Лабораторна робота № 6

Удосконалення програм емулятора дисплейного модуля і клієнта.

Виконав студент
Групи кн21-1
Кончич Данііл
Варіант14

Етапи виконання лабораторної роботи:

1. Розширити специфікацію протоколу обміну даними для підтримки просунутого набору команд :

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

get width:

get height:

2. Внести виправлення в код емулятора дисплейного модуля для підтримки нових команд (див. п1).

3. Внести виправлення в код реалізації інтерфейсу клієнта GrpahicsLib.h для підтримки нових команд (див. п1).

4. Опис попередніх команд в специфікації не міняти.

Хід роботи

Лістинг програми(Client):

```
unit Maim;

interface

uses
  System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,
  FMX.Types, FMX.Controls, FMX.Forms, FMX.Graphics, FMX.Dialogs,
```

```
  FMX.Controls.Presentation, FMX.StdCtrls, IdBaseComponent, IdComponent,
  IdUDPBase, IdUDPClient, FMX.Memo.Types, FMX.ScrollBox, FMX.Memo, System.DateUtils,
idGlobal,
  FMX.Edit, FMX.ComboEdit, FMX.Objects, IdUDPServer, IdSocketHandle;

type TPacket = packed record
  msLen:Byte;
  colorarray:array [1..40,1..40] of cardinal;
  w:integer;
  h:integer;
  msg:string[255];
end;

const commands: array [1..11] of string = (
    'drawline',          'drawellipse',          'drawtext',          'clear',
'drawimage','fillroundedrectangle','drawpixel','drawsymbol','setorientation','getwidth','gethe
ight'
);

// Перечисление для типов команд
type  TCommand=(DRAW_LINE,  DRAW_ELLIPSE,  DRAW_TEXT,  CLEAR,  DRAW_IMAGE,
FILL_ROUNDED_RECTANGLE, DRAW_PIXEL, DRAW_SYMBOL, SET_ORIENTATION, GET_WIDTH, GET_HEIGHT);

type
  TForm1 = class(TForm)
    IdUDPClient1: TIdUDPClient;
    Button1: TButton;
    Memo1: TMemo;
    ComboEdit1: TComboEdit;
    Label1: TLabel;
    Image1: TImage;
    IdUDPServer1: TIdUDPServer;
    procedure Button1Click(Sender: TObject);
    procedure IdUDPServer1UDPRead(AThread: TIdUDPListenerThread;
      const AData: TIdBytes; ABinding: TIdSocketHandle);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    bmp:TBitmap;
    packet:TPacket;
    send_data:TIdBytes;
    sendcommand:TCommand;
  public
    { Public declarations }
    function DrawPixelEncode(const sendcommand, px1,py1,parcolor:string):string;
    function SetOrientationEncode(const sendcommand, deg:string):string;
```

```
          function GetWidthEncode(const sendcommand:string):string;
          function GetHeightEncode(const sendcommand:string):string;
          function                    DrawLineEncode(const                    sendcommand,
parx1,pary1,parx2,pary2,parcolor:string):string;
          function DrawSymbolEncode(const sendcommand, symbol, x,y,parcolor:string):string;
          function                  DrawEllipseEncode(const                   sendcommand,
elx1,ely1,elx2,ely2,parcolor:string):string;
          function                    DrawTextEncode(const                    sendcommand,
tx1,ty1,tx2,ty2,text,parcolor:string):string;
          function ClearEncode(const sendcommand:string; const parcolor:string):string;
          function DrawImageEncode(const sendcommand:string; width,heigth:string):string;
          function           FillRoundedRectangleEncode(const           sendcommand:string;
px1,py1,px2,py2,radius,parcolor:string):string;
        end;

      var
        Form1: TForm1;

      implementation

      {$R *.fmx}

      procedure TForm1.Button1Click(Sender: TObject);
      var spl:TArray<string>; s:string; i:integer; iw,jw:integer; b:TBitmapData;
      begin

        packet.msLen:=Length(Memo1.Text);
        SetLength(packet.msg,packet.msLen);

        s:=Memo1.Text;
        spl:=s.Split([' ']);

        for i:=1 to 11 do
        begin
          if commands[i]=spl[0] then
          begin
            sendcommand:=TCommand(i-1);
            case sendcommand of
            TCommand.DRAW_LINE:
              packet.msg:=DrawLineEncode((i-1).ToString,spl[1],spl[2],spl[3],spl[4],spl[5]);
            TCommand.DRAW_ELLIPSE:
              packet.msg:=DrawEllipseEncode((i-
1).ToString,spl[1],spl[2],spl[3],spl[4],spl[5]);
            TCommand.DRAW_TEXT:
              packet.msg:=DrawTextEncode((i-
1).ToString,spl[1],spl[2],spl[3],spl[4],spl[5],spl[6]);
```

```
                TCommand.CLEAR:
                  packet.msg:=ClearEncode((i-1).ToString,spl[1]);
                TCommand.DRAW_IMAGE:
                begin
                  packet.msg:=DrawImageEncode((i-1).ToString,spl[1],spl[2]);
                  bmp:=TBitmap.CreateFromFile(spl[3]);

                  packet.w:=bmp.Width;
                  packet.h:=bmp.Height;


                  bmp.Map(TMapAccess.Read,b);

                  for iw:=1 to Round(bmp.Width) do
                  for jw:=1 to Round(bmp.Height) do
                    packet.colorarray[iw,jw]:=b.GetPixel(iw,jw);

                  bmp.Unmap(b);
                  Image1.Bitmap.Assign(bmp);

                end;
                TCommand.FILL_ROUNDED_RECTANGLE:
                begin
                  packet.msg:=FillRoundedRectangleEncode((i-
1).ToString,spl[1],spl[2],spl[3],spl[4],spl[5],spl[6]);
                end;
                TCommand.DRAW_PIXEL:
                begin
                  packet.msg:=DrawPixelEncode((i-1).ToString,spl[1],spl[2],spl[3]);
                end;
                TCommand.DRAW_SYMBOL:
                begin
                  packet.msg:=DrawSymbolEncode((i-1).ToString,spl[1],spl[2],spl[3],spl[4]);
                end;
                TCommand.SET_ORIENTATION:
                begin
                  packet.msg:=SetOrientationEncode((i-1).ToString,spl[1]);
                end;
                TCommand.GET_WIDTH:
                begin
                  packet.msg:=GetWidthEncode((i-1).ToString);
                end;
                TCommand.GET_HEIGHT:
                begin
                  packet.msg:=GetHeightEncode((i-1).ToString);
                end;
```

```
        end;
      end;
    end;

    IdUDPClient1.Active:=true;
    IdUDPClient1.Port:=5000;
    IdUDPClient1.Host:=ComboEdit1.Text;
    IdUDPClient1.Connect;

    if IdUDPClient1.Connected then
    begin
      SetLength(send_data,sizeof(packet));
      Move(packet,send_data[0],sizeof(packet));
      IdUDPClient1.SendBuffer(send_data);
    end;

    IdUDPClient1.Active:=false;

  end;

  function TForm1.ClearEncode(const sendcommand:string; const parcolor: string): string;
  var command:integer;
  begin
  try
      command:=Integer.Parse(sendcommand);
      Result:=command.ToString+' '+parcolor;
    except on EConvertError do
    begin
      ShowMessage('Цвет неверный!!!');
      Result:='3 '+'000000';
    end;
    end;
  end;

  function TForm1.DrawSymbolEncode(const sendcommand, symbol, x, y, parcolor: string):
string;
  var xx,yy: Double; command:integer;
  begin
    try
      xx:=Double.Parse(x);
      yy:=Double.Parse(y);
      command:=Integer.Parse(sendcommand);
      Result:=command.ToString+' '+symbol+' '+xx.ToString+' '+yy.ToString+' '+parcolor;
    except on EConvertError do
    begin
      ShowMessage('Координаты буквы неверны!!!');
```

```
        Result:='7 0 0 0 0';
      end;
    end;
    end;

    function TForm1.DrawEllipseEncode(const sendcommand, elx1, ely1, elx2, ely2,
      parcolor: string): string;
    var x1,y1,x2,y2,command:integer;
    begin
      try
        x1:=Integer.Parse(elx1);
        y1:=Integer.Parse(ely1);
        x2:=Integer.Parse(elx2);
        y2:=Integer.Parse(ely2);
        command:=Integer.Parse(sendcommand);
        Result:=command.ToString+'    '+x1.ToString+'    '+y1.ToString+'    '+x2.ToString+'
'+y2.ToString+' '+parcolor;
      except on EConvertError do
      begin
        ShowMessage('Координаты эллипса неверны!!!');
        Result:='1 0 0 0 0 '+parcolor;
      end;
      end;
    end;

    function TForm1.DrawImageEncode(const sendcommand: string; width,
      heigth: string): string;
    var w,h,command:integer;
    begin
      try
        w:=Integer.Parse(width);
        h:=Integer.Parse(heigth);
        command:=Integer.Parse(sendcommand);
        Result:=command.ToString+' '+w.ToString+' '+h.ToString;
      except on EConvertError do
      begin
        ShowMessage('размеры неверны!!!');
        Result:='4 0 0';
      end;
      end;
    end;

    function TForm1.DrawLineEncode(const sendcommand, parx1, pary1, parx2, pary2,
      parcolor: string): string;
    var x1,y1,x2,y2,command:integer;
    begin
```

```
  try
    x1:=Integer.Parse(parx1);
    y1:=Integer.Parse(pary1);
    x2:=Integer.Parse(parx2);
    y2:=Integer.Parse(pary2);
    command:=Integer.Parse(sendcommand);
    Result:=command.ToString+' '+x1.ToString+' '+y1.ToString+' '+x2.ToString+' '
    +y2.ToString+' '+parcolor;
  except on EConvertError do
  begin
    ShowMessage('Координаты линии неверны!!!');
    Result:='0 0 0 0 0 '+parcolor;
  end;
  end;
end;

function TForm1.DrawPixelEncode(const sendcommand, px1, py1,
  parcolor: string): string;
var x1,y1,command:integer;
begin
  try
    x1:=Integer.Parse(px1);
    y1:=Integer.Parse(py1);
    command:=Integer.Parse(sendcommand);
    Result:=command.ToString+' '+x1.ToString+' '+y1.ToString+' '+parcolor;
  except on EConvertError do
  begin
    ShowMessage('Координаты линии неверны!!!');
    Result:='6 0 0 '+parcolor;
  end;
  end;
end;

function TForm1.DrawTextEncode(const sendcommand, tx1, ty1, tx2, ty2, text,
  parcolor: string): string;
var x1,y1,x2,y2,command:integer;
begin
  try
    x1:=Integer.Parse(tx1);
    y1:=Integer.Parse(ty1);
    x2:=Integer.Parse(tx2);
    y2:=Integer.Parse(ty2);
    command:=Integer.Parse(sendcommand);
    Result:=command.ToString+' '+x1.ToString+' '+y1.ToString+' '+x2.ToString+' '
    +y2.ToString+' '+text+' '+parcolor;
  except on EConvertError do
```

```
  begin
    ShowMessage('Координаты линии неверны!!!');
    Result:='2 0 0 0 0 '+text+' '+parcolor;
  end;
  end;
end;

function TForm1.FillRoundedRectangleEncode(const sendcommand: string; px1, py1,
  px2, py2, radius, parcolor: string): string;
var x1,y1,x2,y2,rad,command,color:integer;
begin
  try
    x1:=Integer.Parse(px1);
    y1:=Integer.Parse(py1);
    x2:=Integer.Parse(px2);
    y2:=Integer.Parse(py2);
    rad:=Integer.Parse(radius);
    command:=Integer.Parse(sendcommand);
    Result:=command.ToString+' '+x1.ToString+' '+y1.ToString+' '+
    x2.ToString+' '+y2.ToString+' '+rad.ToString+' '+parcolor;
  except on EConvertError do
  begin
    ShowMessage('Ошибка!!!');
    Result:='5 0 0 0 0 0 0';
  end;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  IdUDPServer1.Active:=True;
end;

function TForm1.GetHeightEncode(const sendcommand: string): string;
var command:integer;
begin
  try
    Result:=command.ToString;
  except on EConvertError do
  begin
    ShowMessage('Ошибка!!!');
    Result:='10 0';
  end;
  end;
end;
function TForm1.GetWidthEncode(const sendcommand: string): string;
```

```
var command:integer;
begin
  try
    Result:=command.ToString;
  except on EConvertError do
  begin
    ShowMessage('Ошибка!!!');
    Result:='9 0';
  end;
  end;
end;

procedure TForm1.IdUDPServer1UDPRead(AThread: TIdUDPListenerThread;
  const AData: TIdBytes; ABinding: TIdSocketHandle);
  var i:integer; s:string; spl:TArray<string>;
begin
  s:='';
  try
    i:=0;
    while(AData[i]<>0) do
    begin

      s:=s+Chr(AData[i]);
      i:=i+1;
    end;
  finally
      Memo1.Lines.Clear;
      Memo1.Lines.Add(s);
  end;
end;

function TForm1.SetOrientationEncode(const sendcommand, deg: string): string;
var command,degrees:integer;
begin
  try
    degrees:=Integer.Parse(deg);
    command:=Integer.Parse(sendcommand);
    Result:=command.ToString+' '+degrees.ToString;
  except on EConvertError do
  begin
    ShowMessage('Ошибка!!!');
    Result:='8 0';
  end;
  end;
end;
end.
```

