

5. Grafikus megjelenítés Delphi programokban

1. Pattogó labda, vezérlővel [Labda](#)
2. Animáció vezérlőkkel [Figura](#)
3. Téglakiütés labdával (játék) [Fallabda](#)
4. A **Chart** vezérlő használatának alapja [Chart](#)
5. Bezier görbék és poligonok dinamikus rajzolása [Amoba](#)
6. Leképezési módok használata [MapMode](#)
7. Pattogó pöttyös labda [PLabda](#)
8. Pattogó pöttyös labdák [Plabdak](#)
9. Grafikus lista [Glist](#)
10. Összetett példa, nyomtatással [Koronkor](#)
11. Ferde szövegek megjelenítése [FerdeÍrás](#)



Készítsünk alkalmazást, melynek ablakában egy labda pattog! A labda mozgását az időzítő vezérelje! (*Labda*)

A feladatot legegyszerűbben úgy oldhatjuk meg, hogy az ablakban egy *Shape* vezérlő által megtestesített labdát mozgatunk. Ahhoz, hogy a programunk eseményvezérelt animációra legyen képes, az időzítő (*Timer*) komponenst kell használnunk. A *Timer* | *Timer* eseménykezelő eljárásban intézkedhetünk a labda mozgásáról. A formon a tervezési idején elhelyeztünk egy címkét (*Label*) nagy kék betűkkel kiírt "Vezérlőgrafika" felirattal. A labdát megtestesítő *Shape* komponens neve *Labda*, mely tulajdonságait form létrehozásakor állítjuk be. A labdát a „*Bring to Front*” menüponttal a szöveg elé tettük.

A program ablakának osztálya:

```
TForm1 = class(TForm)
    Labda: TShape;
    Timer1: TTimer;
    Label1: TLabel;
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormResize(Sender: TObject);
private
    dx, dy : integer;
end;
```

A form objektum létrehozásakor meghívódó eseménykezelő eljárásban gondoskodunk a *Labda* jellemzőinek beállításáról:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    dx:=10;
    dy:=10;
    labda.Shape:=stCircle;
    labda.Pen.color:=clred;
    labda.Brush.color:=clred;
end;
```

A labda mozgatásához definiáltunk egy x- és egy y-irányú elmozdulást (*dx*, *dy*), amelyet időegységenként hozzáadunk a labda aktuális pozíciójához. A labda mozgatásáról az időzítő által kiváltott események kezelésekor gondoskodunk. Megvizsgáljuk, hogy a labda a falnak ütközött-e, és ha igen, akkor alkalmasan megfordítjuk a megfelelő sebességkomponenst.

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    if (labda.left+labda.width+dx>form1.clientwidth) or
        (labda.left+dx<0) then dx:=-dx;
    if (labda.top+labda.height+dy>form1.clientheight) or
        (labda.top+dy<0) then dy:=-dy;
    labda.top:=labda.top + dy;
    labda.left:=labda.left + dx;
end;
```

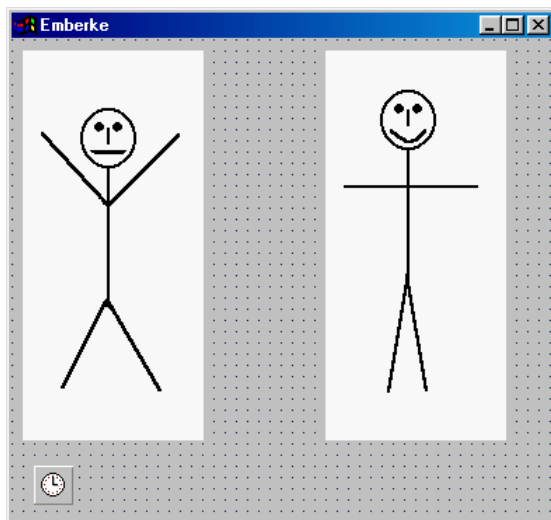
Annyit teszünk még a programban, hogy az ablak átméretezésekor középre helyezzük a labdát és a szöveget:

```
procedure TForm1.FormResize(Sender: TObject);
begin
    labda.left:=(form1.clientwidth-labda.width) div 2;
    labda.top:=(form1.clientheight-labda.height) div 2;
    label1.left:=(form1.clientwidth-label1.width) div 2;
    label1.top:=(form1.clientheight-label1.height) div 2;
end;
```





Látványosabb animációs effektusokat érhetünk el, ha a grafikus vezérlők közül az képet (*Image*) használjuk. Ilyenkor különböző képek cseréjével mozgást is szimulálhatunk. Mindenki emlékszik arra, hogy gyermekként összehajtott papírra rajzolt két pálcikaembert, az egyiket terpeszállásban, a másikat nem. Majd egy ceruzát rángatva hol az egyik, hol pedig a másik ábrát megmutatva rajzfilmet készített.



Készítsünk két rajzot az *Image1* és *Image2* vezérlőkbe (*IMAG1.BMP*, *IMAG2.BMP*) és programozzuk a *Timer1* vezérlőt a képek cseréjére! Mindkét kép *AutoSize* tulajdonságát igazra állítva, a képek méretét azonosra tettük.

A program ablakát a *TForm1* osztály írja le:

```
type
  TForm1 = class(TForm)
    Image1: TImage;
    Image2: TImage;
    Timer1: TTimer;
    procedure FormCreate(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  end;
```

A form létrehozásakor elhelyezzük a képeket, melyek közül az első látható lesz, míg a másik nem:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // a rajzok azonos pozícióba kerülnek
  Image1.Left:=0;
  Image2.Left:=0;
  Image1.Top:=0;
  Image2.Top:=0;
  // az egyik rajz látható a másik nem
  Image1.Visible:=true;
  Image2.Visible:=false;
  // az ablak mérete a rajzokhoz igazodik
  form1.ClientWidth:=Image1.Width;
  form1.ClientHeight:=Image1.Height;
end;
```

Ezek után az időzítő eseményeit kezelő eljárás csak a képek láthatóságát váltogatja:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Image1.Visible:=not Image1.Visible;
  Image2.Visible:=not Image2.Visible;
end;
```



Készítsünk játékprogramot, melyben az ablak alján elhelyezett, a billentyűzet segítségével mozgatható ütővel egy labdát pattogtatunk. A labdával az ablak felső részében elhelyezett téglákat kiütjük! (*Fallabda*)

A program a [Labda](#) program továbbfejlesztése. A form osztályából látható, hogy kezeljük a gombnyomás eseményt is.

```
type
  TForm1 = class(TForm)
    Labda: TShape;
    Timer1: TTimer;
    Uto: TShape;
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
  private
  public
end;
```

A form létrehozásakor dinamikus (*Shape* típusú) *Teglak* tömbben helyezzük el a téglákat.

```
procedure TForm1.FormCreate(Sender: TObject);
var i,j:integer;
begin
  dx:=10;
  dy:=10;
  // a labda objektum formájának és színének beállítása
  labda.Shape:=stCircle;
  labda.Pen.color:=clred;
  labda.Brush.color:=clred;
  // A téglafal felépítése
  for i:=1 to sorok do
    for j:=1 to oszlopok do
      begin
        Teglak[i,j]:=TShape.Create(Form1);
        Teglak[i,j].Brush.Color:=clYellow;
        Teglak[i,j].Brush.Style:=bsSolid;
        Teglak[i,j].Brush.Color:=clYellow;
        Teglak[i,j].Parent:=Form1;
        Teglak[i,j].Visible:=True;
      end;
    Form1.Align:=alNone;
    Teglameretezes(Form1,Labda,Uto);
  end;
```

