Карпенко О. КН20-1

**ЛАБОРАТОРНА РОБОТА 1**

Мета: 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))

Просунутий набір:

set orientation: orientation //(0=0, 1=90, 2=180, 3=270)

get width:

get height:

Хід Роботи:

Специфікація:

Змінні:

X0 – Int16, або short займає два байта(X0H,X0L), приймає цифру з [-32767; 32767];

X1 – Int16, або short займає два байта(X1H,X1L), приймає цифру з [-32767; 32767];

Y0 – Int16, або short займає два байта(Y0H,Y0L), приймає цифру з [-32767; 32767];

Y1 – Int16, або short займає два байта(Y1H,Y1L), приймає цифру з [-32767; 32767];

W – Int16, або short займає два байта(WH,WL), приймає цифру з (0; 32767];

H – Int16, або short займає два байта(HH,HL), приймає цифру з (0; 32767];

radius\_x – Int16, або short бере два байта(radius\_x1H, radius\_x1L), приймає цифру з (0; 32767];

radius\_y – Int16, або short бере два байта(radius\_y1H, radius\_y1L), приймає цифру з (0; 32767];

color – RGB888,перший байт відповідає за красний колір приймає цифру з радіусу[0;255], другий відповідає за зелений приймає цифру з радіусу[0;255], третій відповідає за синій приймає цифру з радіусу[0;255]

text – string в Unicode;

orientation – byte та приймає цифри [0;3]

Команди та розшифровка повідомлення:

clear display: color

Повідомлення: 1, color;

draw pixel: x0, y0, color

Повідомлення: 2, X0H, X0L, Y0H, Y0L, color;

draw line: x0, y0, x1, y1, color

Повідомлення: 3, X0H, X0L, Y0H, Y0L, X1H,X1L, Y1H,Y1L, color;

draw rectangle: x0, y0, w, h, color

Повідомлення: 4, X0H, X0L, Y0H, Y0L, WH,WL, HH,HL, color;

fill rectangle: x0, y0, w, h, color

Повідомлення: 5, X0H, X0L, Y0H, Y0L, WH,WL, HH,HL, color;

draw ellipse: x0, y0, radius\_x, radius\_y, color

Повідомлення: 6, X0H, X0L, Y0H, Y0L, radius\_x1H, radius\_x1L, radius\_y1H, radius\_y1L, color;

fill ellipse: x0, y0, radius\_x, radius\_y, color

Повідомлення: 7, X0H, X0L, Y0H, Y0L, radius\_x1H, radius\_x1L, radius\_y1H, radius\_y1L, color;

draw circle: x0, x0, radius, color

Повідомлення: 8, X0H, X0L, Y0H, Y0L, radius\_x1H, radius\_x1L, color;

fill circle: x0, y0, radius, color

Повідомлення: 9, X0H, X0L, Y0H, Y0L, radius\_x1H, radius\_x1L, color;

draw rounded rectangle: x0, y0, w, h, radius, color

Повідомлення: 10, X0H, X0L, Y0H, Y0L, WH,WL, HH,HL, radius\_x1H, radius\_x1L, radius\_y1H, radius\_y1L, color;

fill rounded rectangle: x0, y0, w, h, radius, color

Повідомлення: 11, X0H, X0L, Y0H, Y0L, WH,WL, HH,HL, radius\_x1H, radius\_x1L, radius\_y1H, radius\_y1L, color;

draw text: x0, y0, color, font\_number, length, text

Повідомлення: 12, X0H, X0L, Y0H, Y0L, color, font\_numberH, font\_numberL, lengthH, lengthL , text[0]H, text[0]L… text[length-1]H, text[length-1]L;

draw image: x0, y0, w, h, data //(data\_length = w \* h \* sizeof(color))

Повідомлення: 13, X0H, X0L, Y0H, Y0L, WH,WL, HH,HL,;

get width

Повідомлення до сервера: 15

Повідомлення до клієнта: WH,WL

get height

Повідомлення до сервера: 16

Повідомлення до клієнта: WH,WL

2.

Кодування команд:

//Command Methods

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

{

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

sendBytesList = new List<byte>();

sendBytesList.Add(command);

sendBytesList.AddRange(RGB);

output = sendBytesList.ToArray();

return output;

}

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

{

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

sendBytesList = new List<byte>();

sendBytesList.Add(command);

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

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

sendBytesList.AddRange(RGB);

output = sendBytesList.ToArray();

return output;

}

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

{

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

sendBytesList = new List<byte>();

sendBytesList.Add(command);

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

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

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

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

sendBytesList.AddRange(RGB);

output = sendBytesList.ToArray();

return output;

}

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

{

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

sendBytesList = new List<byte>();

sendBytesList.Add(command);

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

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

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

sendBytesList.AddRange(RGB);

output = sendBytesList.ToArray();

return output;

}

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

{

List<byte> sendBytesList; byte[] output;

sendBytesList = new List<byte>();

sendBytesList.Add(command);

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

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

sendBytesList.AddRange(RGB);

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

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

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

output = sendBytesList.ToArray();

return output;

}

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

{

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

sendBytesList = new List<byte>();

sendBytesList.Add(command);

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

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

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

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

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

sendBytesList.AddRange(RGB);

output = sendBytesList.ToArray();

return output;

}

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

{

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

map = new Bitmap(path);

sendBytesList = new List<byte>();

sendBytesList.Add(command);

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

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

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

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

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

{

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

{

color = map.GetPixel(i, j);

sendBytesList.Add(color.R);

sendBytesList.Add(color.G);

sendBytesList.Add(color.B);

}

}

output = sendBytesList.ToArray();

return output;

}

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

{

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

sendBytesList = new List<byte>();

sendBytesList.Add(command);

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

output = sendBytesList.ToArray();

return output;

}

Декодування

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

{

int val1place = 1;

RGB = new byte[3];