Лістинг програми(Server):

```
unit Main;

interface

uses
  System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,
  FMX.Types, FMX.Controls, FMX.Forms, FMX.Graphics, FMX.Dialogs,
  FMX.Controls.Presentation, FMX.StdCtrls, IdBaseComponent, IdComponent,
  IdUDPBase, IdUDPServer, IdGlobal, IdSocketHandle, FMX.Memo.Types,
  FMX.ScrollBox, FMX.Memo, System.DateUtils, FMX.Objects, MyCommands,
System.Generics.Collections,
  IdUDPClient, FMX.Edit;

const symbols: array [1..8] of string = (
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'
);




// Запись для приема данных от клиента
type TPacket = packed record
  msLen:Byte;
  colorarray:array [1..40,1..40] of cardinal;
  w:integer;
  h:integer;
  msg:string[255];
end;

// Параметры картинки

type TPicData = class
  pic:TBitmap;
  x:Double;
  y:Double;
  constructor Create(var x,y:Double;var pic:TBitmap); overload;
end;
```

```
// Параметры надписи

type TTextData = class
  text:string;
  x1:Double;
  y1:Double;
  x2:Double;
  y2:Double;
  color:string;
  constructor Create(var text:string; var x1,y1,x2,y2:Double; color:string); overload;
end;


type TEllipseData = class
  x1:Double;
  y1:Double;
  x2:Double;
  y2:Double;
  color:string;
  constructor Create(var x1,y1,x2,y2:Double; color:string); overload;
end;


type TPixelData = class
  x1:Double;
  y1:Double;
  color:string;
  constructor Create(var x1,y1:Double; color:string); overload;
end;

type TSymbolData = class
  x:Double;
  y:Double;
  color:string;
  symbpos:integer;
  constructor Create(var x, y : Double; color : string;  symbpos : integer); overload;
end;
```

```
type TFillRoundedRectangleData = class
  x1:Integer;
  y1:Integer;
  x2:Integer;
  y2:Integer;
  radius:Integer;
  color:string;
  constructor Create(var x1,y1,x2,y2,radius:Integer;color:string); overload;
end;



// Параметры линии


type TLineData = class
  p1:TPointF;
  p2:TPointF;
  color:string;
  constructor Create(var p1,p2:TPointF; color:string); overload;
end;


// Перечисление для типов команд
type TCommand=(DRAW_LINE, DRAW_ELLIPSE, DRAW_TEXT, CLEAR, DRAW_IMAGE, FILL_ROUNDED_RECTANGLE,
DRAW_PIXEL, DRAW_SYMBOL, SET_ORIENTATION, GET_WIDTH, GET_HEIGHT);


type
  TForm1 = class(TForm)
    IdUDPServer1: TIdUDPServer;
    ToolBar1: TToolBar;
    Label2: TLabel;
    PaintBox1: TPaintBox;
    IdUDPClient1: TIdUDPClient;
    Edit1: TEdit;
    procedure FormCreate(Sender: TObject);
    procedure IdUDPServer1UDPRead(AThread: TIdUDPListenerThread;
      const AData: TIdBytes; ABinding: TIdSocketHandle);
    procedure PaintBox1Paint(Sender: TObject; Canvas: TCanvas);
  private
```

```
    { Private declarations }
    bmp:TBitmap;
    packet:TPacket;
    command:TCommand;
    drawcommand:integer;
    piclist:TList<TPicData>;
    textlist:TList<TTextData>;
    linelist:TList<TLineData>;
    ellipselist:TList<TEllipseData>;
    fillroundedrectanglelist:TList<TFillRoundedRectangleData>;
    pixellist:TList<TPixelData>;
    symbollist:TList<TSymbolData>;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.fmx}

procedure TForm1.FormCreate(Sender: TObject);
begin
  IdUDPServer1.Active:=true;
  TMyCommands.linepath:=TPathData.Create;
  TMyCommands.ellipsepath:=TPathData.Create;
  TMyCommands.clearcolor:='000000';
  piclist:=TList<TPicData>.Create;
  textlist:=TList<TTextData>.Create;
  linelist:=TList<TLineData>.Create;
  ellipselist:=TList<TEllipseData>.Create;
  fillroundedrectanglelist:=TList<TFillRoundedRectangleData>.Create;
  pixellist:=TList<TPixelData>.Create;
  symbollist:=TList<TSymbolData>.Create;
end;
```

```
procedure TForm1.IdUDPServer1UDPRead(AThread: TIdUDPListenerThread;
  const AData: TIdBytes; ABinding: TIdSocketHandle);
var s:string; i:integer;     spl:TArray<string>; iw,jw:integer;
    b1:TBitmapData; picdata:TPicData; textdata:TTextData;
    linedata:TLineData; ellipsedata:TEllipseData;
    fillroundedrectangledata:TFillRoundedRectangleData;
    pixeldata:TPixelData; px,py:Double; mysymboldata:TSymbolData;
    symbolpos:integer; symbolx,symboly:Double;  symbolcolor:string;
begin

          Move(AData[0],packet,sizeof(packet));
          s:=packet.msg;
          spl:=s.Split([' ']);

          // Парсим полученную команду от клиента

          command:=TCommand(Integer.Parse(spl[0]));

         case command of
          TCommand.DRAW_LINE:
          begin
            drawcommand:=Integer.Parse(spl[0]);
            TMyCommands.PrepareLine(spl[1],spl[2],spl[3],spl[4],spl[5]);
            linedata:=TLineData.Create(TMyCommands.p1,TMyCommands.p2,TMyCommands.linecolor);
            linelist.Add(linedata);
            PaintBox1.Repaint;
          end;
          TCommand.DRAW_ELLIPSE:
          begin
            drawcommand:=Integer.Parse(spl[0]);
            TMyCommands.PrepareEllipse(spl[1],spl[2],spl[3],spl[4],spl[5]);
            ellipsedata:=TEllipseData.Create(TMyCommands.x1_ellipse,TMyCommands.y1_ellipse,
            TMyCommands.x2_ellipse,TMyCommands.y2_ellipse,TMyCommands.ellipsecolor);
            ellipselist.Add(ellipsedata);
            PaintBox1.Repaint;
          end;
          TCommand.DRAW_TEXT:
          begin
```

```
            drawcommand:=Integer.Parse(spl[0]);
            TMyCommands.PrepareText(spl[1],spl[2],spl[3],spl[4],spl[5],spl[6]);

textdata:=TTextData.Create(TMyCommands.textout,TMyCommands.x1_text,TMyCommands.y1_text,
            TMyCommands.x2_text,TMyCommands.y2_text,TMyCommands.textcolor);
            textlist.Add(textdata);
            PaintBox1.Repaint;
          end;
          TCommand.CLEAR:
          begin
            drawcommand:=Integer.Parse(spl[0]);
            TMyCommands.PrepareClear(spl[1]);
            piclist.Clear;
            textlist.Clear;
            linelist.Clear;
            pixellist.Clear;
            symbollist.Clear;
            ellipselist.Clear;
            fillroundedrectanglelist.Clear;
            Form1.Fill.Color:=StrToInt('$ff'+TMyCommands.clearcolor);
            PaintBox1.Repaint;
          end;
          TCommand.DRAW_IMAGE:
          begin
            drawcommand:=Integer.Parse(spl[0]);
            TMyCommands.PrepareDrawImage(spl[1],spl[2]);
            bmp:=TBitmap.Create();

            bmp.SetSize(packet.w,packet.h);

            bmp.Map(TMapAccess.Write,b1);

            for iw:=1 to Round(bmp.Width) do
            for jw:=1 to Round(bmp.Height) do
            begin
              b1.SetPixel(iw,jw,packet.colorarray[iw,jw]);
            end;
            bmp.Unmap(b1);
```

```
            picdata:=TPicData.Create(TMyCommands.ximage,TMyCommands.yimage,bmp);
            piclist.Add(picdata);

            PaintBox1.Repaint;
          end;
          TCommand.FILL_ROUNDED_RECTANGLE:
          begin
TMyCommands.PrepareFillRoundedRectangle(spl[1],spl[2],spl[3],spl[4],spl[5],spl[6]);
fillroundedrectangledata:=TFillRoundedRectangleData.Create(TMyCommands.x1,TMyCommands.y1,
TMyCommands.x2,TMyCommands.y2,TMyCommands.radius,TMyCommands.fillroundedrectanglecolor);
            fillroundedrectanglelist.Add(fillroundedrectangledata);
            PaintBox1.Repaint;
          end;
          TCommand.DRAW_PIXEL:
          begin
            TMyCommands.PreparePixel(spl[1],spl[2],spl[3]);
            px:=TMyCommands.ppoint.X;
            py:=TMyCommands.ppoint.Y;
            pixeldata:=TPixelData.Create(px, py, TMyCommands.pixelcolor);
            pixellist.Add(pixeldata);
            PaintBox1.Repaint;
          end;
          TCommand.DRAW_SYMBOL:
          begin
            TMyCommands.PrepareSymbol(spl[1],spl[2],spl[3],spl[4]);
            for symbolpos:=1 to 8 do
            begin
              if TMyCommands.symbol=symbols[symbolpos] then
              begin
                symbolx:=TMyCommands.sx;
                symboly:=TMyCommands.sy;
                symbolcolor:=TMyCommands.symbolcolor;
                mysymboldata:=TSymbolData.Create(symbolx, symboly, symbolcolor, (symbolpos-1));
                symbollist.Add(mysymboldata);
              end;
```

```
  end;

  PaintBox1.Repaint;
end;
TCommand.SET_ORIENTATION:
begin
  TMyCommands.PrepareOrientation(spl[1]);
  PaintBox1.RotationAngle:=TMyCommands.degrees;
end;
TCommand.GET_WIDTH:
begin
    IdUDPClient1.Active:=true;
    IdUDPClient1.Port:=5001;
    IdUDPClient1.Host:=Edit1.Text;
    IdUDPClient1.Connect;

    if IdUDPClient1.Connected then
    begin
      IdUDPClient1.Send('Canvas width: '+PaintBox1.Width.ToString);
    end;

    IdUDPClient1.Active:=false;
end;
TCommand.GET_HEIGHT:
begin
    IdUDPClient1.Active:=true;
    IdUDPClient1.Port:=5001;
    IdUDPClient1.Host:=Edit1.Text;
    IdUDPClient1.Connect;

    if IdUDPClient1.Connected then
    begin
      IdUDPClient1.Send('Canvas height: '+PaintBox1.Height.ToString);
    end;

    IdUDPClient1.Active:=false;
end;
```

```
        end;

end;


procedure TForm1.PaintBox1Paint(Sender: TObject; Canvas: TCanvas);
var i:integer; p:TPicData; t:TTextData; l:TLineData;  e:TEllipseData;
    frr:TFillRoundedRectangleData; pixel:TPixelData; a:TSymbolData;
begin
  PaintBox1.Canvas.BeginScene();

        for l in linelist do
          TMyCommands.DrawMyLine(l.p1,l.p2,Canvas,StrToInt('$ff'+l.color));

        for e in ellipselist do
          TMyCommands.DrawMyEllipse(e.x1,e.y1,e.x2,e.y2,Canvas,StrToInt('$ff'+e.color));

        for t in textlist do
          TMyCommands.DrawMyText(t.x1,t.y1,t.x2,t.y2,
             t.text, 30, Canvas, StrToInt('$ff'+t.color));

        for p in piclist do
          TMyCommands.DrawImage(p.x,p.y,p.pic,Canvas);

        for frr in fillroundedrectanglelist do
          TMyCommands.FillRoundedRectangle(frr.x1,frr.y1,frr.x2,frr.y2,frr.radius,
            Canvas,StrToInt('$ff'+frr.color));

        for pixel in pixellist do
        begin
          TMyCommands.DrawMyPixel(TPointF.Create(pixel.x1,pixel.y1),
            Canvas,StrToInt('$ff'+pixel.color));
        end;

        for a in symbollist do
        begin
```

```
TMyCommands.DrawSymbol(a.symbpos,TPointF.Create(a.x,a.y),Canvas,StrToInt('$ff'+a.color));
        end;

  PaintBox1.Canvas.EndScene;

end;




{ TPicData }

constructor TPicData.Create(var x, y: Double; var pic: TBitmap);
begin
  Self.x:=x;
  Self.y:=y;
  Self.pic:=pic;
end;

{ TTextData }

constructor TTextData.Create(var text:string; var x1,y1,x2,y2:Double; color:string);
begin
  Self.text:=text;
  Self.x1:=x1;
  Self.y1:=y1;
  Self.x2:=x2;
  Self.y2:=y2;
  Self.color:=color;
end;

{ TLineData }