A *Unit2* modul tartalmazza a *Teglameretezes* eljárást, amely a téglákat és az ütőt a form méreteihez igazítja. Ugyanezt hívjuk a form átméretezésekor.

```
procedure TForm1.FormResize(Sender: TObject);
begin
  // átméretezéskor a labda és a felirat középre kerül
  Teglameretezes(Form1,Labda,Uto);
end;
```

A labdamozgatás és az ütközések ellenőrzése a **Timer** eseményben történik. Az ütközést a *Unit2 Rect_* függvénye ellenőrzi két objektum között.

```

procedure TForm1.Timer1Timer(Sender: TObject);
  var i,j:integer;
      nincs_vege:boolean;
begin
  // az időzítő mozgatja a labdát
  // ütközéskor irányváltás a határokon
  if (labda.left+labda.width+dx>form1.clientwidth) or
      (labda.left+dx<0) then dx:=-dx;
  if (labda.top+dy<0) then dy:=-dy;
  if (labda.top+labda.height+dy>form1.clientheight) then
    begin
      Timer1.Enabled:=false;
      MessageDlg('Sajnos Ön vesztett',mtCustom, [mbOK],0);
      Application.Terminate;
    end;
  // a labda az ütővel találkozik
  if (Rect_(Labda,Uto)) then
    begin
      dy:=-dy;
    end;
  // a labda és a téglák találkozásának ellenőrzése
  for i:=1 to sorok do
    for j:=1 to oszlopok do
      if (Teglak[i,j].Visible) then
        begin
          if (Rect_(Labda,Teglak[i,j])) then
            begin
              Teglak[i,j].Visible:=False;
              dy:=-dy;
              break;
            end;
          end;
        end;
  nincs_vege:=false;
  // vége van-e a játéknak
  for i:=1 to sorok do
    for j:=1 to oszlopok do
      begin
        nincs_vege:=nincs_vege or Teglak[i,j].Visible;
        if nincs_vege then break;
      end;
  if not(nincs_vege) then
    begin
      Timer1.Enabled:=false;
      MessageDlg('Gratulálunk Ön nyert',mtCustom, [mbOK],0);
      Application.Terminate;
    end;
  labda.top:=labda.top + dy;
  labda.left:=labda.left + dx;
end;

```

A lenyomott nyíl gombok mozgatják az ütőt

```

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  // az ütő mozgatása
  if (key=VK_Left) and (Uto.left>0) then
    begin
      Uto.left:=Uto.left-2*abs(dx);
    end;
  if (key=VK_Right) and ((Uto.left+uto.width)<ClientWidth) then
    begin
      Uto.left:=Uto.left+2*abs(dx);
    end;
end;

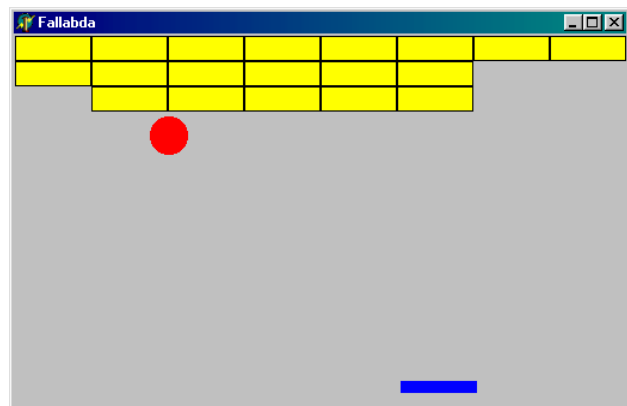
```

A Unit2 Teglameretezes eljárása:

```
procedure Teglameretezes(x:TForm;y,z:TShape);
// A téglák méreteinek meghatározása
var i,j,szel,mag,xpos,ypos:integer;
begin
    szel:=x.ClientWidth div 8;
    mag:=szel div 3;
    dx:=mag div 3;
    dy:=mag div 3;
    Teglak[1,1].Left:=1;
    Teglak[1,1].Height:=mag;
    Teglak[1,1].Width:=szel;
    for i:=1 to sorok do
        begin
            ypos:=1+(i-1)*mag;
            for j:=1 to oszlopok do
                begin
                    xpos:=1+(j-1)*szel;
                    Teglak[i,j].Left:=xpos;
                    Teglak[i,j].Top:=ypos;
                    Teglak[i,j].Height:=mag;
                    Teglak[i,j].Width:=szel;
                end;
            end;
        end;
    y.Width:=szel div 2;
    y.Height:=szel div 2;
    y.left:=(x.clientwidth-y.width) div 2;
    y.top:=(x.clientheight-y.height) div 2;
    z.Width:=szel;
    z.Height:=mag div 2;
    z.Top:=x.ClientHeight-mag;
    z.left:=y.left;
end;
```

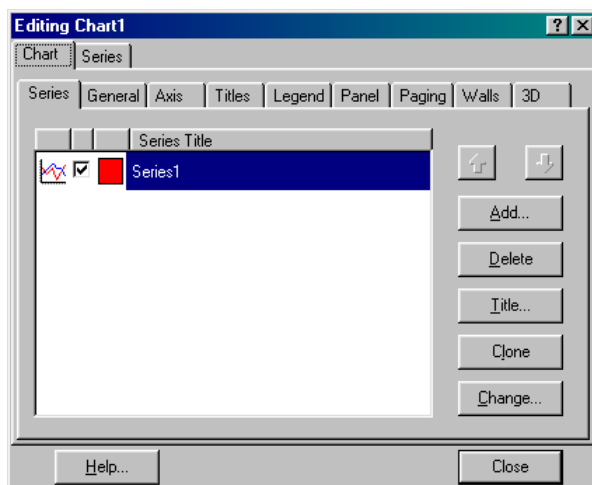
A *Rect_* függvény igaz értékkel tér vissza, ha két paramétere által meghatározott komponens összeér. A számításhoz használjuk a *Windows* modul *PtInRect* függvényét.

```
function Rect_(a:TShape;b:TShape):boolean;
// Az a, b Shape összeér-e?
var pont:TPoint;
    rect:TRect;
begin
    Rect_:=False;
    pont.x:=a.left;
    pont.y:=a.top;
    rect.left:=b.left;
    rect.top:=b.top;
    rect.right:=b.left+b.width;
    rect.bottom:=b.top+b.height;
    if (PtInRect(rect,pont)) then
        Rect_:=True;
    pont.x:=a.left;
    pont.y:=a.top+a.height;
    if (PtInRect(rect,pont)) then
        Rect_:=True;
    pont.x:=a.left+a.width;
    pont.y:=a.top;
    if (PtInRect(rect,pont)) then
        Rect_:=True;
    pont.x:=a.left+a.width;
    pont.y:=a.top+a.height;
    if (PtInRect(rect,pont)) then
        Rect_:=True;
end;
```





