IV. A Delphi grafikája

A Delphi grafikája teljesen ráépül a Windows grafikus alprogramrendszerére, a GDI (*Graphics Device Interface*) filozófiára. A GDI eszközvezérlő programokon keresztül kezeli a grafikus perifériákat és ezáltal lehetővé teszi, hogy a rajzgépet, a nyomtatót, a képernyőt egységesen használjuk. A GDI programozásakor bármilyen hard eszközt, meghajtót figyelmen kívül hagyhatunk. A szinek használata is úgy van megoldva, hogy nem kell foglalkoznunk a konkrét fizikai keveréssel és kialakítással. Ezáltal a pixel adatokat is eszközfüggetlenül használhatjuk. Hasonlóan van megoldva a karakterek, fontok eszközfüggetlen megjelenítése is. A *TrueType* fontok használata biztosítja azt, hogy a megtervezett szöveg nyomtatásban is ugyanolyan lesz, mint ahogy azt a képernyőn láttuk. A GDI nagy előnye az is, hogy saját koordinátarendszerrel dolgozhatunk, virtuális távolságokkal írhatjuk meg, a konkrét hárdvertől függetlenül, az alkalmazásunkat. Mindezen előnyök mellett azonban a GDI továbbra is kétdimenziós, egészkoordinátájú grafikus rendszer maradt. A GDI nem támogatja az animációt.

A GDI filozófiának az alapja az, hogy először meghatározunk egy eszközleírót, amely a fizikai eszközzel való kapcsolatot rögzíti. Ez tulajdonképpen egy rajzeszközhalmaz és egy sor adat kapcsolata. Az adatokkal megadhatjuk a rajzolás módját. Ezután ezt az eszközleírót használva specifikálhatjuk azt az eszközt, emelyen rajzolni szeretnénk. Például, ha egy szöveget szeretnénk megjelentetni a képernyőn, akkor először rögzítjük az eszközkapcsolat révén a karakterkészletet, a színt, a karakterek nagyságát, típusát, azután pedig specifikáljuk a kiírás helyét (*x* és *y* koordinátáit), illetve a kiírandó szöveget.

A Delphi rendszer az összes grafikus objektumot és megjelenítőrutint a ***Graphics*** unitban tárolja. Az eszközkapcsolatot és magát a rajzolás alapegységét is megvalósító objektumot a *TCanvas* osztály képezi. Minden speciális megjelenítő objektum (*Form, Printer, Image*) tartalmaz egy *TCanvas* típusú *Canvas* nevet viselő tulajdonságot. A konkrét eszközkapcsolat meghatározás és rajzolás ezen *Canvas* objektum segítségével történik, amely nem más, mint az eszközkapcsolat objektumorientált megfogalmazása.

A ***Graphics*** unit használja a hagyományos API (*Application Programming Interface*) függvényeket és eljárásokat is. A *Canvas Handle* tulajdonsága tulajdonképpen az eszközkapcsolat HDC típusú leírásával egyezik meg. A tulajdonság segítségével tehát bármikor áttérhetünk a hagyományos API rutinok használatára is.

A *Canvas* objektumot egy festőkészletként képzelhetjük el. A *Canvas* tulajdonságok a rajzolási attribútumokat, a rajzeszközök és a rajzvászon jellegzetességeit állítják, a metódusok pedig a konkrét rajzoláshoz szükséges rutinokat biztosítják. A *Canvas* objektum alapvető tulajdonságai alapvető információkat szolgálnak a toll (vonalas ábrák rajzolása), az ecset (kitöltőminták), a fontok (szövegek megjelenítése) és a bittérképek attribútumairól, jellegzetességeiről.

Tollak

A vonalas ábrák készítésének alapvető eszköze a toll. A tollakat a *TPen* osztály és az objektumok *Pen* tulajdonságai valósítják meg. A tollak jellemzői a szín (*Color*), vonalvastagság (*Width*), vonaltípus (*Style*) és a rajzolási mód (Mode).