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

}

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

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

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

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

}

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

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

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

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val3 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val4 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

}

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

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

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

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val3 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

}

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

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

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

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val3 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val4 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val5 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

}

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

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

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

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val1place += 3;

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

val3 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val4 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

transfer = new byte[val4];

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

text = Encoding.ASCII.GetString(transfer);

}

public static void ImageDecoder(byte[] RecievedData, out Int16 val1, out Int16 val2, out Int16 width, out Int16 height, out Color[,] pic)

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

Color color;

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

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

width = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

height = BitConverter.ToInt16(transfer, 0);

val1place += 2;

pic = new Color[width,height];

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

{

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

{

transfer = new byte[3];

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

color = Color.FromArgb(transfer[0], transfer[1], transfer[2]);

pic[i, j] = color;

val1place += 3;

}

}

3. Тест кодування:

using Client;

using Microsoft.VisualStudio.TestPlatform.TestHost;

using System.ComponentModel.Design;

namespace DataTest

{

[TestClass]

public class UnitTest1

{

byte command;

Int16 input1, input2, input3, input4, input5;

string text1, text2;

byte[] expectedArrCorrect, expectedArrIncorrect, resultArr;

[TestMethod]

public void TestClearDisplayCorrect()

{

command = 1;

byte[] RGB = new byte[] { 255, 0, 0 };

expectedArrCorrect = new byte[] { 1, 255, 0, 0 };

resultArr = ClientProgram.ClearDisplay(command, RGB);

CollectionAssert.AreEqual(expectedArrCorrect, resultArr);

}

[TestMethod]

public void TestClearDisplayIncorrect()

{

command = 1;

byte[] RGB = new byte[] { 255, 0, 0 };

expectedArrIncorrect = new byte[] { command, 0, 250, 0, 1 };

resultArr = ClientProgram.ClearDisplay(command, RGB);

CollectionAssert.AreNotEqual(expectedArrIncorrect, resultArr);

}

[TestMethod]

public void Test3VarConverterCorrect()

{

command = 2;

input1 = 5;

input2 = 10;

byte[] RGB = new byte[] { 255, 255, 255 };

expectedArrCorrect = new byte[] { command, 5, 0, 10, 0, 255, 255, 255 };

resultArr = ClientProgram.ThreeVarsConverter(command, input1, input2, RGB);

CollectionAssert.AreEqual(expectedArrCorrect, resultArr);

}

[TestMethod]

public void Test3VarConverterIncorrect()

{

command = 2;

input1 = 5;

input2 = 10;

byte[] RGB = new byte[] { 255, 255, 255 };

expectedArrIncorrect = new byte[] { command, 5, 0, 10, 0, 255, 255, 105 };

resultArr = ClientProgram.ThreeVarsConverter(command, input1, input2, RGB);

CollectionAssert.AreNotEqual(expectedArrIncorrect, resultArr);

}

[TestMethod]

public void Test5VarConverterCorrect()

{

command = 2;

input1 = 5;

input2 = 10;

input3 = 25;

input4 = 13;

byte[] RGB = new byte[] { 255, 0, 0 };

expectedArrCorrect = new byte[] { command, 5, 0, 10, 0, 25, 0, 13, 0, 255, 0, 0 };

resultArr = ClientProgram.FiveVarsConverter(command, input1, input2, input3, input4, RGB);

CollectionAssert.AreEqual(expectedArrCorrect, resultArr);

}

[TestMethod]

public void Test5VarConverterIncorrect()

{

command = 2;

input1 = 5;

input2 = 10;

input3 = 25;

input4 = 13;

byte[] RGB = new byte[] { 255, 0, 0 };

expectedArrIncorrect = new byte[] { command, 5, 0, 10, 0, 25, 15, 13, 0, 255, 0, 1 };

resultArr = ClientProgram.FiveVarsConverter(command, input1, input2, input3, input4, RGB);

CollectionAssert.AreNotEqual(expectedArrIncorrect, resultArr);

}

[TestMethod]

public void TestCircleConverterCorrect()

{

command = 2;

input1 = 5;

input2 = 10;

input3 = 25;

byte[] RGB = new byte[] { 0, 0, 0 };

expectedArrIncorrect = new byte[] { command, 5, 0, 10, 0, 25, 0, 0, 0, 0 };

resultArr = ClientProgram.CircleConverter(command, input1, input2, input3, RGB);

CollectionAssert.AreEqual(expectedArrIncorrect, resultArr);

}

[TestMethod]

public void TestCircleConverterIncorrect()

{

command = 2;

input1 = 5;

input2 = 10;

input3 = 25;

byte[] RGB = new byte[] { 0, 0, 0 };

expectedArrIncorrect = new byte[] { command, 5, 0, 10, 0, 25, 15, 0, 0, 1 };

resultArr = ClientProgram.CircleConverter(command, input1, input2, input3, RGB);

CollectionAssert.AreNotEqual(expectedArrIncorrect, resultArr);

}

[TestMethod]

public void TestTextConverterCorrect()

{

command = 2;

input1 = 5;

input2 = 10;

byte[] RGB = new byte[] { 0, 0, 0 };

input3 = 25;

text2 = "Hello";

expectedArrCorrect = new byte[] { 2, 5, 0, 10, 0, 0, 0, 0, 25, 0, 5, 0, 72, 101, 108, 108, 111 };

resultArr = ClientProgram.TextConverter(command, input1, input2, input3, text2, RGB);

CollectionAssert.AreEqual(expectedArrCorrect, resultArr);

}

[TestMethod]

public void TestTextConverterIncorrect()

{

command = 2;

input1 = 5;

input2 = 10;

byte[] RGB = new byte[] { 0, 0, 0 };

input3 = 25;

text2 = "Hello";

expectedArrIncorrect = new byte[] { 3, 5, 0, 10, 0, 0, 0, 1, 25, 5, 111, 108, 108, 101, 72 };

resultArr = ClientProgram.TextConverter(command, input1, input2, input3, text2, RGB);