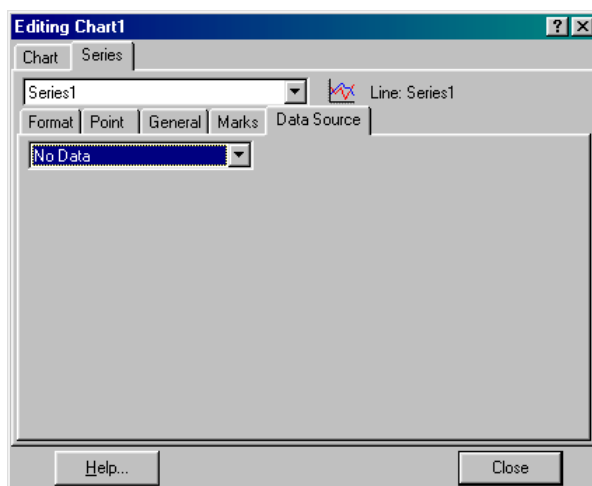
Helyezzük a formra a **Chart** vezérlőt és állítsuk az **Align** tulajdonságát *alClient*-re! A vezérlő felbukkanó menüjének „*Edit Chart...*” menüpontjával adjuk hozzá egy sorozatot!



Ezzel a *TForm1* osztályba bekerül a **TLineSeries** típusú *Series1*.

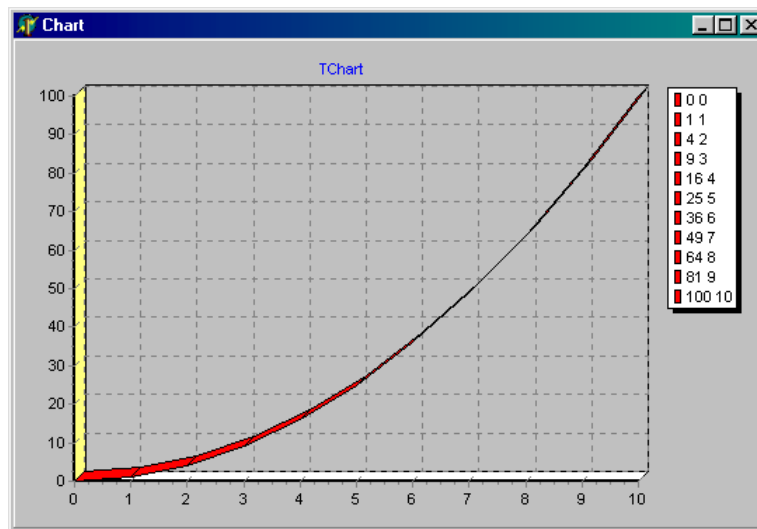
```
type
  TForm1 = class(TForm)
    Chart1: TChart;
    Series1: TLineSeries;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

A **Series** lapon a „*Data Source*” lehetőségei közül a „*No Data*” lehetőséget választva a *Series1* üres lesz.



A *Series1* feltöltése és **Chart**-hoz való kapcsolása programból a form létrehozásakor történik.

```
procedure TForm1.FormCreate(Sender: TObject);  
var i:integer;  
begin  
  For i := 0 to 10 do  
    Series1.AddXY(i,i*i,inttostr(i),clRed);  
  Chart1.SeriesList[0]:=Series1;  
end;
```





Írjunk programot, amely megadott pontokat poligon csúcspontként és Bezier görbe tartópontokként kezeli! Legyen lehetőség a pontok mozgatására is! (*Amoba*)

Mivel a pontszám $4+3*k$ kell legyen, ennek megfelelően definiáljuk a *pontszám* konstanst. A *Pontok* tömb *pontszám* darab **TPoint** típusú rekordot tartalmaz, és a tartópontokat tárolja. Mozgatáshoz szükségünk lesz a kiválasztott pont azonosítására szolgáló *kiválasztott* egészre.

```
const pontszám=19;
var
  //A pontok tömbje
  Pontok : Array [0..pontszám-1] of TPoint;
  //A kiválasztott pont mozgatható
  kiválasztott:integer;
```

A form létrehozásakor meghatározzuk az alapábra pontjainak helyét. Mivel görbéink zártak, az első és az utolsó pont azonos. Inicializáljuk a *kiválasztott* értékét is (-1) jelölve, hogy nincs kiválasztott pont.

```
procedure TForm1.FormCreate(Sender: TObject);
var xk, yk, r, i:integer;
begin
  // Létrehozzuk az alapábra pontjait
  xk:=ClientWidth div 2;
  yk:=ClientHeight div 2;
  r:=ClientHeight div 3;
  for i:=0 to pontszám-2 do
    begin
      Pontok[i].x:=xk+Trunc(r*cos(i/(pontszám-1)*6.28));
      Pontok[i].y:=yk+Trunc(r*sin(i/(pontszám-1)*6.28));
    end;
  //Az első és utolsó pont azonos
  Pontok[pontszám-1].x:=Pontok[0].x;
  Pontok[pontszám-1].y:=Pontok[0].y;
  kiválasztott:=-1;
end;
```

A poligon és a görbe razolása az **OnPaint** eseményben történik.

```
procedure TForm1.FormPaint(Sender: TObject);
begin
  // Kifestéskor a tartópontok poligonja fekete
  Canvas.Pen.Color:=clBlack;
  Canvas.Polyline(Pontok);
  // Kifestéskor a Bezier görbék pirosak
  Canvas.Pen.Color:=clRed;
  Canvas.PolyBezier(Pontok);
end;
```

Egér gomb lenyomásakor kikeressük a legközelebbi pontot, felengedéskor nincs kiválasztott pont.

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  //Gombnyomással pontot választunk
  kiválasztott:=Pontkeres(x,y);
end;

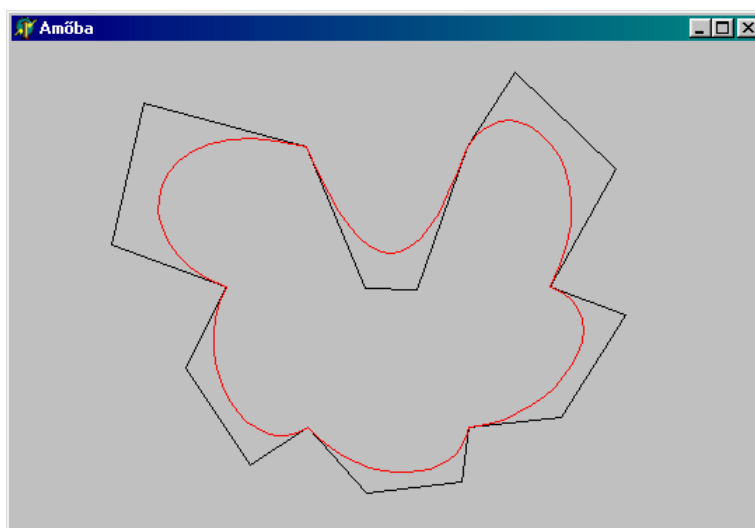
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  // Ha elengedjük az egér gombot nincs kiválasztott pont
  kiválasztott:=-1;
end;
```

A *Pontkeres* függvény paraméterei form-koordináták, visszatérési értéke a kiválasztott pont.

```
function Pontkeres(x,y:integer):integer;  
// A függvény kikeresi az x,y ponthoz legközelebbi sarokpontot  
var tav,tav1,tav2,tav12:real;  
    i:integer;  
begin  
    tav:=10000;  
    Pontkeres:=-1;  
    for i:=0 to pontszam-1 do  
        begin  
            tav1:=(Pontok[i].x-x);  
            tav2:=(Pontok[i].y-y);  
            tav1:=tav1*tav1;  
            tav2:=tav2*tav2;  
            tav12:=tav1+tav2;  
            if tav12<tav then  
                begin  
                    tav:=tav12;  
                    Pontkeres:=i;  
                end;  
            end;  
        end;  
    end;  
end;
```

Lenyomott gombbal történő egérmozgatásakor a kiválasztott pont koordinátái változnak és újrifestjük a formot.

