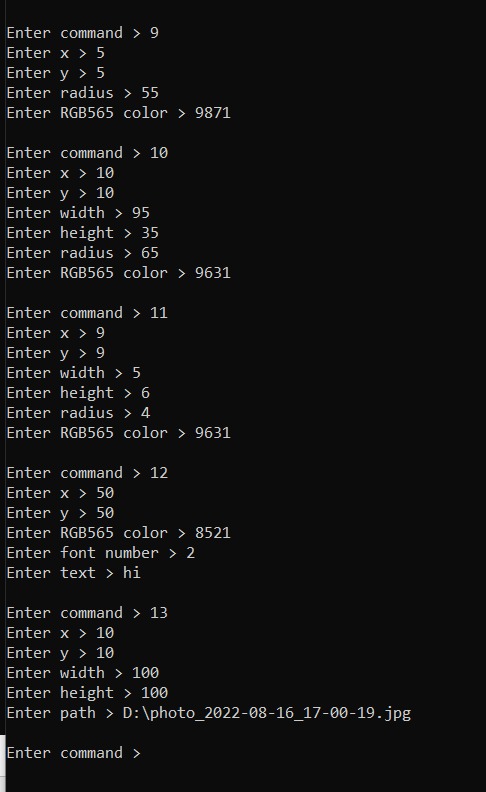
}

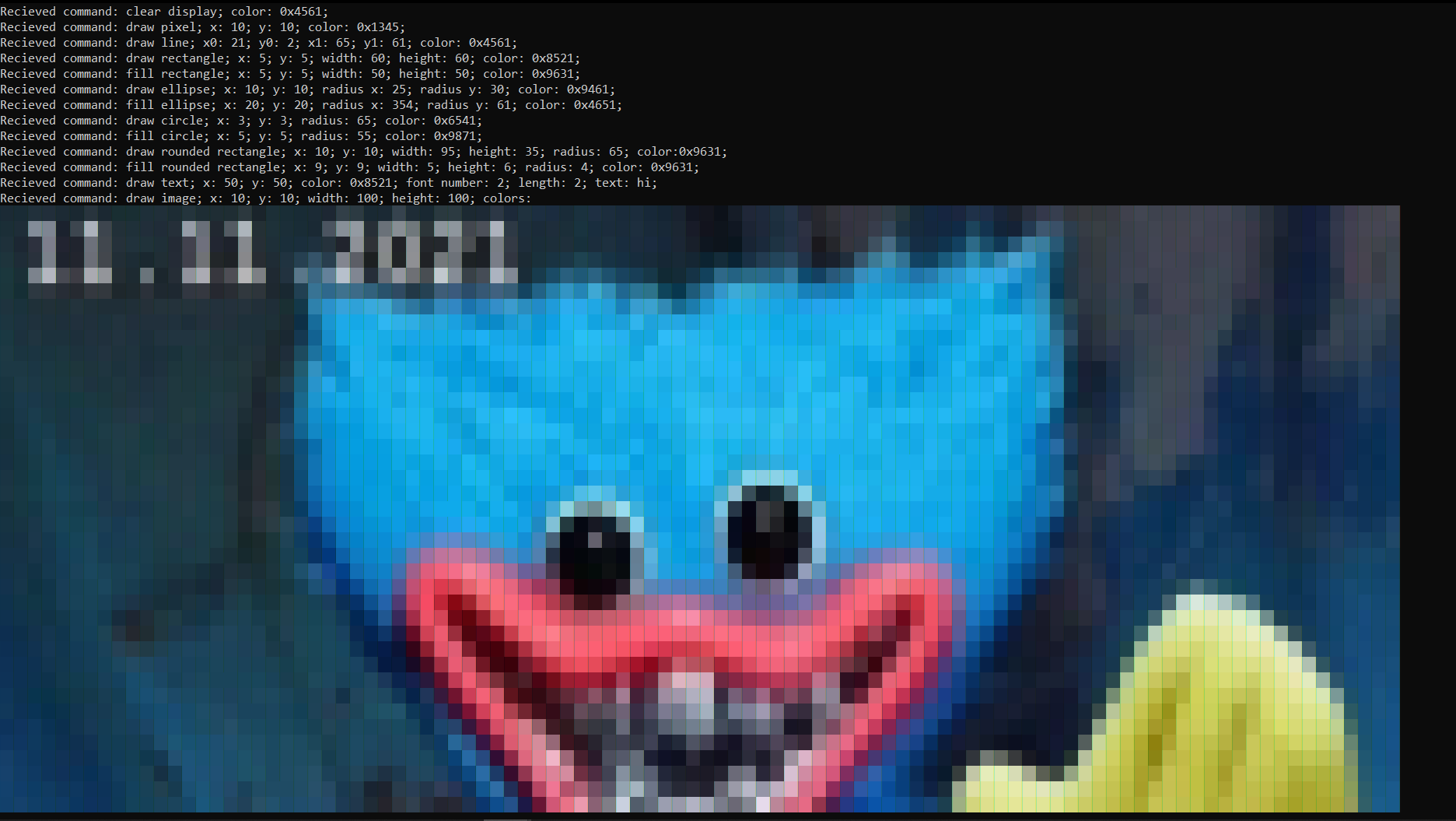
}

}

Файли проекту знаходяться за посиланням <https://github.com/caniplaysolo/ksks1>

Результат виконання:

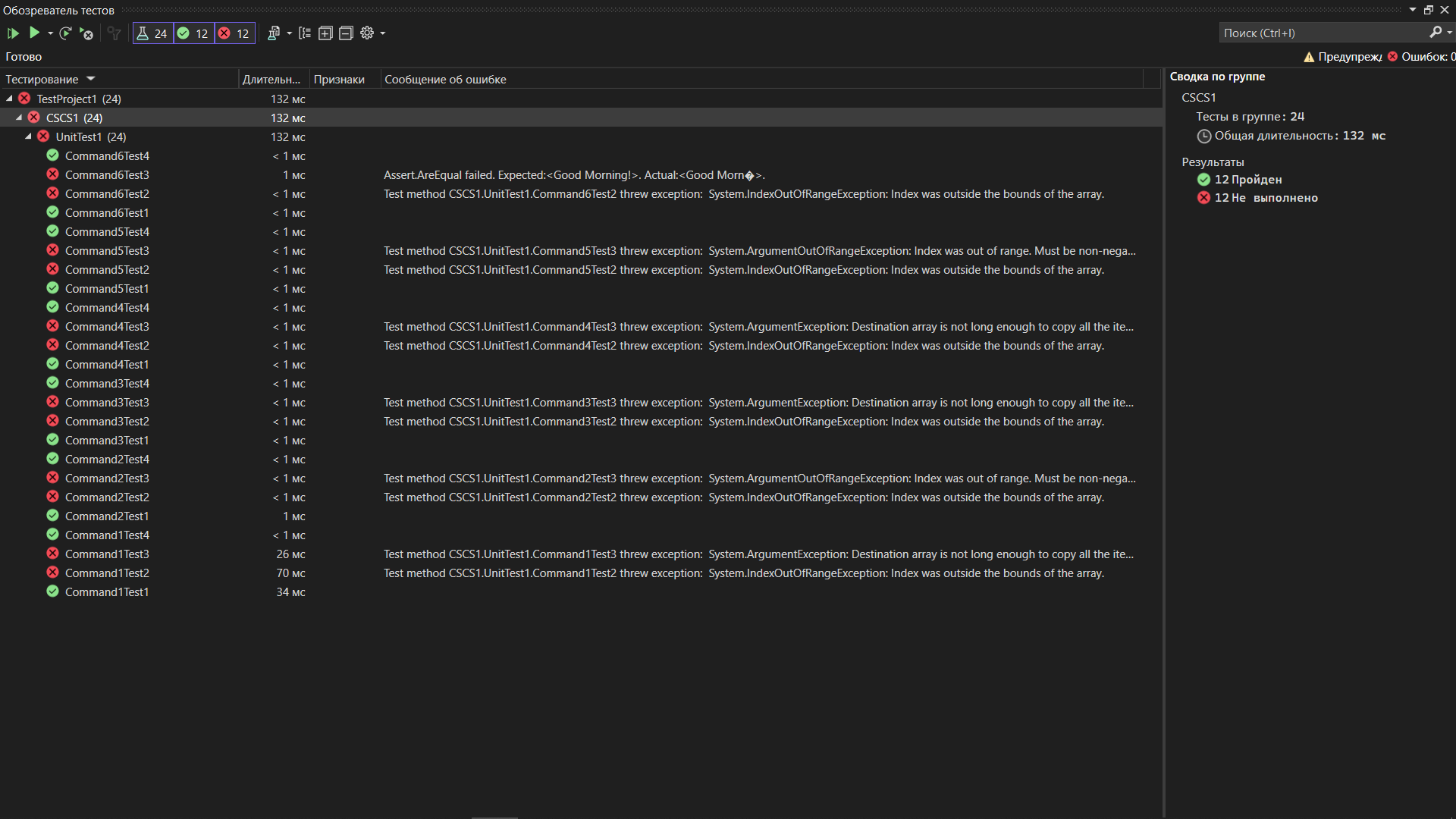




Оригінал фотографії:



Провів тестування:



**Висновок**: розробив програму для реалізації протоколу управління дисплейним модулем.