tonin receptora is ezek közé tartozik. A gyógyszerek körülbelül fele e receptorokon keresztül fejti ki hatását. 2011-ben Kobilka képalkotó eljárással „megörökítette" a pillanatot, amikor az adrenalin receptorát aktiválja a hormon és jelet küld a sejtnek. Ezzel a fontos receptorcsalád, a G-fehérjekapcsolt receptorok belső működését sikerült megérteni és leírni.

Robert J. Lefkowitz 1943-ban született New Yorkban, orvosi diplomáját a Columbia Egyetemen szerezte 1966-ban, jelenleg a Howard Hughes Orvosi Intézet és a Duke Egyetem Orvosi Központjának professzora.

Brian K. Kobilka 1955-ben született Little Fallsban, orvosi diplomáját 1981-ben szerezte a Yale Egyetemen, jelenleg a Stanford Egyetem orvosi karának professzora.

Robert J. Lefkowitz

Brian K. Kobilka

A 2012-es irodalmi Nobel-díjat az 1955-ben született *Mo Yan*, a napjainkban élő legnevesebb kínai írónak ítélte a Svéd Akadémia azzal az indoklása, hogy egyedülálló munkássága során egy sajátos műfaj, a „hallucinatorikus realizmus" révén ötvözi a meséket, a történelmet és a jelent.

Október 12-én az utolsó megnevezést, a béke Nobel-díjra az Európai Unió kapta azzal az indoklással, hogy hat évtizede járul hozzá a béke, a demokrácia és az emberi jogok előmozdításához.

Mo Yan

M. E.

Ismerd meg!

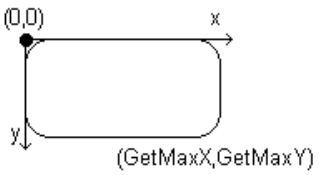
Számítógépes grafika

XXIV. rész

Grafika DOS alatt – II.

A Borland cég által írt *Graph* unit közel 80 rutint tartalmazó grafikus gyűjtemény, amely egészen a bitműveletektől a magas szintű funkciókig mindenféle rutint tartalmaz.

Hogy egy Graph unit-ot használó programot futtathassunk, szükségünk van egy vagy több grafikus meghajtóra (**.BGI** állományok *Borland Graphic Interface*) az **.EXE** programon kívül. Ha a programunk fontokat is használ, akkor szükségünk van még font (**.CHR**) állományokra is. Ezeket az állományokat a telepítő program a megfelelő (rendszerint ...\bp\bgi) alkönyvtárba helyezi el. A **.BGI** állományokat be lehet fordítani az **.EXE** állományba. Erre a célra a **BINOBJ** nevű programot kell felhasználni. Ennek a segítségével a .**BGI** állományt **.OBJ** állománnyá alakíthatjuk át, majd ezt a ***{$L név}*** direktívával az **.EXE** állományba fordíthatjuk.



1. ábra
A BGI grafika koordinátarendszere

Példaként álljon itt az **EGAVGA.BGI** grafikus meghajtó befordítása. Először a **DOS** promptnál elindítjuk a **BINOBJ** programot a kívánt paraméterekkel: a meghajtó-állomány neve, az **OBJ** állomány neve és a **public**-ként deklarált főeljárás neve:

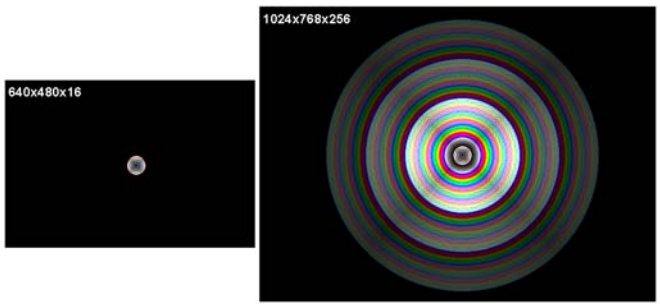
```
c:\>BINOBJ  EGAVGA.BGI  VGADRV.OBJ  VGADriver
```