constructor TLineData.Create(var p1,p2:TPointF; color:string);
begin
  Self.p1:=p1;
  Self.p2:=p2;
```

```
  Self.color:=color;
end;

{ TEllipseData }

constructor TEllipseData.Create(var x1, y1, x2, y2: Double; color: string);
begin
  Self.x1:=x1;
  Self.y1:=y1;
  Self.x2:=x2;
  Self.y2:=y2;
  Self.color:=color;
end;

{ TFillRoundedRectangleData }

constructor TFillRoundedRectangleData.Create(var x1, y1, x2, y2,
  radius: Integer; color: string);
begin
  Self.x1:=x1;
  Self.y1:=y1;
  Self.x2:=x2;
  Self.y2:=y2;
  Self.radius:=radius;
  Self.color:=color;
end;

{ TPixelData }

constructor TPixelData.Create(var x1, y1: Double; color: string);
begin
  Self.x1:=x1;
  Self.y1:=y1;
  Self.color:=color;
end;

{ TAData }
```

```
constructor TSymbolData.Create(var x, y: Double; color: string; symbpos : integer);
begin
  Self.symbpos:=symbpos;
  Self.x:=x;
  Self.y:=y;
  Self.color:=color;
end;

end.
unit MyCommands;

interface
uses
  System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,
  FMX.Types, FMX.Controls, FMX.Forms, FMX.Graphics, FMX.Dialogs,
  FMX.Controls.Presentation, FMX.StdCtrls, IdBaseComponent, IdComponent,
  IdUDPBase, IdUDPServer, IdGlobal, IdSocketHandle, FMX.Memo.Types,
  FMX.ScrollBox, FMX.Memo, System.DateUtils, FMX.Objects, System.Generics.Collections;



type
  TMyCommands=class
  public
  class var linepath:TPathData;
  class var ellipsepath:TPathData;
  class var p1:TPointF;
  class var p2:TPointF;
  class var sx:Double;
  class var sy:Double;
  class var degrees:integer;
  class var symbol:string;
  class var ppoint:TPointF;
  class var linecolor:string;
  class var ellipsecolor:string;
  class var textcolor:string;
  class var symbolcolor:string;
  class var pixelcolor:string;
```

```
  class var fillroundedrectanglecolor:string;

  class var clearcolor:string;

  class var ximage,yimage:Double;

  class var x1_text,y1_text,x2_text,y2_text:Double;

  class var x1,y1,x2,y2,radius:Integer;

  class var x1_ellipse,y1_ellipse,x2_ellipse,y2_ellipse:Double;

  class var textout:string;

  class procedure DrawImage(const x, y: double; const bmp: TBitmap; const Canvas:TCanvas);

  class procedure DrawMyLine(const p1,p2:TPointF;const Canvas:TCanvas; const color:Cardinal);

  class procedure DrawMyPixel(const ppoint:TPointF; const Canvas:TCanvas; const color:Cardinal);

  class procedure DrawSymbol(const mysymbol:integer; ppoint:TPointF; const Canvas:TCanvas; const
color:Cardinal);

  class procedure DrawMyEllipse(const x1_ellipse,y1_ellipse,x2_ellipse,y2_ellipse:Double; const
Canvas:TCanvas; const color:Cardinal);

  class procedure FillRoundedRectangle(const x1,y1,x2,y2,radius:Integer; const Canvas:TCanvas;
const color:Cardinal);

  class    procedure    DrawMyText(const    x1_text,y1_text,x2_text,y2_text:Double;    const
textout:string; const fontsize:integer; const Canvas:TCanvas; const color:Cardinal);

  class procedure ClearCanvas(const Form:TForm; const Canvas:TCanvas; const color:Cardinal);

  class function PreparePixel(const x1,y1,parcolor:string):integer;

  class function PrepareLine(const parx1,pary1,parx2,pary2,parcolor:string):integer;

  class function PrepareEllipse(const elx1,ely1,elx2,ely2,parcolor:string):integer;

  class function PrepareText(const tx1,ty1,tx2,ty2,text,parcolor:string):integer;

  class function PrepareSymbol(const symbol, sx, sy,parcolor:string):integer;

  class function PrepareFillRoundedRectangle(const x1,y1,x2,y2,rad,parcolor:string):integer;

  class function PrepareClear(parcolor:string):integer;

  class function PrepareDrawImage(x,y:string):integer;

  class function PrepareOrientation(deg:string):integer;



end;



implementation



{ TMyCommands }


class procedure TMyCommands.ClearCanvas(const Form:TForm; const Canvas:TCanvas; const color:
Cardinal);
```

```
begin
  Canvas.Clear(color);
  Form.Fill.Color:=color;
end;


class procedure TMyCommands.DrawSymbol(const mysymbol:integer; ppoint: TPointF; const Canvas:
TCanvas;
  const color: Cardinal);
var p1,p2:TPointF; xcenter,ycenter:Double;
begin
  Canvas.Stroke.Color:=color;
  Canvas.Stroke.Thickness:=2;


  case mysymbol of
  0:  // A
  begin
    xcenter:=ppoint.X;
    ycenter:=ppoint.Y;
    p1:=TPointF.Create(xcenter-10,ycenter);
    p2:=TPointF.Create(xcenter+10,ycenter);
    Canvas.DrawLine(p1,p2,1.0);
    p1:=TPointF.Create(xcenter,ycenter-20);
    p2:=TPointF.Create(xcenter+10,ycenter);
    Canvas.DrawLine(p1,p2,1.0);
    p1:=TPointF.Create(xcenter,ycenter-20);
    p2:=TPointF.Create(xcenter-10,ycenter);
    Canvas.DrawLine(p1,p2,1.0);
    p1:=TPointF.Create(xcenter-10,ycenter);
    p2:=TPointF.Create(xcenter-10,ycenter+20);
    Canvas.DrawLine(p1,p2,1.0);
    p1:=TPointF.Create(xcenter+10,ycenter);
    p2:=TPointF.Create(xcenter+10,ycenter+20);
    Canvas.DrawLine(p1,p2,1.0);
  end;
  1:  // B
  begin

    xcenter:=ppoint.X;
```

```
  ycenter:=ppoint.Y;

  p1:=TPointF.Create(xcenter-10,ycenter-20);
  p2:=TPointF.Create(xcenter-10,ycenter+20);

  Canvas.DrawLine(p1,p2,1.0);

  p1:=TPointF.Create(xcenter+10,ycenter-10);
  p2:=TPointF.Create(xcenter-10,ycenter-20);

  Canvas.DrawLine(p1,p2,1.0);

  p1:=TPointF.Create(xcenter-10,ycenter);
  p2:=TPointF.Create(xcenter+10,ycenter-10);

  Canvas.DrawLine(p1,p2,1.0);

  p1:=TPointF.Create(xcenter-10,ycenter);
  p2:=TPointF.Create(xcenter+10,ycenter+10);

  Canvas.DrawLine(p1,p2,1.0);

  p1:=TPointF.Create(xcenter-10,ycenter+20);
  p2:=TPointF.Create(xcenter+10,ycenter+10);

  Canvas.DrawLine(p1,p2,1.0);

end;
2:  // C
begin

  xcenter:=ppoint.X;
  ycenter:=ppoint.Y;

  p1:=TPointF.Create(xcenter-10,ycenter);
  p2:=TPointF.Create(xcenter+10,ycenter-20);

  Canvas.DrawLine(p1,p2,1.0);
```

```
  p1:=TPointF.Create(xcenter-10,ycenter);
  p2:=TPointF.Create(xcenter+10,ycenter+20);

  Canvas.DrawLine(p1,p2,1.0);

end;
3:  // D
begin

  xcenter:=ppoint.X;
  ycenter:=ppoint.Y;

  p1:=TPointF.Create(xcenter-10,ycenter-20);
  p2:=TPointF.Create(xcenter-10,ycenter+20);

  Canvas.DrawLine(p1,p2,1.0);

  p1:=TPointF.Create(xcenter-10,ycenter-20);
  p2:=TPointF.Create(xcenter+10,ycenter);

  Canvas.DrawLine(p1,p2,1.0);

  p1:=TPointF.Create(xcenter-10,ycenter+20);
  p2:=TPointF.Create(xcenter+10,ycenter);

  Canvas.DrawLine(p1,p2,1.0);

end;
4: //E
begin
xcenter:=ppoint.X;
ycenter:=ppoint.Y;

  p1:=TPointF.Create(xcenter-10,ycenter-20);
  p2:=TPointF.Create(xcenter-10,ycenter+20);

  Canvas.DrawLine(p1,p2,1.0);
```

```
  p1:=TPointF.Create(xcenter-10,ycenter-20);
  p2:=TPointF.Create(xcenter+10,ycenter-20);

  Canvas.DrawLine(p1,p2,1.0);

  p1:=TPointF.Create(xcenter-10,ycenter);
  p2:=TPointF.Create(xcenter+10,ycenter);

  Canvas.DrawLine(p1,p2,1.0);

  p1:=TPointF.Create(xcenter-10,ycenter+20);
  p2:=TPointF.Create(xcenter+10,ycenter+20);

  Canvas.DrawLine(p1,p2,1.0);

end;
5: //F
begin
xcenter:=ppoint.X;
ycenter:=ppoint.Y;

  p1:=TPointF.Create(xcenter-10,ycenter-20);
  p2:=TPointF.Create(xcenter-10,ycenter+20);

  Canvas.DrawLine(p1,p2,1.0);

  p1:=TPointF.Create(xcenter-10,ycenter-20);
  p2:=TPointF.Create(xcenter+10,ycenter-20);

  Canvas.DrawLine(p1,p2,1.0);

  p1:=TPointF.Create(xcenter-10,ycenter);
  p2:=TPointF.Create(xcenter+10,ycenter);

  Canvas.DrawLine(p1,p2,1.0);

end;
```

```
6: //G
begin
 xcenter:=ppoint.X;
 ycenter:=ppoint.Y;

  p1:=TPointF.Create(xcenter-10,ycenter-20);
  p2:=TPointF.Create(xcenter-10,ycenter+20);

  Canvas.DrawLine(p1,p2,1.0);

  p1:=TPointF.Create(xcenter-10,ycenter-20);
  p2:=TPointF.Create(xcenter+10,ycenter-20);

  Canvas.DrawLine(p1,p2,1.0);

  p1:=TPointF.Create(xcenter-10,ycenter+20);
  p2:=TPointF.Create(xcenter+10,ycenter+20);

  Canvas.DrawLine(p1,p2,1.0);

  p1:=TPointF.Create(xcenter+10,ycenter+20);
  p2:=TPointF.Create(xcenter+10,ycenter);

  Canvas.DrawLine(p1,p2,1.0);

  p1:=TPointF.Create(xcenter,ycenter);
  p2:=TPointF.Create(xcenter+10,ycenter);

  Canvas.DrawLine(p1,p2,1.0);

end;
7: //H
begin
  xcenter:=ppoint.X;
  ycenter:=ppoint.Y;

  p1:=TPointF.Create(xcenter-10,ycenter-20);
  p2:=TPointF.Create(xcenter-10,ycenter+20);
```

```
    Canvas.DrawLine(p1,p2,1.0);

    p1:=TPointF.Create(xcenter+10,ycenter-20);
    p2:=TPointF.Create(xcenter+10,ycenter+20);

    Canvas.DrawLine(p1,p2,1.0);
    p1:=TPointF.Create(xcenter-10,ycenter);
    p2:=TPointF.Create(xcenter+10,ycenter);

    Canvas.DrawLine(p1,p2,1.0);
  end;
  end;
end;

class procedure TMyCommands.DrawImage(const x, y: double; const bmp: TBitmap; const
Canvas:TCanvas);
begin
    Canvas.DrawBitmap(bmp, TRectF.Create(0, 0, bmp.Width, bmp.Height),
    TRectF.Create(0 + x, 0 + y, bmp.Width + x, bmp.Height + y), 1.0, true);
end;

class procedure TMyCommands.DrawMyEllipse(const x1_ellipse, y1_ellipse,
  x2_ellipse, y2_ellipse: Double; const Canvas: TCanvas; const color: Cardinal);
var rect:TRectF;
begin
  rect:=TRectF.Create(x1_ellipse,y1_ellipse,x2_ellipse,y2_ellipse);
  Canvas.Stroke.Color:=color;
  Canvas.Stroke.Thickness:=3;
  Canvas.Stroke.Dash:=TStrokeDash.Solid;
  Canvas.DrawEllipse(rect,1.0);
end;

class procedure TMyCommands.DrawMyLine(const p1, p2: TPointF;
  const Canvas: TCanvas; const color: Cardinal);
begin
  Canvas.Stroke.Color:=color;
  Canvas.Stroke.Thickness:=5;
```

```
  Canvas.Stroke.Dash:=TStrokeDash.Solid;
  Canvas.DrawLine(p1,p2,1.0);
end;

class procedure TMyCommands.DrawMyPixel(const ppoint: TPointF;
  const Canvas: TCanvas; const color: Cardinal);
var PixelRegion: TRectF; PixelPos: TPointF;
begin
  Canvas.Stroke.Color:=color;
  Canvas.Stroke.Thickness:=1;
  PixelPos := Canvas.AlignToPixel(ppoint);
  PixelRegion := TRectF.Create(PixelPos, 1, 1);
  Canvas.DrawRect(PixelRegion, 0, 0, AllCorners, 1);
end;

class procedure TMyCommands.DrawMyText(const x1_text, y1_text, x2_text,
  y2_text: Double; const textout: string; const fontsize: integer;
  const Canvas: TCanvas; const color: Cardinal);
begin
  Canvas.Font.Size:=fontsize;
  Canvas.Font.Style:=[TFontStyle.fsBold];
  Canvas.Fill.Color:=color;

Canvas.FillText(TRectF.Create(x1_text,y1_text,x2_text,y2_text),textout,true,1.0,[],TTextAlign.
Leading,TTextAlign.Leading);
end;

class procedure TMyCommands.FillRoundedRectangle(const x1,y1,x2,y2,
  radius: Integer; const Canvas: TCanvas; const color: Cardinal);
begin
  Canvas.Fill.Color:=color;

Canvas.FillRect(TRectF.Create(x1,y1,x2,y2),radius,radius,[TCorner.TopRight,TCorner.BottomRight
,TCorner.TopLeft,TCorner.BottomLeft],1);
end;

class function TMyCommands.PrepareSymbol(const symbol, sx, sy, parcolor: string): integer;
begin
  try
```