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,  
    Y: Integer);  
begin  
    // Ha van kiválasztott pont (lenyomott egérgomb), akkor az mozog  
    if (kivalasztott>-1) then  
        begin  
            if (kivalasztott<>0) and (kivalasztott<>pontszam-1) then  
                begin  
                    Pontok[kivalasztott].x:=x;  
                    Pontok[kivalasztott].y:=y;  
                end  
            else  
                begin  
                    //Az első és utolsó pont azonos  
                    Pontok[0].x:=x;  
                    Pontok[0].y:=y;  
                    Pontok[pontszam-1].x:=x;  
                    Pontok[pontszam-1].y:=y;  
                end;  
            Refresh;  
        end;  
    end;  
end;
```

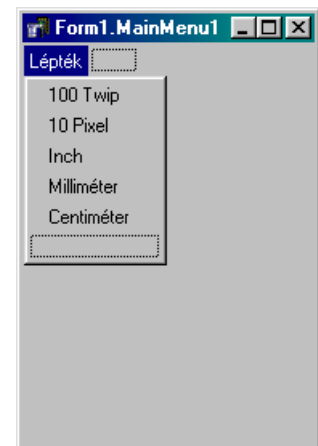




A form menütervezőjében készítsük el az alábbi menüt!

Az egységeket reprezentáló háló rajzolásához a *lepes* típusos konstanst, a leképezési mód azonosítására az *mm* típusos konstanst használjuk. Mivel a kirajzolásnál tudnunk kell, hogy merre mutat az *y*-tengely ennek azonosítására a *lefele* típusos konstans szolgál.

```
const
  // lépésköz a kirajzoláshoz
  lepes:integer=10;
  // leképezési mód
  mm:integer=MM_TEXT;
  // merre mutat az y tengely
  lefele:integer=1;
```



A form létrehozásánál beállítjuk az alapértékeket.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // alapbeállítás a pixeles
  N10Pixel1.checked:=true;
  // lépésköz a kirajzoláshoz
  lepes:=10;
  // leképezési mód
  mm:=MM_TEXT;
  // az y tengely lefelé mutat
  lefele:=1;
end;
```

Az egyes menüpontokban a menüválasztásnak megfelelően beállítjuk az értékeket és újrarajzoltatjuk a formot.

```
procedure TForm1.N100Twip1Click(Sender: TObject);
begin
  // menübeállítás
  menutorol;
  TMenuItem(Sender).checked:=true;
  // lépésköz a kirajzoláshoz
  lepes:=100;
  // leképezési mód
  mm:=MM_TWIPS;
  // az y tengely felfelé mutat
  lefele:=-1;
  // újrarajzolás
  Refresh;
end;
```

```
procedure TForm1.N10Pixel1Click(Sender: TObject);
begin
  // menübeállítás
  menutorol;
  TMenuItem(Sender).checked:=true;
  // lépésköz a kirajzoláshoz
  lepes:=10;
  // leképezési mód
  mm:=MM_TEXT;
  // az y tengely lefelé mutat
  lefele:=1;
  // újrarajzolás
  Refresh;
end;
```

```

procedure TForm1.Inch1Click(Sender: TObject);
begin
    // menübeállítás
    menutorol;
    TMenuItem(Sender).checked:=true;
    // lépésköz a kirajzoláshoz
    lepes:=100;
    // leképezési mód
    mm:=MM_LOENGLISH;
    // az y tengely felfelé mutat
    lefele:=-1;
    // újrarajzolás
    Refresh;
end;

procedure TForm1.Millimter1Click(Sender: TObject);
begin
    // menübeállítás
    menutorol;
    TMenuItem(Sender).checked:=true;
    // lépésköz a kirajzoláshoz
    lepes:=10;
    // leképezési mód
    mm:=MM_LOMETRIC;
    // az y tengely felfelé mutat
    lefele:=-1;
    // újrarajzolás
    Refresh;
end;

procedure TForm1.Centimter1Click(Sender: TObject);
begin
    // menübeállítás
    menutorol;
    TMenuItem(Sender).checked:=true;
    // lépésköz a kirajzoláshoz
    lepes:=100;
    // leképezési mód
    mm:=MM_LOMETRIC;
    // az y tengely felfelé mutat
    lefele:=-1;
    // újrarajzolás
    Refresh;
end;

```

A kiválasztott menüelem kijelöléséhez és a többi alapállapotba hozásához a *menutorol* eljárást használjuk.

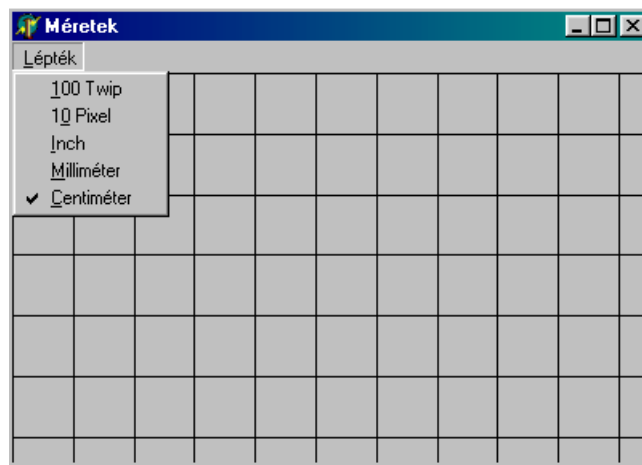
```

procedure menutorol;
begin
    // egyik menü sincs kiválasztva
    with Form1 do
        begin
            N100Twip1.checked:=false;
            N10Pixel1.checked:=false;
            Inch1.checked:=false;
            Millimter1.checked:=false;
            Centimter1.checked:=false;
        end;
end;