Majd megírjuk a megfelelő *Pascal* programot:

```
1. program VGAMode;
2.
3. uses Graph;
4.
5. procedure VGADriver; external;
6. {$L VGADRV.OBJ}
7.
8. procedure InitVGA(Mode: integer);
9. var gd: integer;
10.begin
11.   gd := RegisterBGIDriver(@VGADriver);
12.   if gd < 0 then
13.    begin
14.      writeln(GraphErrorMsg(GraphResult));
15.      Halt(1);
16.    end;
17.   gd := VGA;
18.   InitGraph(gd, Mode, '');
19.   gd := GraphResult;
20.   if gd <> GrOK then
21.    begin
22.      writeln(GraphErrorMsg(gd));
23.      Halt(1);
24.    end;
25.end;
26.
27.begin
28.  InitVGA(VGAHi);
29.  Line(0, 0, GetMaxX, GetMaxY);
30.  readln;
31.  CloseGraph;
32.end.
```

Az alábbi *Pascal* program hagyományosan (a **.BGI** grafikus meghajtó belefordítása nélkül) inicializálja a grafikus üzemmódot, és különböző színű koncentrikus köröket rajzol ki:

```
1.  program EGAVGA;
2.
3.  uses Graph;
4.
5.  var
6.    GraphMode, GraphDriver, GrErr: integer;
7.    i: byte;
8.  begin
9.    GraphDriver := Detect;
10.   InitGraph(GraphDriver, GraphMode, '');
11.   GrErr := GraphResult;
12.   if GrErr <> grOK then
13.    begin
14.      writeln('Graphics error: ',
15.                GraphErrorMsg(GrErr));
16.      readln;
17.      Halt(1);
18.   end;
19.   for i := 0 to GetMaxColor do
20.     begin
21.       SetColor(i);
22.       Circle(GetMaxX div 2, GetMaxY div 2, i+1);
23.     end;
24.   readln;
25.   CloseGraph;
26. end.
```



2. ábra

640×480-as, 16 színű EGAVGA, illetve 1024×768-as felbontású, 256 színű BGI grafikus üzemmódok

Mivel az **SVGA256.BGI** nem szabványos grafikus meghajtó (nem a Borland írta, hanem letölthető pl. a *http://pascal.sources.ru/graph/ svga256t.htm* honlapról), ezt így kell inicializálni és használni pl. 256 színű koncentrikus körök kirajzolására:

```
1.  program SVGA256;
2.
3.  uses Graph;
4.
5.  {$F+}
6.  function DetectSVGA256: integer;
7.  begin
```

```
8.    { 0: 320x200x256
9.      1: 640x400x256
10.     2: 640x480x256
11.     3: 800x600x256
12.     4: 1024x768x256 }
13.   DetectSVGA256 := 4;
14. end;
15. {$F-}
16.
17. var
18.   GraphMode, GraphDriver, GrErr: integer;
19.   i: byte;
20. begin
21.   GraphDriver := InstallUserDriver('SVGA256',
22.                                     @DetectSVGA256);
23.   GraphDriver := Detect;
24.   InitGraph(GraphDriver, GraphMode, '');
25.   GrErr := GraphResult;
26.   if GrErr <> grOK then
27.    begin
28.      writeln('Graphics error: ',
29.              GraphErrorMsg(GrErr));
30.      readln;
31.      Halt(1);
32.   end;
33.   for i := 0 to GetMaxColor do
34.     begin
35.       SetColor(i);
36.       Circle(GetMaxX div 2, GetMaxY div 2, i+1);
37.     end;
38.   readln;
39.   CloseGraph;
40. end.
```

Függvények, eljárások

a.) *Grafikus módot inicializáló eljárások*

`procedure` **`DetectGraph`**`(var GraphDriver, GraphMode: integer);`

Megvizsgálja a hardvert, és megállapítja a grafikus kiépítettségét. A `GraphDriver`-ben kapjuk meg a grafikus meghajtó számát, a `GraphMode`-ban pedig a használható legnagyobb felbontású üzemmód számát.

`procedure` **`InitGraph`**`(var GraphDriver, GraphMode: integer; PathToDriver: string);`

Inicializálja a grafikus rendszert és átállítja a hardvert (képernyőt) grafikus módra. A `PathToDrive` a **.BGI** állomány elérési útvonalát jelenti.

`function` **`GetDriverName`**`: string;`

Megadja az aktuális grafikus képernyőmeghajtó nevét (pl. EGAVGA).

`function` **`GetGraphMode`**`: integer;`

Visszaadja az aktuális grafikusmód kódját.