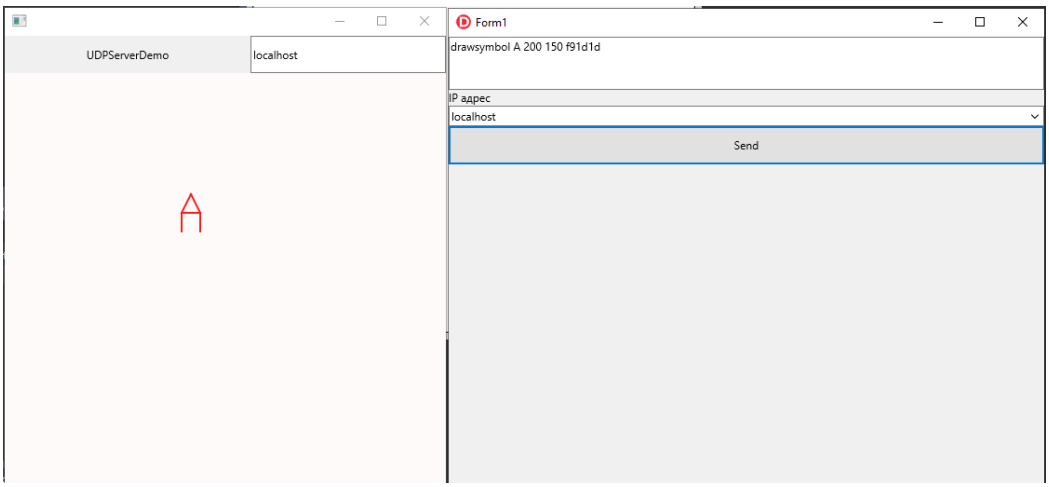
```
    Self.sx:=Double.Parse(sx);
    Self.sy:=Double.Parse(sy);
    symbolcolor:=parcolor;
    Self.symbol:=symbol;
    Result:=1;
  except on EConvertError do
  begin
    Result:=0;
  end;
  end;
end;

class function TMyCommands.PrepareClear(parcolor: string): integer;
begin
  try
    clearcolor:=parcolor;
    Result:=1;
  except on EConvertError do
  begin
    Result:=0;
  end;
  end;
end;

class function TMyCommands.PrepareDrawImage(x,y:string): integer;
begin
  try
    ximage:=Double.Parse(x);
    yimage:=Double.Parse(y);
    Result:=1;
  except on EConvertError do
  begin
    Result:=0;
  end;
  end;
end;

class function TMyCommands.PrepareEllipse(const elx1, ely1, elx2, ely2,
```