A Delphi rendszer a színeket a *TColor = -(COLOR_ENDCOLORS + 1)..$2FFFFF*; típussal kezeli le. A szindefinícióban a piros, zöld és kék értékeket az *rr*, *gg* és *bb* számok jellemzik (*$00bbggrr*). Saját szín keverésére is van lehetőség a **function** *RGB (R: byte; G: byte; B: byte): longint*; függvény segítségével. A ***Graphics*** unit a leggyakrabban használt színeket konstansként deklarálja (*clBlack = TColor($000000), clRed = TColor($0000FF)* stb.).

A húzott vonal vastagságát a *Width* tulajdonság által lehet megadni. A mértékegység itt a pixel.

A húzott vonal típusát a *Style* tulajdonsággal lehet beállítani. Ez a tulajdonság *TPenStyle = (psSolid, psDadh, psDot, psDashDot, psDashDotDot, psClear, psInside-Frame)*; típusú.

A *Mode* tulajdonság segítségével a rajzolási módot állíthatjuk be. A rajzolási mód azt jelenti, hogy bizonyos logikai műveleteket használva, a háttér színe és a toll színe fogja meghatározni a vonal színét. A megfelelő logikai műveleteket a *TPenMode = (pmBlack, pmWhite, pmNop, pmNot, pmCopy, pmNotCopy,pmMergePenNot, pmMaskPenNot, pmMergeNotPen, pmMaskNotPen, pmMerge,pmNotMerge, pmMask, pmNotMask, pmXor, pmNotXor)*; típus definiálja.

Ebben a szellemben, a *TPen* osztály a következő deklarációkat foglalja magába:

```
TPen = class(TGraphicsObject)
 private
  FMode: TPenMode;
  procedure GetData(var PenData: TPenData);
  procedure SetData(const PenData: TPenData);
 protected
  function GetColor: TColor;
  procedure SetColor(Value: TColor);
  function GetHandle: HPen;
  procedure SetHandle(Value: HPen);
  procedure SetMode(Value: TPenMode);
  function GetStyle: TPenStyle;
  procedure SetStyle(Value: TPenStyle);
  function GetWidth: Integer;
  procedure SetWidth(Value: Integer);
 public
  constructor Create;
  destructor Destroy; override;
  procedure Assign(Source: TPersistent); override;
  property Handle: HPen read GetHandle write SetHandle;
 published
  property Color: TColor read GetColor write SetColor default
                                                    clBlack;
  property Mode: TPenMode read FMode write SetMode default pmCopy;
  property Style: TPenStyle read GetStyle write SetStyle default
                                                    psSolid;
  property Width: Integer read GetWidth write SetWidth default 1;
 end;
```

Ecsetek

Ábrák kifestéséhez ecseteket használunk. A *Canvas* objektum hasonlóan kezeli a tollakat és az ecseteket. Minden festő metódus az aktuális ecsetet használja. Az ecset objektumorientált koncepciója a *TBrush* osztály által valósul meg. A *Brush* változók jellemzői a szín és a kifestés módja. A kifestés módja a tulajdonképpeni kitöltőmintát adja meg. Ez a következő típusdeklarációnak felel meg: *TBrushStyle = (bsSolid, bsClear, bsHorizontal, bsVertical, bsFDiagonal, bsBDiagonal, bsCross, bsDiagCross)*;. Ha beállítjuk a *Bitmap* tulajdonságát, akkor az így megadott bittérképet használja festőmintaként. A *TBrush* osztály tehát a következ[:

```
TBrush = class(TGraphicsObject)
       private
        procedure GetData(var BrushData: TBrushData);
        procedure SetData(const BrushData: TBrushData);
```

```
protected
  function GetBitmap: TBitmap;
  procedure SetBitmap(Value: TBitmap);
  function GetColor: TColor;
  procedure SetColor(Value: TColor);
  function GetHandle: HBrush;
  procedure SetHandle(Value: HBrush);
  function GetStyle: TBrushStyle;
  procedure SetStyle(Value: TBrushStyle);
public
  constructor Create;
  destructor Destroy; override;
  procedure Assign(Source: TPersistent); override;
  property Bitmap: TBitmap read GetBitmap write SetBitmap;
  property Handle: HBrush read GetHandle write SetHandle;
published
  property Color: TColor read GetColor write SetColor default
                                                    clWhite;
  property Style: TBrushStyle read GetStyle write SetStyle
                                                  default bsSolid;
end;
```

Fontok

A karakterek eszközfüggetlen megjelenítését a Windows a *TrueType* fontok segítségével érte el. A *TrueType* fontok tulajdonképpen pontok és speciális algoritmusok halmaza, amelyek eszköztől és felbontástól függetlenül képesek karaktereket megjeleníteni.