CollectionAssert.AreNotEqual(expectedArrIncorrect, resultArr);

}

[TestMethod]

public void TestRoundedRectangleConverterCorrect()

{

command = 2;

input1 = 5;

input2 = 10;

input3 = 25;

input4 = 133;

input5 = 15;

byte[] RGB = new byte[] { 0, 0, 0 };

expectedArrCorrect = new byte[] { 2, 5, 0, 10, 0, 25, 0, 133, 0, 15, 0, 0, 0, 0 };

resultArr = ClientProgram.RoundedRectangleConverter(command, input1, input2, input3, input4, input5, RGB);

CollectionAssert.AreEqual(expectedArrCorrect, resultArr);

}

[TestMethod]

public void TestRoundedRectangleConverterIncorrect()

{

command = 2;

input1 = 5;

input2 = 10;

input3 = 25;

input4 = 133;

input5 = 15;

byte[] RGB = new byte[] { 0, 0, 0 };

expectedArrIncorrect = new byte[] { 3, 5, 0, 10, 0, 25, 0, 133, 0, 15, 0, 0, 0, 1 };

resultArr = ClientProgram.RoundedRectangleConverter(command, input1, input2, input3, input4, input5, RGB);

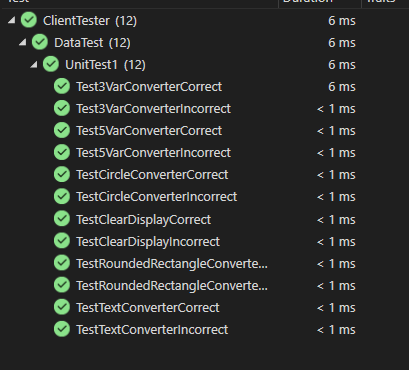
CollectionAssert.AreNotEqual(expectedArrIncorrect, resultArr);

}

}

}

Результат



Тест Декодування:

using Server;

using System.Drawing;

namespace ServerTest

{

[TestClass]

public class UnitTest1

{

byte[] rgbExpected, rgbActual;

byte[] message;

short val1Ex, val2Ex, val3Ex, val4Ex, val5Ex;

short val1Ac, val2Ac, val3Ac, val4Ac, val5Ac;

string textAc, textEx;

[TestMethod]

public void ClearDisplayTestCorrect()

{

message = new byte[] { 1, 255, 255, 255};

rgbExpected = new byte[] { 255, 255, 255};

Server.ServerProgram.ClearDisplay(message, out rgbActual);

CollectionAssert.AreEqual(rgbExpected, rgbActual);

}

[TestMethod]

public void ClearDisplayTestIncorrect()

{

message = new byte[] { 1, 255, 205, 255 };

rgbExpected = new byte[] { 255, 255, 255 };

Server.ServerProgram.ClearDisplay(message, out rgbActual);

CollectionAssert.AreNotEqual(rgbExpected, rgbActual);

}

[TestMethod]

public void ThreeVarTestCorrect()

{

message = new byte[] { 2, 5, 0, 10, 0, 255, 255, 255 };

rgbExpected = new byte[] { 255, 255, 255};

val1Ex = 5;

val2Ex = 10;

Server.ServerProgram.ThreeVarsDecode(message, out val1Ac, out val2Ac, out rgbActual );

CollectionAssert.AreEqual(rgbExpected, rgbActual);

Assert.AreEqual(val1Ex, val1Ac);

Assert.AreEqual(val2Ex, val2Ac);

}

[TestMethod]

public void ThreeVarTestIncorrect()

{

message = new byte[] { 2, 92, 0, 102, 0, 215, 255, 255 };

rgbExpected = new byte[] { 255, 255, 255 };

val1Ex = 5;

val2Ex = 10;

Server.ServerProgram.ThreeVarsDecode(message, out val1Ac, out val2Ac, out rgbActual);

CollectionAssert.AreNotEqual(rgbExpected, rgbActual);

Assert.AreNotEqual(val1Ex, val1Ac);

Assert.AreNotEqual(val2Ex, val2Ac);

}

[TestMethod]

public void FiveVarTestCorrect()

{

message = new byte[] { 3, 5, 0, 10, 0, 25, 0, 13, 0, 255, 0, 0 };

rgbExpected = new byte[] { 255, 0, 0 };

val1Ex = 5;

val2Ex = 10;

val3Ex = 25;

val4Ex = 13;

Server.ServerProgram.FiveVarsDecode(message, out val1Ac, out val2Ac, out val3Ac, out val4Ac, out rgbActual);

CollectionAssert.AreEqual(rgbExpected, rgbActual);

Assert.AreEqual(val1Ex, val1Ac);

Assert.AreEqual(val2Ex, val2Ac);

Assert.AreEqual(val3Ex, val3Ac);

Assert.AreEqual(val4Ex, val4Ac);

}

[TestMethod]

public void FiveVarTestIncorrect()

{

message = new byte[] { 3, 50, 0, 100, 0, 250, 0, 130, 0, 255, 2, 0 };

rgbExpected = new byte[] { 255, 0, 0 };

val1Ex = 5;

val2Ex = 10;

val3Ex = 25;

val4Ex = 13;

Server.ServerProgram.FiveVarsDecode(message, out val1Ac, out val2Ac, out val3Ac, out val4Ac, out rgbActual);

CollectionAssert.AreNotEqual(rgbExpected, rgbActual);

Assert.AreNotEqual(val1Ex, val1Ac);

Assert.AreNotEqual(val2Ex, val2Ac);

Assert.AreNotEqual(val3Ex, val3Ac);

Assert.AreNotEqual(val4Ex, val4Ac);

}

[TestMethod]

public void CircleTestCorrect()