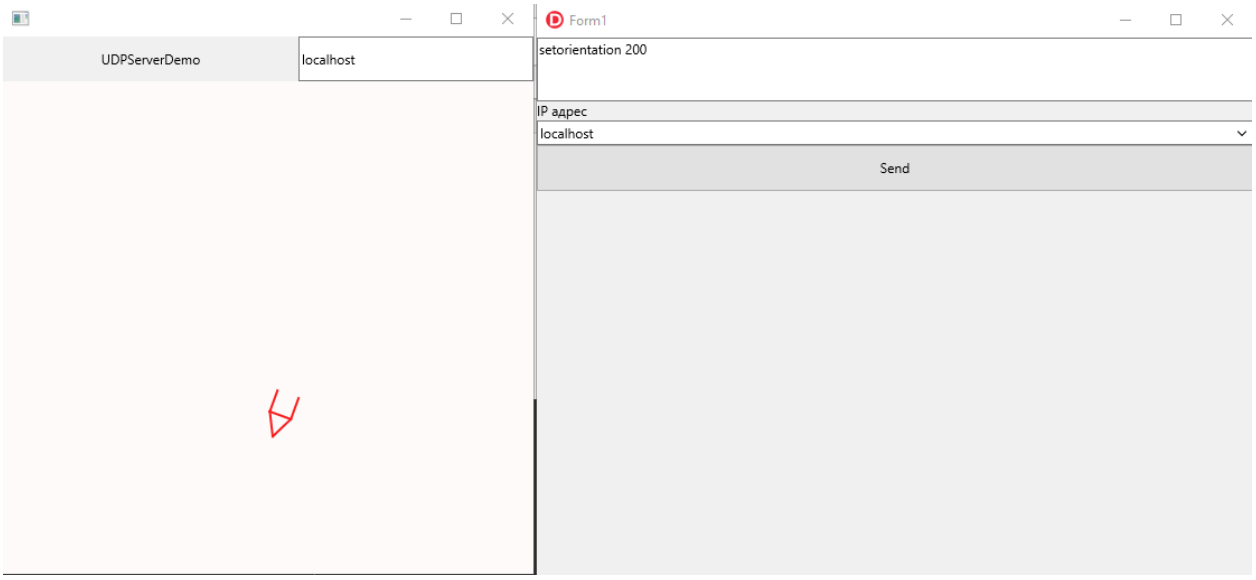
```
  parcolor: string): integer;
begin
  try
    x1_ellipse:=Double.Parse(elx1);
    y1_ellipse:=Double.Parse(ely1);
    x2_ellipse:=Double.Parse(elx2);
    y2_ellipse:=Double.Parse(ely2);
    ellipsecolor:=parcolor;
    Result:=1;
  except on EConvertError do
  begin
    Result:=0;
  end;
  end;
end;

class function TMyCommands.PrepareFillRoundedRectangle(const x1, y1, x2, y2,
  rad, parcolor: string): integer;
begin
  try
    Self.x1:=Integer.Parse(x1);
    Self.y1:=Integer.Parse(y1);
    Self.x2:=Integer.Parse(x2);
    Self.y2:=Integer.Parse(y2);
    fillroundedrectanglecolor:=parcolor;
    radius:=Integer.Parse(rad);
    Result:=1;
  except on EConvertError do
  begin
    Result:=0;
  end;
  end;
end;

class function TMyCommands.PrepareLine(const parx1, pary1, parx2,
  pary2, parcolor : string): integer;
begin
  try
```