`function` **`GetModeName`**`(ModeNumber: Integer): string;`

A függvény értéke a bemeneti paraméterhez mint kódhoz tartozó grafikus mód teljes neve.

`function` **`GetMaxMode`**`: integer;`

Az aktuálisan betöltött grafikus meghajtó legnagyobb felbontású üzemmódjának számát adja vissza.

```
procedure GetModeRange(GraphDriver: integer; var LoMode, HiMode: integer);
```

A megadott grafikus meghajtó kódjához (`GraphDriver`) tartozó grafikus üzemmódok közül a legkisebb és legnagyobb értékűt adja vissza.

```
procedure GraphDefaults;
```

A grafikus kurzort a bal felső sarokba teszi és alaphelyzetbe állítja a grafikus rendszert.

```
function GetMaxX: integer;
```

A grafikus képernyő utolsó oszlopának számát adja vissza.

```
function GetMaxY: integer;
```

Megadja a grafikus képernyő utolsó sorának számát.

```
function GetX: integer;
```

Az aktuális képpont (CP – *Current Position*) vízszintes koordinátáját adja vissza.

```
function GetY: integer;
```

Az aktuális képpont (CP) függőleges koordinátáját adja vissza.

```
procedure SetGraphBufSize(BufSize: word);
```

Megváltoztatja az alapértelmezett grafikus buffer méretét. A belső buffer mérete `BufSize`-nak definiálódik a heap-en az `InitGraph` eljárás hívásakor. Alapértelmezés: 4 KB.

```
procedure CloseGraph;
```

Lezárja a grafikusmódot, visszatér a szöveges képernyőmódhoz.

```
procedure RestoreCrtMode;
```

Visszaállítja a grafikus rendszer installálása előtt használt képernyőmódot.

```
procedure SetGraphMode(Mode: integer);
```

Grafikus módba állítja a rendszert és letörli a képernyőt.

```
function InstallUserDriver(Name: string; AutoDetectPtr: pointer): integer;
```

A *Pascal* grafikus rendszerében előre nem telepített képernyőtípushoz új grafikus meghajtót telepít. `Name`: az új képernyőmeghajtó (**.BGI**) állomány neve, `AutoDetectPtr`: ha **`nil`**, automatikusan vizsgálja a hardvert; ha saját függvényt írunk rá, a címét kell itt megadnunk (pl. `@Vizsgal`)

```
function InstallUserFont(FontFileName: string ): integer;
```

Új fontot telepít. Új fontnak nevezzük azt a fontot, amely még nincs beépítve a BGI rendszerbe. Ezt egy **.CHR** állomány tárolja.

```
function RegisterBGIDriver(driver: pointer): integer;
```

E függvény segítségével a grafikus rendszerben nem szereplő képernyőtípusokon is dolgozhatunk. A függvény a grafikus rendszer részévé tesz egy **.BGI** driver állományt, amely egy grafikus képernyőt kezel. A paraméterben a heap-ben a meghajtónak lefoglalt terület kezdőcímét kell megadni. Az `InitGraph` használata előtt mindig alkalmazni kell, ha nincs előredefiniált grafikus meghajtó a képernyőhöz. A függvény visszatérési értéke a meghajtó száma lesz.

```
function RegisterBGIFont(Font: pointer): integer;
```

Olyan betűtípus használatakor alkalmazzuk, amely nem része a grafikus rendszernek. A telepítendő új fontot először töltsük a memóriába (`Font` kezdőcímtől), majd ezzel a paraméterrel hívjuk meg a függvényt.

b.) *Grafikus hibakezelés*

```
function GraphResult: integer;
```

A függvény értéke a legutóbbi grafikus művelet hibakódja.

```
function GraphErrorMsg(ErrorCode: integer): string;
```

A függvény értéke a paraméterben megadott kódú grafikus hiba szövegét adja.

c.) *Ablak- és lapkezelő eljárások*

```
procedure SetVisualPage(Page: word);
```

Beállítja a látható képernyőt (amennyiben több van). Ez nem feltétlenül lesz aktív képernyőlap. Aktívvá a `SetActivePage` eljárással tehető egy képernyőlap.

```
procedure SetActivePage(Page: word);
```

Az aktív grafikus képernyőlapot állítja be. Ez nem feltétlenül lesz látható a képernyőn. Egy képernyőlap a `SetVisualPage` eljárással tehető láthatóvá.

```
procedure SetViewPort(x1, y1, x2, y2: integer; Clip: boolean);
```