{

message = new byte[] { 5, 5, 0, 10, 0, 25, 0, 0, 0, 0 };

rgbExpected = new byte[] { 0, 0, 0 };

val1Ex = 5;

val2Ex = 10;

val3Ex = 25;

Server.ServerProgram.CircleDecoder(message, out val1Ac, out val2Ac, out val3Ac, out rgbActual);

CollectionAssert.AreEqual(rgbExpected, rgbActual);

Assert.AreEqual(val1Ex, val1Ac);

Assert.AreEqual(val2Ex, val2Ac);

Assert.AreEqual(val3Ex, val3Ac);

}

[TestMethod]

public void CircleTestIncorrect()

{

message = new byte[] { 3, 50, 0, 100, 0, 250, 0, 255, 2, 0 };

rgbExpected = new byte[] { 255, 0, 0 };

val1Ex = 5;

val2Ex = 10;

val3Ex = 25;

Server.ServerProgram.CircleDecoder(message, out val1Ac, out val2Ac, out val3Ac, out rgbActual);

CollectionAssert.AreNotEqual(rgbExpected, rgbActual);

Assert.AreNotEqual(val1Ex, val1Ac);

Assert.AreNotEqual(val2Ex, val2Ac);

Assert.AreNotEqual(val3Ex, val3Ac);

}

[TestMethod]

public void RoundedRectangleCorrect()

{

message = new byte[] { 2, 5, 0, 10, 0, 25, 0, 133, 0, 15, 0, 0, 0, 0 };

rgbExpected = new byte[] { 0, 0, 0 };

val1Ex = 5;

val2Ex = 10;

val3Ex = 25;

val4Ex = 133;

val5Ex = 15;

Server.ServerProgram.RoundedRectangleDecoder(message, out val1Ac, out val2Ac, out val3Ac, out val4Ac, out val5Ac, out rgbActual);

CollectionAssert.AreEqual(rgbExpected, rgbActual);

Assert.AreEqual(val1Ex, val1Ac);

Assert.AreEqual(val2Ex, val2Ac);

Assert.AreEqual(val3Ex, val3Ac);

Assert.AreEqual(val4Ex, val4Ac);

Assert.AreEqual(val5Ex, val5Ac);

}

[TestMethod]

public void RoundedRectangleIncorrect()

{

message = new byte[] { 2, 50, 0, 100, 0, 250, 0, 135, 0, 150, 0, 0, 0, 1 };

rgbExpected = new byte[] { 0, 0, 0 };

val1Ex = 5;

val2Ex = 10;

val3Ex = 25;

val4Ex = 133;

val5Ex = 15;

Server.ServerProgram.RoundedRectangleDecoder(message, out val1Ac, out val2Ac, out val3Ac, out val4Ac, out val5Ac, out rgbActual);

CollectionAssert.AreNotEqual(rgbExpected, rgbActual);

Assert.AreNotEqual(val1Ex, val1Ac);

Assert.AreNotEqual(val2Ex, val2Ac);

Assert.AreNotEqual(val3Ex, val3Ac);

Assert.AreNotEqual(val4Ex, val4Ac);

Assert.AreNotEqual(val5Ex, val5Ac);

}

[TestMethod]

public void TextCorrect()

{

message = new byte[] { 2, 5, 0, 10, 0, 0, 0, 0, 25, 0, 5, 0, 72, 101, 108, 108, 111 };

rgbExpected = new byte[] { 0, 0, 0 };

val1Ex = 5;

val2Ex = 10;

val3Ex = 25;

val4Ex = 5;

textEx = "Hello";

Server.ServerProgram.TextDecoder(message, out val1Ac, out val2Ac, out val3Ac, out val4Ac, out rgbActual, out textAc);

CollectionAssert.AreEqual(rgbExpected, rgbActual);

Assert.AreEqual(val1Ex, val1Ac);

Assert.AreEqual(val2Ex, val2Ac);

Assert.AreEqual(val3Ex, val3Ac);

Assert.AreEqual(val4Ex, val4Ac);

Assert.AreEqual(textEx, textAc);

}

[TestMethod]

public void TextIncorrect()

{

message = new byte[] { 2, 50, 0, 100, 0, 0, 0, 0, 250, 0, 50, 0, 72, 101, 198, 108, 111 };

rgbExpected = new byte[] { 0, 0, 0 };

val1Ex = 5;

val2Ex = 10;

val3Ex = 25;

val4Ex = 5;

textAc = "Hello";

Server.ServerProgram.RoundedRectangleDecoder(message, out val1Ac, out val2Ac, out val3Ac, out val4Ac, out val5Ac, out rgbActual);

CollectionAssert.AreNotEqual(rgbExpected, rgbActual);

Assert.AreNotEqual(val1Ex, val1Ac);

Assert.AreNotEqual(val2Ex, val2Ac);

Assert.AreNotEqual(val3Ex, val3Ac);

Assert.AreNotEqual(val4Ex, val4Ac);

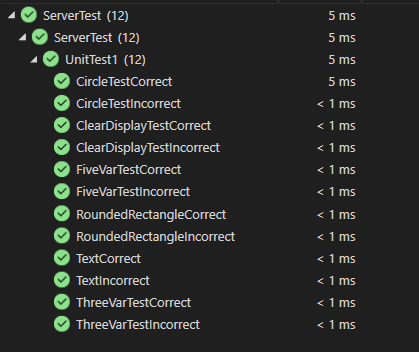
Assert.AreNotEqual(textEx, textAc);

}

}

}

Результат:



4.