```

A kirajzolás a beállításoknak megfelelően az **OnPaint** esemény kezelőjében történik. A rács rajzolása előtt a **SetMapMode** függvényt használjuk a leképezési mód beállítására.

```
procedure TForm1.FormPaint(Sender: TObject);
var x,y:integer;
    oldmm:integer;
    sarok:TPoint;
begin
    // leképezési mód beállítása
    oldmm:=SetMapMode(Canvas.Handle,mm);
    sarok.x:=ClientWidth;
    sarok.y:=ClientHeight;
    // a bal alsó sarok képernyőpont logikai koordinátái
    DPtoLP(Canvas.Handle,sarok,1);
    // függőleges vonalak rajzolása
    x:=0;
    while x<sarok.x do
    begin
        Canvas.Moveto(x,0);
        Canvas.Lineto(x,sarok.y);
        x:=x+abs(lepes);
    end;
    // vízszintes vonalak rajzolása
    y:=0;
    while abs(y)<abs(sarok.y) do
    begin
        Canvas.Moveto(0,y);
        Canvas.Lineto(sarok.x,y);
        y:=y+lefele*lepes;
    end;
    // leképezési mód visszaállítása
    SetMapMode(Canvas.Handle,oldmm);
end;
```





A programot a [Labda](#) programhoz hasonlóan készíthetjük el, azonban a pöttyös labda megjelenítéséhez a *Labdau* modulban saját labda objektumot készítünk. A labda pozíciójának tárolására az *FPos* (*TPoint* típusú) változót használjuk, és a megfelelő tulajdonság a *Pos*, melynek beállítását és lekérdezését a *SetPos* és *GetPos* metódusok végzik. A labda megjelenítéséhez a *Kirajz* metódust készítjük el. A *TLabda* osztály deklarációja:

```
type
  TLabda = class(TObject)
    // a labda objektum a TObject leszármazottja
  private
    // a pozíció belső tárolása
    FPos:TPoint;
    // a pozíció olvasása és írása
    procedure SetPos(Pos:TPoint);
    function GetPos:TPoint;
  public
    //a labdát megjelenítő bitkép
    Labda:TBitmap;
    //A szülő
    Parent:TForm;
    //a labda sebessége
    dx,dy:integer;
    // a labda mérete
    meret:integer;
    constructor Create(Par:TForm; szin:TColor);
  published
    //a labda pozíciója
    Property Pos:TPoint read GetPos write SetPos;
    procedure Kirajz;
end;
```

A tulajdonságok írására és olvasására szolgáló metódusok:

```
procedure TLabda.SetPos(Pos:TPoint);
begin
    // ha a pozíció változik a régi pozícióba a labdát
    Kirajz;
    // a pozíció változása után az új pozícióba a labdát
    FPos.x:=Pos.x;
    FPos.y:=Pos.y;
    Kirajz;
end;

function TLabda.GetPos:TPoint;
begin
    // a pozíció olvasása
    result:=FPos;
end;
```

A labda kirajzolása:

```
procedure TLabda.Kirajz;
begin
    // a képernyőre helyezi a labdát inverz módon
    Parent.Canvas.CopyMode:=cmSrcInvert;
    Parent.Canvas.Draw(pos.x,pos.y,Labda);
end;
```

A labda létrehozásakor a **TBitmap** osztályt használjuk:

```
constructor TLabda.Create(Par:TForm; szin:TColor);
var seged:TBitmap;
begin
    // a labda mérete
    meret:=60;
    // a mozgás sebessége
    dx:=10;
    dy:=10;
    // a szülő megadása
    Parent:=Par;
    //megrajzoljuk a labda bittérképet (seged)
    seged:=TBitmap.Create;
    seged.Height:=meret;
    seged.Width:=meret;
    with seged.Canvas do
    begin
        Brush.Color:=szin;
        Pen.Color:=szin;
        Ellipse(0,0,meret,meret);
        Brush.Color:=clWhite;
        Pen.Color:=clWhite;
        Ellipse(1*meret div 12,2*meret div 12,4*meret div 12,5*meret div 12);
        Ellipse(7*meret div 12,2*meret div 12,10*meret div 12,5*meret div 12);
        Ellipse(2*meret div 12,10*meret div 12,5*meret div 12,7*meret div 12);
        Ellipse(8*meret div 12,10*meret div 12,11*meret div 12,7*meret div 12);
    end;
    //a megrajzolt bittérkép (seged) inverze lesz a labda
    Labda:=TBitmap.Create;
    Labda.Height:=meret;
    Labda.Width:=meret;
    with Labda.Canvas do
    begin
        Brush.Color:=Parent.Color;
        Pen.Color:=Parent.Color;
        Ellipse(0,0,meret,meret);
        CopyMode:=cmSrcInvert;
        Draw(0,0,seged);
    end;
    seged.free;
end;
```

Ezek után az ablakmodulban már könnyen elkészíthetjük a labdát.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Labda:=TLabda.Create(Form1,clRed);
end;
```

Kilépéskor lebontjuk a labdát:

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
    Labda.Free;
end;
```

A [Labda](#) programhoz hasonlóan a *Timer1* mozgatja a labdát.

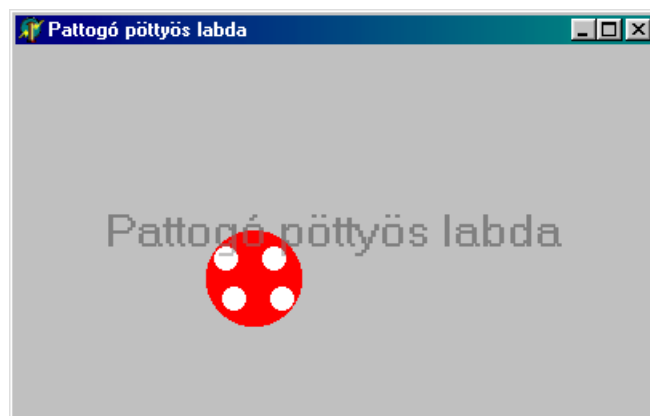
```
procedure TForm1.Timer1Timer(Sender: TObject);
var segedpos:TPoint;
begin
    // az időzítő mozgatja a labdát ütközéskor irányváltás a határokon
    if (Labda.Pos.x+Labda.meret+Labda.dx>form1.clientwidth) or
        (Labda.Pos.x+Labda.dx<0) then Labda.dx:=-Labda.dx;
    if (Labda.Pos.y+Labda.meret+Labda.dy>form1.clientheight) or
        (Labda.Pos.y+Labda.dy<0) then Labda.dy:=-Labda.dy;
    segedpos.x:=Labda.Pos.x+Labda.dx;
    segedpos.y:=Labda.Pos.y+Labda.dy;
    Labda.Pos:=segedpos;
end;
```

Ha átméretezzük a formot, akkor a labda középre kerül.

```
procedure TForm1.FormResize(Sender: TObject);
var segedpos:TPoint;
begin
  // átméretezéskor a labda középre kerül
  segedpos.x:=(form1.clientwidth-Labda.meret) div 2;
  segedpos.y:=(form1.clientheight-Labda.meret) div 2;
  Labda.Pos:=segedpos;
  Refresh;
end;
```

Az **OnPaint** még szöveget is kiír.

```
procedure TForm1.FormPaint(Sender: TObject);
var kiir:string;
begin
  // a képernyő közepére szöveget írunk
  Canvas.Brush.Color:=Form1.Color;
  Canvas.Font.Color:=clDkGray;
  Canvas.Font.Size:=20;
  Canvas.Font.Style:=[fsBold];
  Canvas.CopyMode:=cmSrcCopy;
  kiir:='Pattogó pöttyös labda';
  Canvas.TextOut((ClientWidth-Canvas.TextWidth(kiir)) div 2,
                 (ClientHeight-Canvas.TextHeight(kiir)) div 2,kiir);
  // és alappozícióba helyezzük a labdát
  Labda.Kirajz;
end;
```





A programot a [PLabda](#) alapján készíthetjük el (azonos *Labdau* modul).