Beállít egy aktuális képernyőablakot a grafikus képernyőn. Az (`x1`, `y1`) és (`x2`, `y2`) definiálják az ablak bal felső és jobb alsó sarkait. A `Clip` a vágás állapotát adja meg. Ha `true`, a kiírás az ablak szélén túl nem folytatódik.

```
procedure GetViewSettings(var ViewPort: ViewPortType);
```

Visszatérési rekordja a grafikus képernyőn definiált aktuális ablak koordinátáit és vágási paramétereit tartalmazza.

```
procedure ClearDevice;
```

Letörli az aktív grafikus képernyőt, és alapállapotba (0, 0) helyezi a grafikus kurzort. A képernyő háttérszínű lesz.

```
procedure ClearViewPort;
```

Letörli az aktuális grafikus ablakot, a grafikus kurzort pedig a (0, 0) helyre teszi. Az ablak háttérszínű lesz.

```
procedure GetAspectRatio(var Xasp, Yasp: word);
```

A képernyő vízszintes és függőleges képméretarányát, azaz a képszélességet, képfelbontást adja vissza. A képméretarány (`Xasp : Yasp`).

```
procedure SetAspectRatio(Xasp, Yasp: word);
```

Beállítja az aktuális képméretarányt megadó módosító tényezőt.

d.) *Grafikus kurzor mozgatás*

```
procedure MoveTo(X, Y: integer);
```

Az aktuális pozíciót az (`X`, `Y`) koordinátájú pontra teszi.

```
procedure MoveRel(Dx, Dy: integer);
```

Az aktuális pozíciót eredeti helyzetéből relatívan mozgatja a grafikus képernyőn. Ha az aktuális pozíció az (X, Y), az ((X + `Dx`), (Y + `Dy`)) koordinátájú helyre mozgatja.

e.) *Pontok*

```
procedure PutPixel(X, Y: integer; Pixel: word);
```

Az (`X`, `Y`) koordinátájú képpontot `Pixel` színűre festi.

```
function GetPixel(X, Y: integer);
```

Az (`X`, `Y`) koordinátájú pont színét adja vissza.

f.) *Vonalak*

```
procedure Line(x1, y1, x2, y2: integer);
```

Az (`x1`, `y1`) pontból szakaszt húz (`x2`, `y2`)-be.

```
procedure LineTo(X, Y: integer);
```

Az aktuális pozíciótól (CP) (`X`, `Y`)-ba szakaszt húz.

```
procedure LineRel(Dx, Dy: integer);
```

Az aktuális CP-től kezdve relatívan rajzol, majd a CP-t az új pozícióra állítja. A vonal a CP-ből (x_0, y_0) megy az (x_1, y_1) pontig, ahol $x_1 = x_0 + \texttt{Dx}$, $y_1 = y_0 + \texttt{Dy}$.

```
procedure SetLineStyle(LineStyle: word; Pattern: word;
Thickness: word);
```

Beállítja az aktuális vonalvastagságot és színt.

```
procedure GetLineSettings(var LineInfo: LineSettingsType);
```

Információt ad az aktuális vonalmintáról, stílusról és vonalvastagságról, ahogy azt a `SetLineStyle` definiálta.

```
procedure SetWriteMode(WriteMode: integer);
```

Az egyenesek rajzolásának módját állítja be. (`XORPUT`, `ANDPUT`, `ORPUT`, stb.)

g.) *Körök, körívek és más görbék*

```
procedure Circle(X, Y: integer; Radius: word);
```

A **`SetColor`**-ral beállított aktuális színnel egy (`X`, `Y`) középpontú, `Radius` sugarú kört rajzol.

```
procedure Arc(X, Y: integer; StAngle, EndAngle, Radius: word);
```

Körívet rajzol (nem szükségszerűen kört). A körív az `StAngle` (kezdő szög)-től az `EndAngle`-ig tart, `Radius` sugárral és (`X`, `Y`)-t használva középpontul.

```
procedure Ellipse(X, Y: integer; StAngle, EndAngle: word;
XRadius, YRadius: word);
```

Megrajzolja egy ellipszis körvonalát. A körvonalat az `StAngle` szögtől `EndAngle`-ig rajzolja `XRadius` nagytengelyű és `YRadius` kistengelyű sugárral az (`X`, `Y`) középpontból.

```
procedure GetArcCoords(var ArcCoords: ArcCoordsType);
```