Код Сервера.

using System;

using System.Data;

using System.Drawing;

using System.IO;

using System.Net;

using System.Net.Sockets;

using System.Runtime.CompilerServices;

using System.Runtime.ConstrainedExecution;

using System.Text;

using System.Text.RegularExpressions;

using static System.Net.Mime.MediaTypeNames;

namespace Server

{

public class ServerProgram

{

const int SIO\_UDP\_CONNRESET = -1744830452;

const int port = 1984;

public static int SizeCheck(Int16 x, Int16 y,Int16 Width, Int16 Height)

{

if( x > Width || y > Height || x < -Width || y < -Height)

{

return 1;

}

else

{

return 0;

}

}

public static void SetScreenSize(byte[] RecievedData, out Int16 width, out Int16 height)

{

int val1place = 1;

byte[] transfer = new byte[2];

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

width = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

height = BitConverter.ToInt16(transfer, 0);

}

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

{

int val1place = 1;

RGB = new byte[3];

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

}

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

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

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

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

}

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

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

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

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val3 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val4 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

}

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

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

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

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val3 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

}

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

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

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

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val3 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val4 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val5 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

}

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

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

RGB = new byte[3];

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

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val1place += 3;

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

val3 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val4 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

transfer = new byte[val4];

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

text = Encoding.ASCII.GetString(transfer);

}

public static void ImageDecoder(byte[] RecievedData, out Int16 val1, out Int16 val2, out Int16 width, out Int16 height, out Color[,] pic)

{

byte[] transfer;

int val1place = 1;

transfer = new byte[2];

Color color;

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

val1 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

val2 = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

width = BitConverter.ToInt16(transfer, 0);

val1place += 2;

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

height = BitConverter.ToInt16(transfer, 0);

val1place += 2;

pic = new Color[width,height];

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

{

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

{

transfer = new byte[3];

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

color = Color.FromArgb(transfer[0], transfer[1], transfer[2]);

pic[i, j] = color;

val1place += 3;

}

}

}

public static void StartServer()

{

Int16 val1, val2, val3, val4, val5, width = 0, height = 0; byte[] RGB, sendMessage; string text; byte command; Color[,] pic;

Console.WriteLine("Server Begin");

UdpClient server = new UdpClient(port);

server.Client.IOControl((IOControlCode)SIO\_UDP\_CONNRESET,new byte[] { 0, 0, 0, 0 },null);

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

IPEndPoint remoteEP;

try

{

while (true)

{

Console.WriteLine("Waiting for message");

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

command = RecievedData[0];

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

switch (command)

{

case 1:

Console.WriteLine("command:Clear Display");

ClearDisplay(RecievedData, out RGB);

Console.WriteLine($"color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}");

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

sendMessage = Encoding.ASCII.GetBytes(text);

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

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

break;

case 2:

Console.WriteLine("command:Draw Pixel");

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

if (SizeCheck(val1, val2, width, height) == 1)

{

text = "Error:Figure out of bounds";

Console.WriteLine(text);

}

else

{

Console.WriteLine($"Coordinates: x = {val1}, y = {val2}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}");

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

}

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

sendMessage = Encoding.ASCII.GetBytes(text);

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

break;

case 3:

Console.WriteLine("command:Draw Line");

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

if (SizeCheck(val1, val2, width, height) == 1 || SizeCheck(val3, val4, width, height) == 1)

{

text = "Error:Figure out of bounds";

Console.WriteLine(text);

}

else

{

Console.WriteLine($"Coordinates: x1 = {val1}, y1 = {val2}, x2 = {val3}, y2 = {val4}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}");

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

}

sendMessage = Encoding.ASCII.GetBytes(text);

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

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

break;

case 4:

Console.WriteLine("command:Draw Rectangle");

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

if (SizeCheck(val1, val2, width, height) == 1)

{

text = "Error:Figure out of bounds";

Console.WriteLine(text);

}

else

{

Console.WriteLine($"Coordinates: x1 = {val1}, y1 = {val2}, width = {val3}, height = {val4}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}");

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

}

sendMessage = Encoding.ASCII.GetBytes(text);

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

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

break;

case 5:

Console.WriteLine("command:Fill Rectangle");

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

if (SizeCheck(val1, val2, width, height) == 1)

{

text = "Error:Figure out of bounds";

Console.WriteLine(text);

}

else

{

Console.WriteLine($"Coordinates: x1 = {val1}, y1 = {val2}, width = {val3}, height = {val4}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}");

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

}

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

sendMessage = Encoding.ASCII.GetBytes(text);

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

break;

case 6:

Console.WriteLine("command:Draw Ellipse");

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

if (SizeCheck(val1, val2, width, height) == 1)

{

text = "Error:Figure out of bounds";

Console.WriteLine(text);

}

else

{

Console.WriteLine($"Coordinates: x1 = {val1}, y1 = {val2}, radius x = {val3}, radius y = {val4}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}");

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

}

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

sendMessage = Encoding.ASCII.GetBytes(text);

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

break;

case 7:

Console.WriteLine("command:Fill Ellipse");

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

if (SizeCheck(val1, val2, width, height) == 1)

{

text = "Error:Figure out of bounds";

Console.WriteLine(text);

}

else

{

Console.WriteLine($"Coordinates: x1 = {val1}, y1 = {val2}, radius x = {val3}, radius y = {val4}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}");

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

}

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

sendMessage = Encoding.ASCII.GetBytes(text);

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

break;

case 8:

Console.WriteLine("command:Draw Circle");

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

if (SizeCheck(val1, val2, width, height) == 1)

{

text = "Error:Figure out of bounds";

Console.WriteLine(text);

}

else

{

Console.WriteLine($"Coordinates: x1 = {val1}, y1 = {val2}, radius = {val3}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}");

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

}

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

sendMessage = Encoding.ASCII.GetBytes(text);

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

break;

case 9:

Console.WriteLine("command:Fill Circle");

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

if (SizeCheck(val1, val2, width, height) == 1)

{

text = "Error:Figure out of bounds";

Console.WriteLine(text);

}

else

{

Console.WriteLine($"Coordinates: x1 = {val1}, y1 = {val2}, radius = {val3}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}");

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

}

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

sendMessage = Encoding.ASCII.GetBytes(text);