A *Canvas* tulajdonsága a *Font* is, amely egy *TFont* típusú objektum és a karakterek beállításait szolgálja. A *TFont* tulajdonságai a font mérete (*Size: integer*), a karakterek színe (*Color: TColor*), a karakter által lefoglalt cella magassága (*Height: integer*), a font neve (*Name: TFontName*) valamint a karakter stílusa (*Style: TFontStyles*). A dőlt, félkövér, aláhúzott vagy áthúzott betűket a következő típus segítségével lehet definiálni:

```
TFontStyle = (fsBold, fsItalic, fsUnderline, fsStrikeOut);
TFontStyles = set of TFontStyle;
```

A *TFontName* típust a következő deklaráció határozza meg:

```
TFontName = string(LF_FACESIZE - 1);
```

Természetesen, amikor karaktereket akarunk megjelentetni, akkor beállíthatjuk a *TFont* objektum ezen tulajdonságait, de elegánsabb megoldás az, hogy egy *TFontDialog* típusú dialógusdoboz segítségével állítjuk be a karakterek jellemzőit.

Bittérképek

A bittérképek speciális memóriaterületeket jelölnek, amelyeknek bitjei egy-egy kép megjelenését definiálják. Fekete-fehér képernyőn nagyon egyszerű ez a megjelenítés, ha az illető bit 0, akkor a a képpont fekete, ha pedig 1, akkor a képpont fehér. Szines képernyők esetén nem elegendő egyetlen bit a képpont tárolásához, ekkor vagy több szomszédos bit segítségével kódoljuk a képpontot, vagy a bittérképet több színsíkra tagoljuk és ezek együttesen határozzák meg a képpontot.

A bittérképet a *TBitmap* típus valósítja meg, amely számos információt tartalmaz a bittérkép méretéről (*Height, Width*), típusáról (*Monochrome*), arról, hogy tartalmaz-e értékes információt (*Empty*), valamint metódusai segítségével kimenthetjük, beolvashatjuk (*SaveToFile, LoadFromFile, LoadFromStream, SaveToStream*) vagy a vágóasztal segítségével átadhatjuk a tárolt információt (*LoadFromClipboardFormat, SaveToClipboardFormat*).

Maga a *TBitmap* is tartalmaz egy *Canvas* tulajdonságot, amely segítségével rajzolhatunk, írhatunk a bittérképre.

A Canvas

Ezen ismeretek birtokában rátérhetünk a *TCanvas* objektum ismertetésére. Mint már említettük, a *Canvas* nem más, mint az eszközkapcsolat-leíró objektum-orientált megfogalmazása. A *Canvas* tulajdonságok a rajzolás jellemzőit állítják be, a *Canvas* metódusok pedig megvalósítják a rajzolást. A *TCanvas* típus a következő:

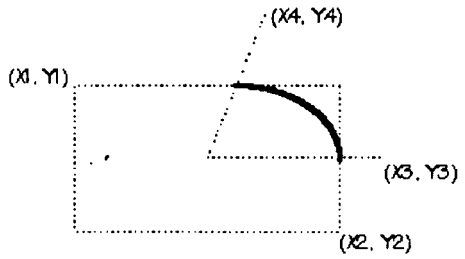
```
TCanvas = class(TPersistent)
 private
  FHandle: HDC;
  State: TCanvasState;
  FFont: TFont;
  FPen: TPen;
  FBrush: TBrush;
  FPenPos: TPoint;
  FCopyMode: TCopyMode;
  FOnChange: TNotifyEvent;
  FOnChanging: TNotifyEvent;
  FLock: TRTLCriticalSection;
  FLockCount: Integer;
  procedure CreateBrush;
  procedure CreateFont;
  procedure CreatePen;
  procedure BrushChanged(ABrush: TObject);
  procedure DeselectHandles;
  function GetClipRect: TRect;
  function GetHandle: HDC;
  function GetPenPos: TPoint;
  function GetPixel(X, Y: Integer): TColor;
  procedure FontChanged(AFont: TObject);
  procedure PenChanged(APen: TObject);
  procedure SetBrush(Value: TBrush);
  procedure SetFont(Value: TFont);
  procedure SetHandle(Value: HDC);
  procedure SetPen(Value: TPen);
  procedure SetPenPos(Value: TPoint);
  procedure SetPixel(X, Y: Integer; Value: TColor);
 protected
  procedure Changed; virtual;
  procedure Changing; virtual;
  procedure CreateHandle; virtual;
  procedure RequiredState(ReqState: TCanv  State);
 public
  constructor Create;
  destructor Destroy; override;
  procedure Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
  procedure BrushCopy(const Dest: TRect; Bitmap: TBitmap;
                      const Source: TRect; Color: TColor);
  procedure Chord(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
  procedure CopyRect(const Dest: TRect; Canvas: TCanvas;
                      const Source: TRect);
  procedure Draw(X, Y: Integer; Graphic: TGraphic);
  procedure DrawFocusRect(const Rect: TRect);
  procedure Ellipse(X1, Y1, X2, Y2: Integer);
  procedure FillRect(const Rect: TRect);
  procedure FloodFill(X, Y: Integer; Color: TColor; FillStyle:
                      TFillStyle);
  procedure FrameRect(const Rect: TRect);
  procedure LineTo(X, Y: Integer);
```

```
    procedure Lock;
    procedure MoveTo(X, Y: Integer);
    procedure Pie(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
    procedure Polygon(const Points: array of TPoint);
    procedure Polyline(const Points: array of TPoint);
    procedure Rectangle(X1, Y1, X2, Y2: Integer);
    procedure Refresh;
    procedure RoundRect(X1, Y1, X2, Y2, X3, Y3: Integer);
    procedure StretchDraw(const Rect: TRect; Graphic: TGraphic);
    function TextExtent(const Text: string): TSize;
    function TextHeight(const Text: string): Integer;
    procedure TextOut(X, Y: Integer; const Text: string);
    procedure TextRect(Rect: TRect; X, Y: Integer; const Text:
                                                  string);
    function TextWidth(const Text: string): Integer;
    function TryLock: Boolean;
    procedure Unlock;
    property ClipRect: TRect read GetClipRect;
    property Handle: HDC read GetHandle write SetHandle;
    property LockCount: Integer read FLockCount;
    property PenPos: TPoint read GetPenPos write SetPenPos;
    property Pixels[ X, Y: Integer] : TColor read GetPixel write
                                                  SetPixel;
    property OnChange: TNotifyEvent read FOnChange write FOnChange;
    property OnChanging: TNotifyEvent read FOnChanging write
                                                  FOnChanging;
  published
    property Brush: TBrush read FBrush write SetBrush;
    property CopyMode: TCopyMode read FCopyMode write FCopyMode
                                                  default cmSrcCopy;
    property Font: TFont read FFont write SetFont;
    property Pen: TPen read FPen write SetPen;
  end;
```

A *Canvas* rajzolási módszerei hasonlítanak a Turbo Pascal grafikájához, egy pár fontosabb eltéréssel. A pixelgrafika itt a *Pixels[X, Y: Integer]: TColor;* tulajdonság segítségével valósul meg. Az *X* és az *Y* indexek a képernyő megfelelő pontjának a koordinátáit jelentik, a tömbelem pedig a pont színét. Teljes kifestett ellipszist rajzolhatunk az *Ellipse(X1, Y1, X2, Y2: Integer)*; metódus segítségével. A megadott paraméterek azt a téglalapot definiálják, amely tartalmazza az ellipszist. Az ellipszis középpontja a téglalap középpontja lesz, illetve tengelyei is megegyeznek a téglalap tengelyeivel. Az ellipszisívek, ellipsziscikkek és ellipszisszeletek rajzolása egy kissé szokatlan. Ezek a következ[metódusok segítségével történnek:

```
procedure Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
procedure Pie(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
procedure Chord(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
```