A legutóbbi **`Arc`** utasítással megrajzolt kör vagy ellipszis középpontját, és az ív kezdő és végpontját adja vissza.

```
procedure PieSlice(X, Y: integer; StAngle, EndAngle, Radius:
word);
```

Megrajzol és kitölt egy körcikket. Az (`X`, `Y`) a középpont. A szelet `StAngle` szögtől `EndAngle`-ig tart, `Radius` sugárral.

```
procedure Sector(X, Y: integer; StAngle, EndAngle, XRadius,
YRadius: word);
```

Megrajzol és kitölt egy ellipszis cikkelyt. A változók jelentését lásd az `Ellipse` ill. a `PieSlice` eljárásnál.

```
procedure FillEllipse(X, Y: integer; XRadius, YRadius: word);
```

Megrajzol egy kitöltött ellipszist. (`X`, `Y`) a középpont; `XRadius` és `YRadius` a függőleges és vízszintes sugarak.

h.) *Sokszögek, satírozások*

```
procedure Rectangle(x1, y1, x2, y2: integer);
```

Megrajzolja egy téglalap körvonalát az aktuális színnel és vonalstílussal. A téglalap bal felső és jobb alsó sarkát az (`x1`, `y1`) és (`x2`, `y2`) koordináták adják meg.

```
procedure Bar(x1, y1, x2, y2: integer);
```

Téglalapot rajzol az aktuális kitöltési stílussal és színnel. Az (`x1`, `y1`) a téglalap bal felső sarkának, az (`x2`, `y2`) pedig a jobb alsó sarkának koordinátáit tartalmazza.

```
procedure Bar3D(x1, y1, x2, y2: integer; Depth: word; Top:
boolean);
```

3 dimenziós téglatestet rajzol az aktuális kitöltési stílussal és színnel. A téglatest első lapja olyan, mintha a `Bar` eljárással az `x1`, `y1`, `x2`, `y2` paraméterekkel

rajzoltuk volna meg. A `Depth` paraméterbe a téglatest mélységét kell írni. A `Top` azt határozza meg, hogy meg kell-e rajzolni a test felső lapját vagy sem.

```
procedure DrawPoly(NumPoints: word; var PolyPoints);
```

Megrajzolja egy sokszög körvonalát az aktuális színnel és vonalstílussal. A `NumPoints` a csúcsok számát határozza meg, amelyek koordinátái a `PolyPoints`-ban vannak tárolva. Egy koordináta két word-ból áll, egy `X` és egy `Y` értékből.

```
procedure FillPoly(NumPoints: word; var PolyPoints);
```

Megrajzol egy kitöltött sokszöget. A `NumPoints` a sokszög csúcsainak számát adja meg, a csúcsok koordinátáit pedig a `PolyPoints` paraméterbe kell tenni. Egy csúcs koordinátája két word-ból áll, az egyik az `X`, a másik az `Y`.

```
procedure FloodFill(X, Y: integer; Border: word);
```

Egy körülhatárolt területet a megadott színnel és mintával tölt ki. Az (`X`, `Y`) koordinátájú pontnak benne kell lennie a területben. A `Border` a színezendő területet határoló szín kódja.

```
procedure SetFillStyle(Pattern: word; Color: word);
```

Beállítja a kitöltési mintát és színt. A `Pattern` a minta számát, a `Color` pedig a minta színét tartalmazza.

```
procedure GetFillSettings(var FillInfo: FillSettingsType);
```

Az aktuális színt és töltési mintát adja vissza, amit a `SetFillStyle` vagy `SetFillPattern` állított be legutoljára.

```
procedure SetFillPattern(Pattern: FillPatternType; Color: word);
```

Egy felhasználó által definiált kitöltési mintát állít be. A `Color` a minta színét adja meg.

```
procedure GetFillPattern(var FillPattern: FillPatternType);
```

Az aktuálisan kiválasztott tónus és minta értékét adja vissza, amelyeket a `SetFillStyle` vagy a `SetFillPattern` parancsokkal állítottunk be.

i.) *Képmentés és visszaállítás*

```
function ImageSize(x1, y1, x2, y2: integer): word;
```