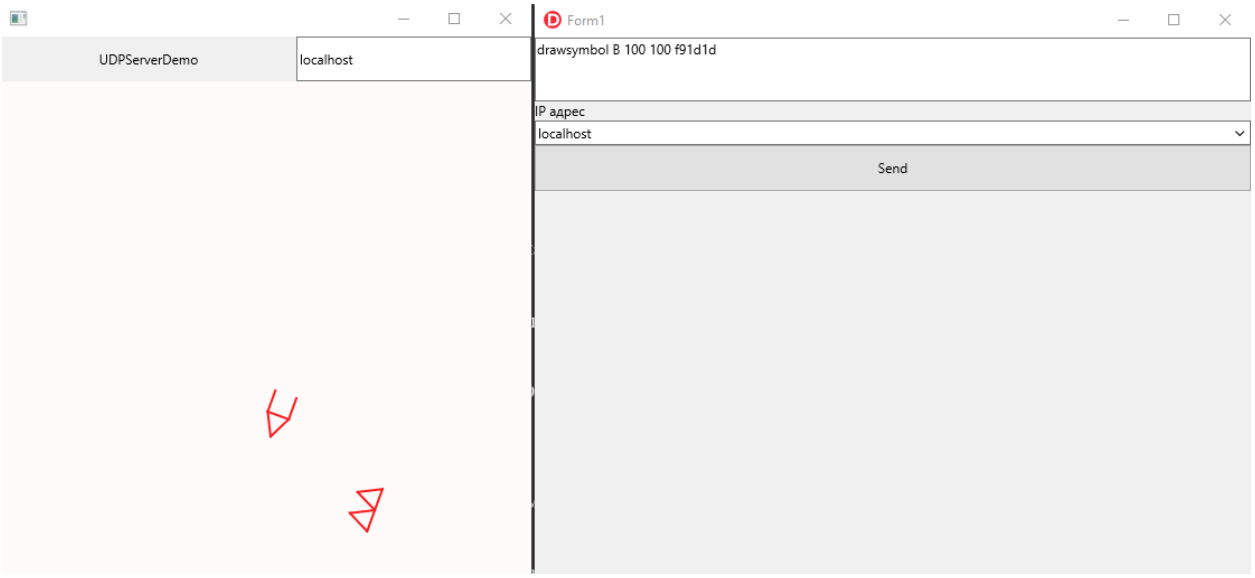
```
    p1.X:=Double.Parse(parx1);
    p1.Y:=Double.Parse(pary1);
    p2.X:=Double.Parse(parx2);
    p2.Y:=Double.Parse(pary2);
    linecolor:=parcolor;
    Result:=1;
  except on EConvertError do
  begin
    ShowMessage('Неверно введены координаты линии!!!');
    Result:=0;
  end;
  end;
end;


class function TMyCommands.PrepareOrientation(deg: string): integer;
begin
  try
    Self.degrees:=Integer.Parse(deg);
    Result:=1;
  except on EConvertError do
  begin
    ShowMessage('Неверный угол!!!');
    Result:=0;
  end;
  end;
end;

class function TMyCommands.PreparePixel(const x1, y1,
  parcolor: string): integer;
begin
  try
    ppoint.X:=Double.Parse(x1);
    ppoint.Y:=Double.Parse(y1);
    pixelcolor:=parcolor;
    Result:=1;
  except on EConvertError do
  begin
```