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

break;

case 10:

Console.WriteLine("command:Draw Rounded Rectangle");

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

if (SizeCheck(val1, val2, width, height) == 1)

{

text = "Error:Figure out of bounds";

Console.WriteLine(text);

}

else

{

Console.WriteLine($"Coordinates: x1 = {val1}, y1 = {val2}, width = {val3}, height = {val4}, radius = {val5}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}");

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

}

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

sendMessage = Encoding.ASCII.GetBytes(text);

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

break;

case 11:

Console.WriteLine("command:Fill Rounded Rectangle");

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

if (SizeCheck(val1, val2, width, height) == 1)

{

text = "Error:Figure out of bounds";

Console.WriteLine(text);

}

else

{

Console.WriteLine($"Coordinates: x1 = {val1}, y1 = {val2}, width = {val3}, height = {val4}, radius = {val5}, color: Red = {RGB[0]}, Green = {RGB[1]}, Blue = {RGB[2]}");

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

}

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

sendMessage = Encoding.ASCII.GetBytes(text);

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

break;

case 12:

Console.WriteLine("command:Draw Text");

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

if (SizeCheck(val1, val2, width, height) == 1)

{

text = "Error:Figure out of bounds";

Console.WriteLine(text);

}

else

{

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

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

}

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

sendMessage = Encoding.ASCII.GetBytes(text);

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

break;

case 13:

Console.WriteLine("command:Draw Image");

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

if (SizeCheck(val1, val2, width, height) == 1)

{

text = "Error:Figure out of bounds";

Console.WriteLine(text);

}

else

{

Console.WriteLine($"Coordinates: x1 = {val1}, y1 = {val2}");

text = $"Image Drawn: Coordinates: x1 = {val1}, y1 = {val2}, Image Width = {val3}, Image Height = {val4}";

}

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

sendMessage = Encoding.ASCII.GetBytes(text);

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

break;

case 14:

Console.WriteLine("Command: Set Orientation");

text = $"Orientation set:";

sendMessage = Encoding.ASCII.GetBytes(text);

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

break;

case 15:

Console.WriteLine("Command: Get Width");

text = $"Width: {width} px";

sendMessage = Encoding.ASCII.GetBytes(text);

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

break;

case 16:

Console.WriteLine("Command: Get Height");

text = $"Height: {height} px";

Console.WriteLine(text);

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

sendMessage = Encoding.ASCII.GetBytes(text);

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

break;

case 254:

Console.WriteLine("Command: Set Screen Size");

SetScreenSize(RecievedData, out width, out height);

Console.WriteLine($"Width: {width}");

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

text = $"Screen Size Set: Width = {width}, Height = {height}";

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

sendMessage = Encoding.ASCII.GetBytes(text);

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

break;

}

}

}

catch (SocketException e)

{

Console.WriteLine(e);

}

finally

{

server.Close();

}

}

public static void Main()

{

StartServer();

}

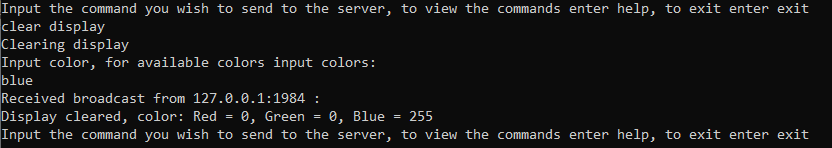
}

}

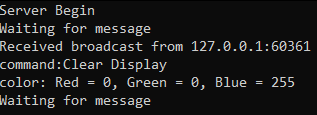
Перевірка сервера:

Команда Clear Display

Клієнт:

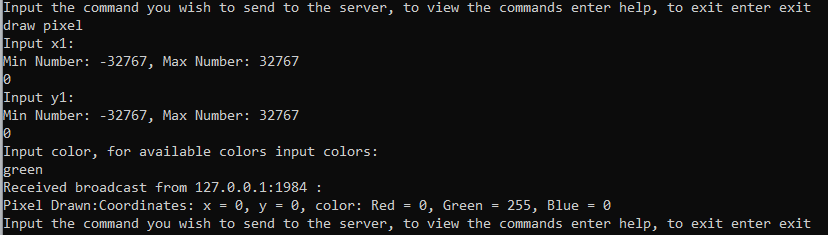


Сервер:

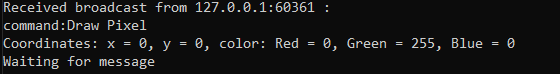


Draw Pixel

Клієнт

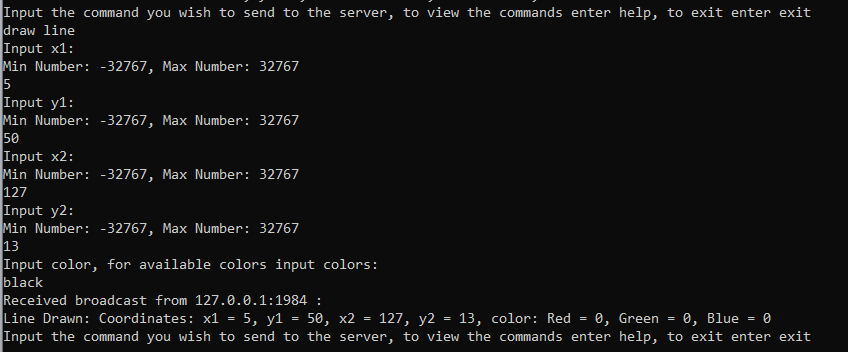


Сервер

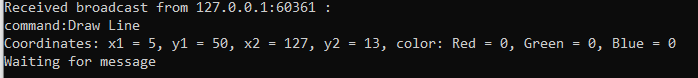


Draw Line

Клієнт

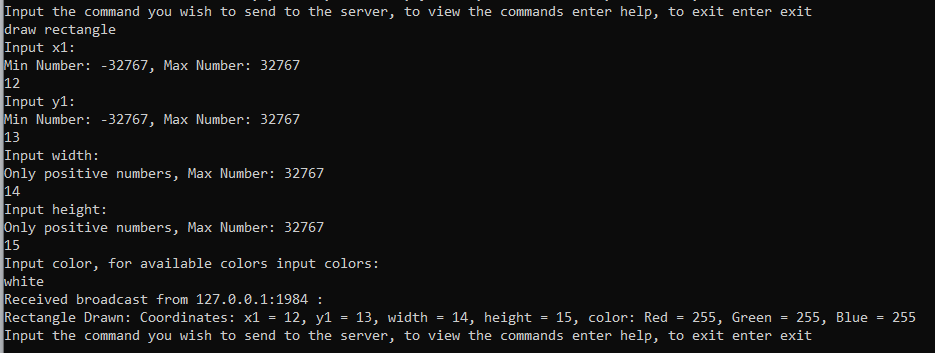


Сервер

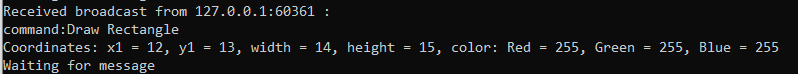


Draw Rectangle

Клієнт

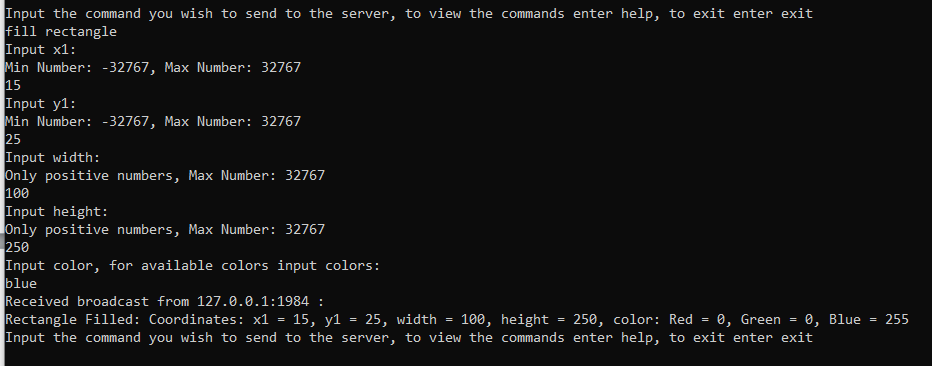


Сервер

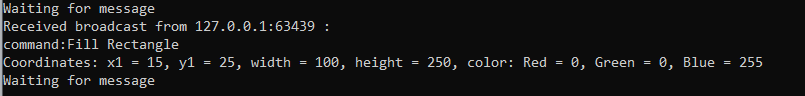


Fill Rectangle

Клієнт

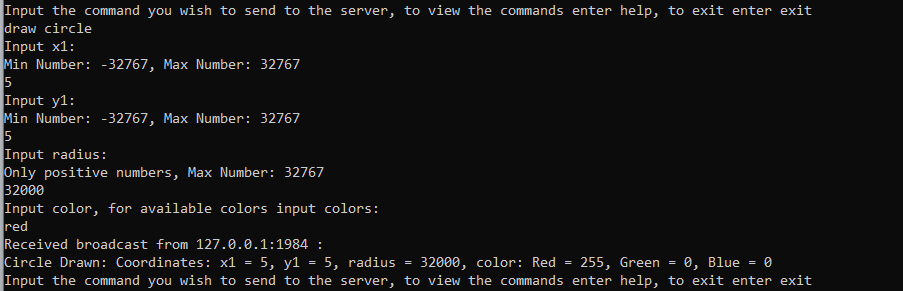


Сервер

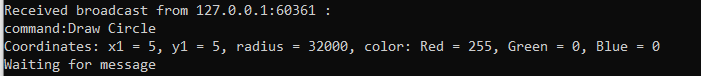


Draw Circle

Клієнт

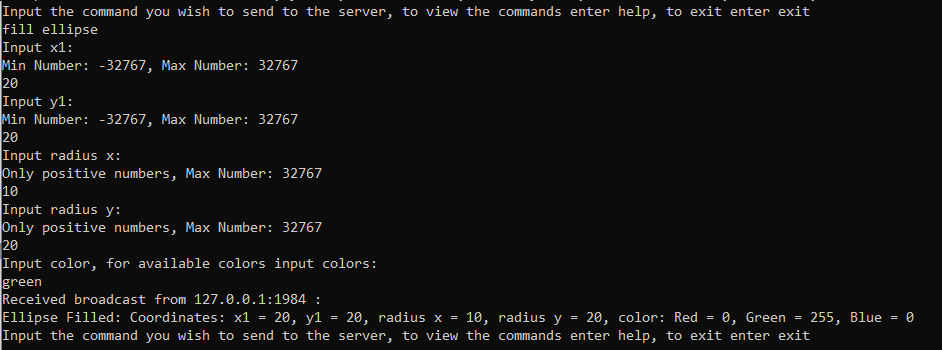


Сервер

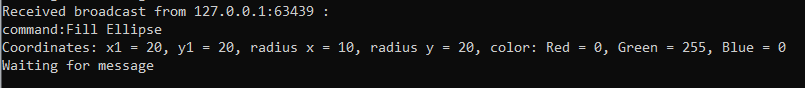


Fill Ellipse

Клієнт