A főprogramban a *labdaszam* konstans határozza meg a labdák számát. A pattogó labdákat a *TLabda* típusú tömb tárolja. Kihaszználjuk, hogy a labda színét a *TLabda* konstruktorában állíthatjuk be, ehhez a *szin* típusos konstans tömböt használjuk.

```
const labdaszam=5;
var
  Labda:array [1..labdaszam] of TLabda;
const
  szin:array [1..labdaszam] of TColor=(clRed, clBlue, clGreen,clYellow,clPurple);
```

A form létrehozásakor és törlésekor objektumok egy tömbjét hozzuk létre, illetve szabadítjuk fel.

```
procedure TForm1.FormCreate(Sender: TObject);
var i:integer;
begin
  for i:=1 to labdaszam do
    begin
      // létrehozzuk a labdákat és elszórjuk a képernyőn
      Labda[i]:=TLabda.Create(Form1,szin[i]);
      // a labdák kezdősebességet kapnak
      Labda[i].dx:=1+i;
      Labda[i].dy:=1+i;
    end;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var i:integer;
begin
  // a labdák törlése
  for i:=1 to labdaszam do
    Labda[i].Free;
end;
```

A *Unit2 Rect_* függvénye vizsgálja, hogy a két labda ütközik-e.

```
function Rect_(a:TLabda;b:TLabda):boolean;
// Az a, b Labda összeér-e?
var pont1,pont2:TPoint;
    rect:TRect;
begin
  Rect_:=False;
  pont1.x:=a.pos.x+a.dx +a.meret div 2;
  pont1.y:=a.pos.y+a.dy +a.meret div 2;
  pont2.x:=b.pos.x+b.dx +b.meret div 2;
  pont2.y:=b.pos.y+b.dy +b.meret div 2;
  if (pont1.x-pont2.x)*(pont1.x-pont2.x)+
    (pont1.y-pont2.y)*(pont1.y-pont2.y)<
    (a.meret+b.meret)*(a.meret+b.meret) div 4 then
    Rect_:=True;
end;
```

Ebben a programban is - a [PLabda](#)-hoz hasonlóan - egyetlen időzítő mozgatja a labdákat, azonban az egymással való ütközést is figyelni kell.

```

procedure TForm1.Timer1Timer(Sender: TObject);
var segedpos:TPoint;
    i,j:integer;
    v1,v2:integer;
begin
    // az időzítő mozgatja a labdákat
    for i:=1 to labdaszam-1 do
        for j:=i+1 to labdaszam do
            if Rect_(Labda[i],Labda[j]) then
                begin
                    // ha a kétlabda ütközik, akkor teljesen rugalmas modellel
                    // cserélik ki sebességeiket
                    v1:=Labda[i].dx;
                    v2:=Labda[j].dx;
                    Labda[i].dx:=v2;
                    Labda[j].dx:=v1;
                    v1:=Labda[i].dy;
                    v2:=Labda[j].dy;
                    Labda[i].dy:=v2;
                    Labda[j].dy:=v1;
                end;
            // ütközéskor irányváltás a határokon
            for i:=1 to labdaszam do
                begin
                    if (Labda[i].Pos.x+Labda[i].meret+Labda[i].dx>form1.clientwidth) or
                        (Labda[i].Pos.x+Labda[i].dx<0) then Labda[i].dx:=-Labda[i].dx;
                    if (Labda[i].Pos.y+Labda[i].meret+Labda[i].dy>form1.clientheight) or
                        (Labda[i].Pos.y+Labda[i].dy<0) then Labda[i].dy:=-Labda[i].dy;
                    segedpos.x:=Labda[i].Pos.x+Labda[i].dx;
                    segedpos.y:=Labda[i].Pos.y+Labda[i].dy;
                    Labda[i].Pos:=segedpos;
                end;
            end;
    end;

```

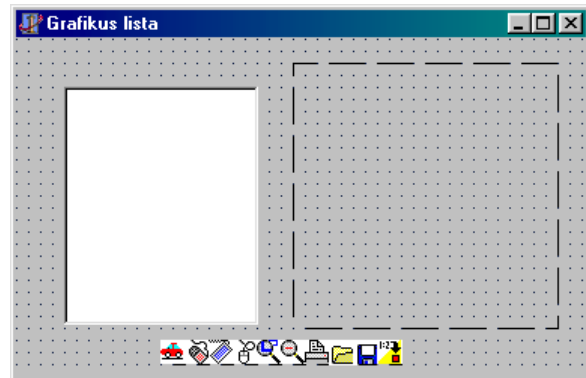




A vezérlő- és a *Canvas*-grafikát együtt használva készíthetünk olyan listaablakot, amely képeket jelenít meg. Az alkalmazásban a listaablakból kiválasztott képet nagyítva is megjelenítjük egy **Image** vezérlőben. A képek listaablakban való tárolásához a listaablakok egy speciális, a felhasználó által rajzolt típusát kell használnunk. A tervezés alatt tíz előre elkészített bittérképet betöltünk az *Image1*,...*Image10* vezérlők **Picture** tulajdonságába és mindegyiket láthatatlanná tesszük. Egy listaablakot (*ListBox1*) is létrehozunk a képek listában való tárolására, egy **Image** vezérlőt (*Kep*) pedig a kiválasztott kép megjelenítése céljából helyezünk el formon.

Az elmondottak szerint kialakított form osztálya:

```
type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    Kep: TImage;
    Image1 : TImage;
    Image2 : TImage;
    Image3 : TImage;
    Image4 : TImage;
    Image5 : TImage;
    Image6 : TImage;
    Image7 : TImage;
    Image8 : TImage;
    Image9 : TImage;
    Image10: TImage;
  procedure FormCreate(Sender: TObject);
  procedure ListBox1DrawItem(Control: TWinControl; Index: Integer;
    Rect: TRect; State: TOwnerDrawState);
  procedure ListBox1Click(Sender: TObject);
  private
    Kepek: array [0..9] of TImage;
  end;
```



A *Kepek* tömböt azért hoztuk létre, hogy egyszerűbben tudjuk kezelni a képeket. A form létrehozásakor a képek referenciáját betöltjük a tömbbe, feltöltjük a listaablakot és aktuálissá tesszük az első listaelemet.

```
procedure TForm1.FormCreate(Sender: TObject);
var i:integer;
begin
  // mind a 10 képet tömbbe tesszük
  kepek[0]:=Image1;
  kepek[1]:=Image2;
  kepek[2]:=Image3;
  kepek[3]:=Image4;
  kepek[4]:=Image5;
  kepek[5]:=Image6;
  kepek[6]:=Image7;
  kepek[7]:=Image8;
  kepek[8]:=Image9;
  kepek[9]:=Image10;
  // feltöltjük a listaablakot
  listbox1.items.clear;
  for i:=0 to 9 do
  begin
    listbox1.items.add('KÉP'+inttostr(i+1))
  end;
  // az első az aktuális elem
  listbox1.itemindex:=0;
  // a program fogja rajzolni a listaablak elemeit
  listbox1.style:=lbOwnerDrawFixed;
  // a kiválasztott listaelem képe nagyított lesz
  kep.stretch:=true;
  // a nagyított kép az aktuális kép
  kep.picture:=kepek[0].picture;
end;
```