A megadott ablak (vagy terület) byte-okban mért memóriabeli helyfoglalását adja vissza. A mérendő téglalap bal felső és jobb alsó sarkát határozzák meg a paraméterek.

```
procedure GetImage(x1, y1, x2, y2: integer; var BitMap);
```

A kijelölt terület bittérképét bufferbe menti. Az (`x1`, `y1`) és (`x2`, `y2`) adják meg a másolandó terület bal felső és jobb alsó sarkát. A bittérképet a `BitMap` változóba menti.

```
procedure PutImage(X, Y: integer; var BitMap; BitBlt: word);
```

Egy terület tartalmát visszatölti a képernyőre. A visszatöltés a `BitMap` típus nélküli változóból történik egy olyan téglalapba, amelynek bal felső sarka (`X`, `Y`). A `BitBlt` változóval a visszatöltés módját definiálhatjuk (`XorPut`, `NormalPut`, `AndPut`, `OrPut`, stb.).

j.) *Szövegkezelés*

```
procedure SetTextStyle(Font, Direction: word; CharSize: word);
```

Grafikus módban beállítja a szöveges kiírás paramétereit (stílusát). A `Font` a betűtípus száma, a `Direction` a kiírás iránya, a `CharSize` pedig a kiírás mérete.

```
procedure SetUserCharSize(MultX, DivX, MultY, DivY: Word);
```

Megváltoztatja a grafikus betűk szélességét és magasságát. Az aktuális betűtípus szélessége: `MultX / DivX`-szeresre, a magassága pedig `MultY / DivY`-szeresre növekszik.

```
procedure SetTextJustify(Horiz, Vert: word);
```

Ez az eljárás a következő szövegkiírások pozicionálásának fajtáját állítja be. A `Horiz` a vízszintes, a `Vert` a függőleges igazítás kódját tartalmazzák. Ezek a kódok az igazító konstansok is lehetnek.

```
procedure GetTextSettings(var TextInfo: TextSettingsType);
```

A `SetTextStyle` és `SetTextJustify` eljárásokkal beállított szövegstílusról adja meg a következő információkat: szövegfont (betűtípus), irány, méret, pozicionálás.

```
function TextHeight(TextString: string): word;
```

A függvény értéke a paraméterben megadott szöveg képpontokban mért magassága.

```
function TextWidth(TextString: string): word;
```

A függvény értéke a paraméterben megadott szöveg képpontokban mért szélessége.

```
procedure OutText(TextString: string);
```

Az aktuális pozícióba szöveget ír ki a grafikus képernyőn.

```
procedure OutTextXY(X, Y: integer; TextString: string);
```

Kiír egy stringet a grafikus képernyőre az (`X`, `Y`) ponttól kezdve.

k.) *Színhasználat*

```
procedure GetDefaultPalette(var Palette: PaletteType);
```

Az adott képernyőtípusnak megfelelő alapértelmezett palettát leíró rekordot adja vissza. Ezt a rekordot az `InitGraph` állítja be.

```
procedure SetPalette(ColorNum: word; Color: shortint);
```

Az aktuális paletta `ColorNum` színét `Color` színűre cseréli.

```
procedure SetAllPalette(var Palette);
```

Lecseréli az aktív paletta összes színét a megadott paletta színeire. Csak grafikus módban, akkor is csak a következő grafikus meghajtóknál alkalmazható: EGA, EGA64, VGA. Az IBM8514 és VGA256 színes módjában nem használható.

```
procedure GetPalette(var Palette: PaletteType);
```

Az aktuális palettát és méretét adja vissza.

```
function GetPaletteSize: integer;
```

A grafikus üzemmód palettájának méretét, azaz a színek számát adja vissza.

```
function GetMaxColor: word;
```

A **`SetColor`** eljárással beállítható legmagasabb értékű szín kódját adja meg.

```
procedure SetBkColor(Color: word);
```

Beállítja az aktuális háttérszínt. Csak az aktuális paletta színeiből választhatunk.

```
procedure SetColor(Color: word);
```

Beállítja az aktuális rajzoló színt. Csak az aktuális paletta színeiből választhatunk.

```
function GetBkColor: word;
```

Visszaadja az aktuális háttérszínt.

```
function GetColor: word;
```

Visszaadja az aktuális színt, melyet a legutóbbi `SetColor`-ral definiáltunk.

Kovács Lehel