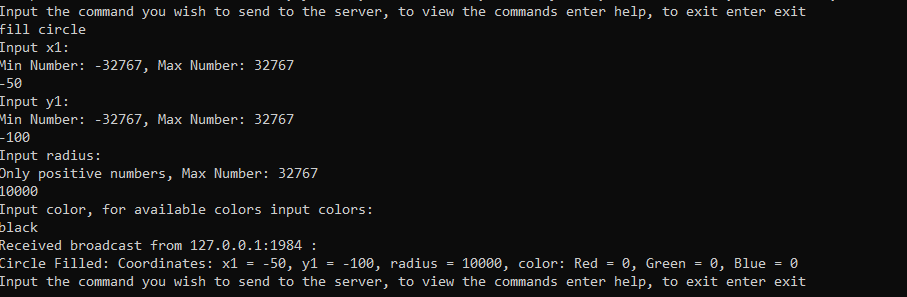


Сервер

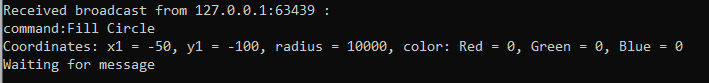


Fill Circle

Клієнт

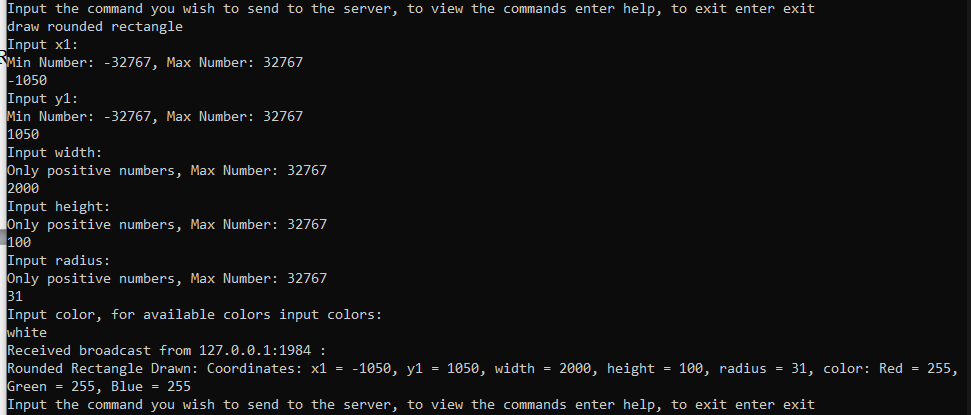


Сервер



Draw Rounded Rectangle

Клієнт

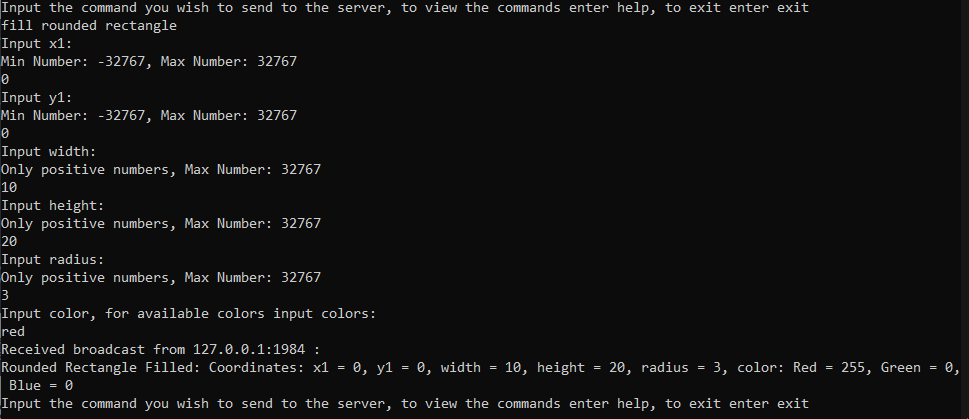


Сервер

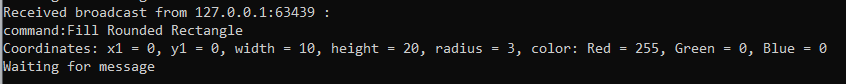


Fill Rounded Rectangle

Клієнт

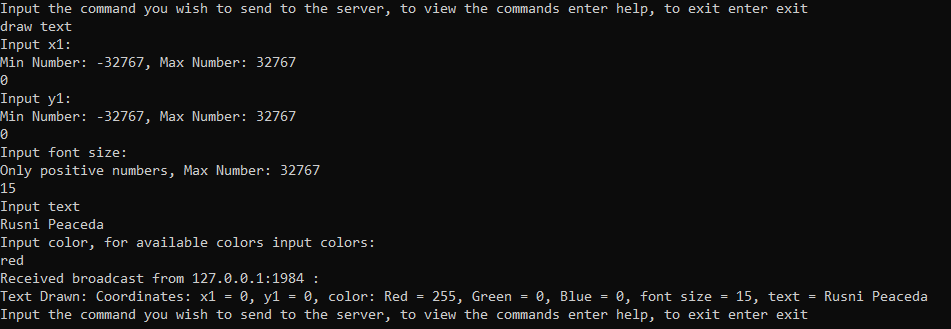


Сервер



Draw Text

Клієнт

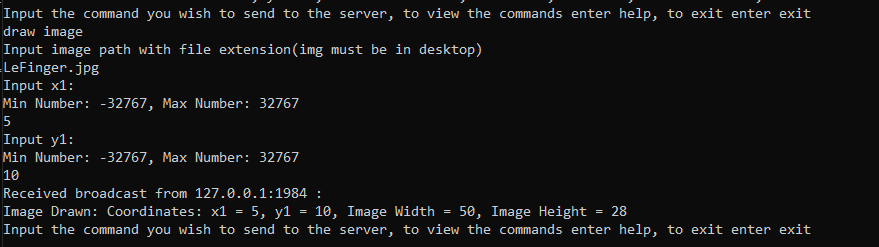


Сервер

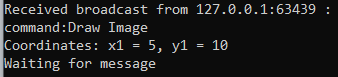


Draw Image

Клієнт



Сервер



Висновок: Протягом лабораторної роботи, я розробив програму для реалізації протоколу управління дисплейним модулем.