A felhasználó által rajzolt listaablak esetén a Windows minden szükséges esetben meghívja a *DrawItem* eseménykezelőt, melynek átadja az éppen kifestendő elem indexét (*Index*), a kifestendő elem befoglaló téglalapját (*Rect*) és azt is, hogy az elem kiválasztott, letiltott vagy éppen birtokolja az inputfókusz (State). Ezen adatok alapján magunk jeleníthetjük meg a listaelemet.

```

procedure TForm1.ListBox1DrawItem(Control: TWinControl;
                                   Index: Integer; Rect: TRect;
                                   State: TOwnerDrawState);
var rectu:TRect;
begin
    // kifestéskor a listaelem magassága határozza meg a kép méretét
    rectu.left:=rect.left+1;
    rectu.top:=rect.top+1;
    rectu.right:=rect.left+listbox1.itemheight-2;
    rectu.bottom:=rect.top+listbox1.itemheight-2;
    // a kép mellé szöveget is írunk
    listbox1.canvas.textrect(rect, rect.left+25, rect.top+2,
                             'KÉP'+inttostr(index+1));

    // a kép kirajzolása
    listbox1.canvas.StretchDraw(rectu,
                                (kepek[index] as TImage).picture.bitmap);
end;

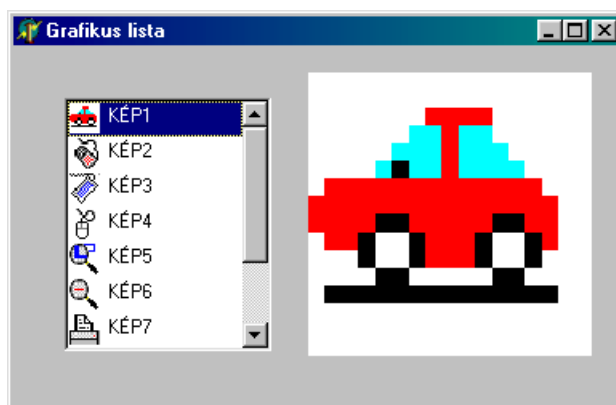
```

A listaablakot ezek után ugyanúgy kezelhetjük, mintha nem magunk festenénk. Amikor elemet választunk, akkor az ahhoz tartozó képet kinagyítjuk a *Kép* vezérlőben.

```

procedure TForm1.ListBox1Click(Sender: TObject);
begin
    // a nagyított kép az aktuális kép
    kep.picture:=kepek[listbox1.itemindex].picture;
end;

```

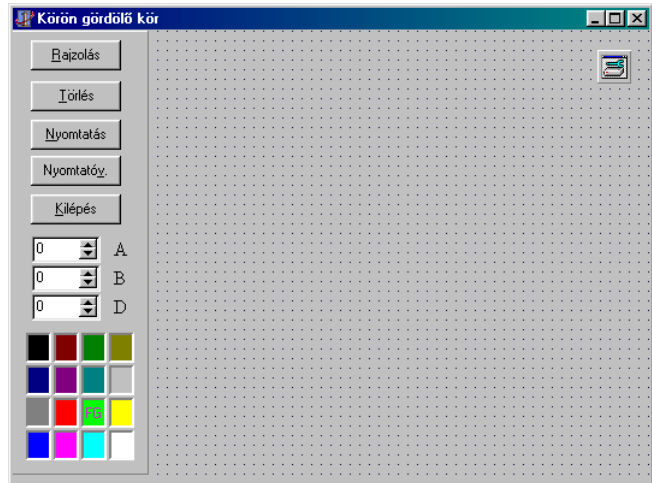




A feladat megoldása során körön legördülő kör adott pontja által leírt pályát jelenítjük meg. A feladat matematikai oldalát nem részletezzük, így csak annyit kell tudnunk, hogy a körök illetve a pont adatait görgethető szövegszerkesztő vezérlő (*SpinEdit*) segítségével adhatjuk meg. *D*-vel jelöltük a rögzített kör sugarát, míg a gördülő kör és a rajta levő pont adatait az *A* és *B* jelű vezérlőkkel állíthatjuk be.

A program vezérléséhez használt ablakelemeket külön panelen (*Panel*) helyeztük el. A főablakot leíró osztály deklarációja:

```
type
  TForm1 = class(TForm)
    ButtonPanel: TPanel;
    RajzolGomb: TButton;
    NyomtatasGomb: TButton;
    TorlesGomb: TButton;
    NyomtatoValasztoGomb: TButton;
    KilepesGomb: TButton;
    SpinA: TSpinEdit;
    SpinB: TSpinEdit;
    SpinD: TSpinEdit;
    SpinALabel: TLabel;
    SpinBLabel: TLabel;
    SpinDLabel: TLabel;
    ColorGrid1: TColorGrid;
    PrinterSetupDialog1: TPrinterSetupDialog;
  procedure KilepesGombClick(Sender: TObject);
  procedure RajzolGombClick(Sender: TObject);
  procedure TorlesGombClick(Sender: TObject);
  procedure NyomtatasGombClick(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure FormPaint(Sender: TObject);
  procedure NyomtatoValasztoGombClick(Sender: TObject);
  procedure FormResize(Sender: TObject);
  end;
```



Az ablak létrehozásakor csak a program indításához szükséges kezdőértékeket állítjuk be:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Torolt := true;
  SpinA.Value := 166;
  SpinB.Value := 68;
  SpinD.Value := 65;
end;
```

A rajzolást végző *Gordit* eljárást külön modulban (*GORDUL*) helyeztük el. Ahhoz, hogy tetszőleges *Canvas* jellemzővel rendelkező komponensre rajzolni tudjunk, a rajzolás (*Canvas*) objektumát paraméterként kapja az eljárás. Ugyancsak paraméterekben adjuk át a görgetés jellemző adatait (*a*, *b*, *c*), a rajzterület középpontjának (*cx*, *cy*) koordinátáit és a rajzszínt.

```

procedure Gordit(Canvas: TCanvas; a, b, d, cx, cy: Integer;
                  Rajzszin: TColor) ;
const
    szogosztas = 100;
var
    rab, vonalszam, i : Integer;
    alfa, beta, dalfa, aperb : Real;
    xpt, ypt : Real;
begin
    try
        rab := a-b; alfa := 0.0;
        dalfa := pi/szogosztas; aperb := a/b;
        vonalszam := 2 * szogosztas * (b div Lnko(a,b));
        Canvas.Pen.Color := Rajzszin;
        Canvas.MoveTo((round(rab+d)+cx), cy);
        for i := 1 to vonalszam do
            begin
                alfa := alfa + dalfa;
                beta := alfa * aperb;
                xpt := rab * cos(alfa) + d * cos(beta);
                ypt := rab * sin(alfa) - d * sin(beta);
                Canvas.LineTo(round(xpt)+cx, round(ypt)+cy);
            end
        except
            ShowMessage('Számolási hiba történt.'#13#10+
                        'Kérem változtassa meg az A és B'
                        'paraméterek értékét!');
        end;
    end;

```

A *Gordit* eljárást a program különböző pontjairól hívjuk. A program ablakába a *Rajzolás* gomb megnyomásakor, illetve az ablak újrafestésekor rajzolunk (ha az ablak tartalmát nem töröltük – *Torolt*).