A metódusoknak meg kell adni az ellipszist befogadó téglalapot *(X1, Y1, X2, Y2)*, egy kezdőpontot *(X3, Y3)* valamint egy végpontot *(X4, Y4)*. A kezdő és a végpont egy szögtartományt definiál. Ez ellipszisív, -cikk vagy -szelet ebben a szögtartományban lesz meghúzva, az aktuális tollal és rajzolási móddal, az óramutató járásával ellentétes irányban:



Lekerekített sarkú téglalapot rajzolhatunk a *RoundRect(X1, Y1, X2, Y2, X3, Y3: Integer)*; metódus segítségével. Az *X3, Y3* az ellipszis nagy illetve kis tengelye.

A rajzvászonra a *TextOut(X, Y: Integer; const Text: String)*; illetve a *TextRect(Rect: TRect; const Text: String)*; metódus segítségével írhatunk. A *TextOut* az *(X, Y)* ponttól kezdve kiírja a *Text* szöveget, a *TextRect* pedig a *Text* szöveget csak a *Rect* téglalap által meghatározott részben jeleníti meg. Azt, hogy mekkora helyet foglal le a kiírt szöveg, a *TextExtent(const Text: string): TSize;* függvény segítségével tudhatjuk meg. Ha csak a szöveg hosszára vagy magasságára vagyunk kíváncsiak, akkor a *TextHeight(const Text: string): Integer;* vagy a *TextWidth(const Text: string): Integer;* függvényeket használjuk.

Ha valamilyen grafikus ábrát, vagy bittérképet kívánuk megjeleníteni a rajzvászonon, akkor a *Draw(X, Y: Integer; Graphic: TGraphic);* vagy a *StretchDraw(const Rect: TRect; Graphic: TGraphic)*; metódust használjuk. A *StretchDraw* metódus nagyítva vagy kicsinyítve jelenteti meg az ábrát úgy, hogy ez teljesen töltse ki a *Rect* téglalapot.

Nyomtatás

Delphiben a grafikus nyomtatás a *Printers* unit használatával valósul meg. Ez a unit deklarál egy *TPrinter* típusú *Printer* objektumot, amelynek tulajdonságai között szerepel a *Canvas* is. Ha a erre a *Canvas*-ra rajzolunk vagy írunk, akkor az megjelenik a nyomtatón. Az aktuális papírméretről információkat nyerhetünk a *Printer* objektum *PageHeight* illetve *PageWidth* tulajdonságai segítségével. A nyomtatást a *BeginDoc* metódussal kezdeményezhetjük és az *EndDoc* metódussal fejezzük be. Bármikor áttérhetünk új oldalra a *NewPage* metódus meghívásával.

```
with Printer do
 begin
  BeginDoc;
   Canvas.TextOut (20, 20, 'Az els[ lap.');
  Canvas.MoveTo(50, 50);
  Canvas.LineTo(200, 200);
  Canvas.Rectangle(40, 40, 250, 220);
  NewPage;
  Canvas.TextOut(20, 20, 'A második lap.');
  EndDoc;
 end;
```

Kovács Lehel
Kolozsvár

A molekulák egyik óriásbébije: a C_{60}-as molekula

Buckminster Fuller építész, aki az 1967-es montreali EXPO gömbalakú, amerikai pavilonját tervezte, bizonyára nem gondolta, hogy a szén harmadik kristályos módosulatát róla fogják elnevezni. A szóban forgó pavilon ugyanis egy óriási futballabda volt, amit szabályos öt- és hatszögű szeletekből alakított ki. Amint utólag kiderült, a fullerén molekula (hiszen róla van szó) kísértetiesen hasonlít egy ilyen, szabályos öt- és hatszögű szeletekből álló futballabdához (bucky-ball – 1. ábra).

A fullerének közfigyelmet felkeltő története 1984-ben kezdődött, amikor először észleltek, grafitból ívkisüléssel készült korom tömegspektrumában éles