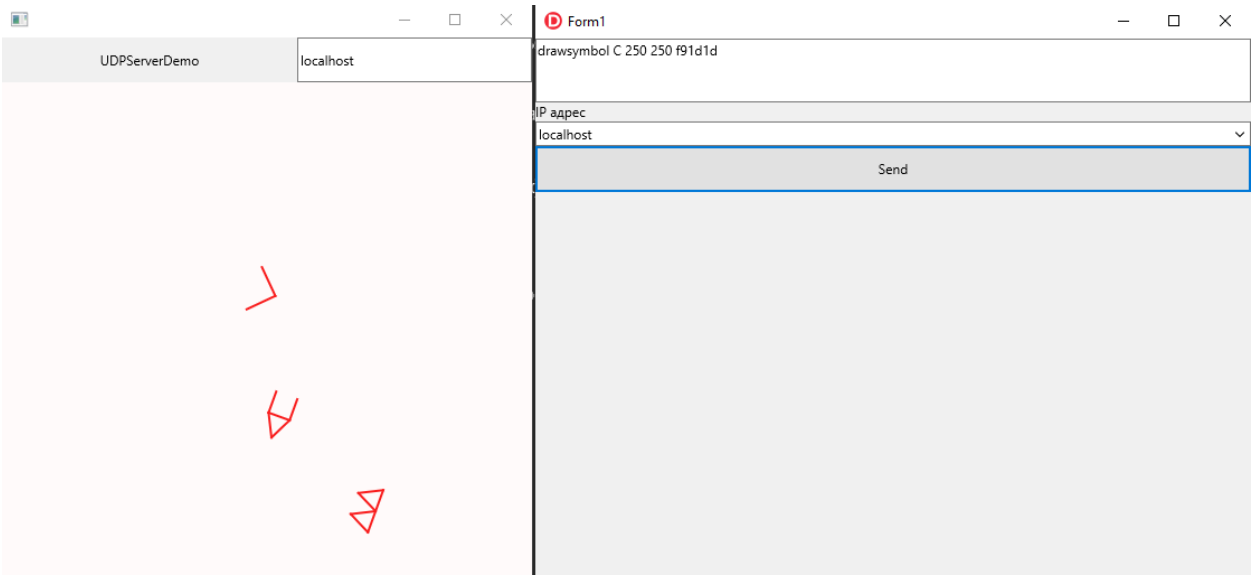
```
    ShowMessage('Неверно введены координаты пиксела!!!');
    Result:=0;
  end;
  end;
end;

class function TMyCommands.PrepareText(const tx1, ty1, tx2, ty2, text,
  parcolor: string): integer;
begin
  try
    x1_text:=Double.Parse(tx1);
    y1_text:=Double.Parse(ty1);
    x2_text:=Double.Parse(tx2);
    y2_text:=Double.Parse(ty2);
    textcolor:=parcolor;
    textout:=text;
    Result:=1;
  except on EConvertError do
  begin
    Result:=0;
  end;
  end;
end;

end.
```