```

procedure TForm1.RajzolGombClick(Sender: TObject);
var
    cx, cy : Integer;
begin
    Torolt := False;
    cy := ClientHeight div 2;
    cx := (ClientWidth + ButtonPanel.Width) div 2;
    Canvas.Pixels[cx, cy] := clBlack;
    Gordit(Canvas, SpinA.Value, SpinB.Value, SpinD.Value,
           cx, cy, ColorGrid1.ForegroundColor);
end;

procedure TForm1.FormPaint(Sender: TObject);
var
    cx, cy : Integer;
begin
    if not Torolt then
        begin
            cy := ClientHeight div 2;
            cx := (ClientWidth div 2) + (ButtonPanel.Width div 2);
            Canvas.Pixels[cx, cy] := clBlack;
            Gordit(Canvas, SpinA.Value, SpinB.Value, SpinD.Value,
                   cx, cy, ColorGrid1.ForegroundColor);
        end;
    end;

```

Ugyancsak a *Gordit* eljárást hívjuk a nyomtatóra történő rajzoláskor. Ekkor a rajz jellemzőit úgy alakítjuk, hogy a rajz jól elférjen a papíron.

```

procedure TForm1.NyomtatasiGombClick(Sender: TObject);
var
    cx, cy : Integer;
    SkalaTenyezo : Integer;
    Felirat : string;
begin
    with Printer do
        begin
            SkalaTenyezo := PageWidth div ClientHeight;
            cx := PageWidth div 2;
            cy := PageHeight div 2;
            BeginDoc;
            Gordit(Printer.Canvas,
                SpinA.Value*SkalaTenyezo,
                SpinB.Value*SkalaTenyezo,
                SpinD.Value*SkalaTenyezo,
                cx, cy, clBlack);
            Felirat := 'A=' + IntToStr(SpinA.Value) +
                '    B=' + IntToStr(SpinB.Value) +
                '    D=' + IntToStr(SpinD.Value);
            cy := Trunc(PageHeight - (PageHeight * 0.07));
            cx := (PageWidth - Canvas.TextWidth(Felirat)) div 2;
            Canvas.TextOut(cx, cy, Felirat);
            EndDoc;
        end;
    end;

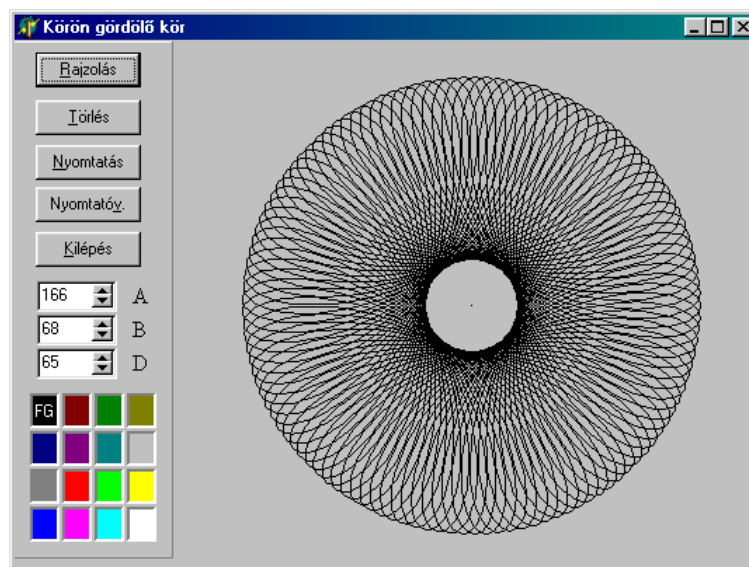
```

A nyomtatás paramétereit a *Nyomtatóv. gomb* megnyomásakor megjelenő szabványos párbeszédablak (*PrinterSetupDialog*) segítségével állíthatjuk be.

```

procedure TForm1.NyomtatasiValasztasGombClick(Sender: TObject);
begin
    PrinterSetupDialog1.Execute;
end;

```





Készítsünk olyan programot, amely a vízszintestől eltérő szögben jelenít meg szöveget! (A feladat nem oldható meg a *Canvas* eszközeivel!) (*FerdeIras*)

A feladatot megoldó programban gombnyomásra különböző szögekkel és méretben szöveget írunk ki. A programban a szövegek színét, kiírásának szögét (*a*) és méretét (*m*) ciklusban változtatjuk. A körbeforgatást 24-szer végezzük el. Az időzítést a *Sleep* hívással biztosítjuk.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  a, m, i : integer;
begin
  canvas.Brush.style:=bsClear;
  for i:=1 to 24 do
    begin
      a:=0;
      m:=10;
      while a<=360 do
        begin
          if i mod 2 = 0 then
            SetTextColor(canvas.handle,clYellow)
          else
            SetTextColor(canvas.handle,clOlive);
          // ferde kiírás
          FTextout(form1.canvas, a, m, 300,200,'Borland Delphi');
          a:=a+20;
          inc(m,3);
        end;
        sleep(200); // felfüggeszti az aktuális szál működését
      end;
    end;
end;
```

A kiírást az *FTextout* eljárás végzi, ahol a GDI szabályainak megfelelően logikai fontot készítünk, azt beválasztjuk az eszközkapcsolatba, írunk, majd visszaállítjuk az eszközkapcsolat eredeti állapotát.

```
procedure FTextOut(Canvas : TCanvas; szog, mag:integer; x,y:integer; sz:string);
var
  lf : TLogFont;
  hold, hf : HFont;
begin
  // A ferde betűk kiírása csak a GDI eszközeivel oldható meg
  // Logikai font definíció
  with lf do
    begin
      lfHeight := mag; // Magasság
      lfWidth := round(0.4*mag); // szélesség
      lfWeight :=FW_NORMAL; // kövérség
      lfEscapement := szog*10; // szög/10
      lfOrientation := szog*10;
      lfPitchAndFamily := 3;
      lfFaceName := 'Times New Roman';
      lfItalic := 0;
      lfUnderline := 0;
      lfStrikeOut := 0;
    end;
  hf := CreateFontIndirect(lf);
  // Az eszközkapcsolat fontját cseréljük
  hold:=SelectObject(Canvas.Handle, hf);
  canvas.Textout(x,y,'Borland Delphi');
  canvas.font.style:=[fsUnderline];
  // Az eszközkapcsolat fontját visszacseréljük
  SelectObject(Canvas.Handle, hold);
  // Letöröljük a logikai fontot
  DeleteObject(hf);
end;
```