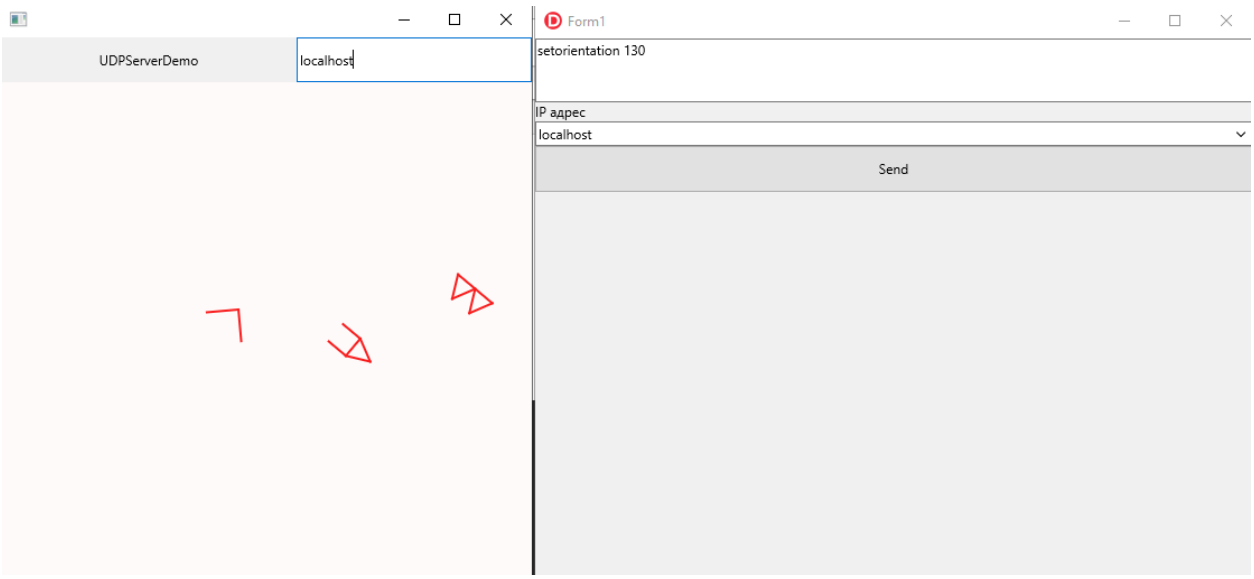
Результат роботи:




UDPServerDemo
localhost
Form1
setorientation 200
IP адрес
localhost
Send


UDPServerDemo
localhost
Form1
drawsymbol B 100 100 f91d1d
IP адрес
localhost
Send


UDPServerDemo
localhost
Form1
drawsymbol C 250 250 f91d1d
IP адрес
localhost
Send



Висновок: удосконалив програми емулятора дисплейного модуля і клієнта.