# SDL gyorstalpaló

A babók szeretik a játékokat. Mi is...

A linuxos játékok egyre terjednek. Részben a mostanában beindult linuxos multimédia-fejlesztéseknek köszönhetően, részben pedig egyszerűen azért, mert a babók szeretik a játékokat. Az utóbbi néhány évben számos kitűnő linuxos multimédiás eszköz látott napvilágot. Ilyen például a GGI grafikus csatoló vagy az ALSA hangrendszer. Az utóbbi időben az SDL programkönyvtár is fejlődésnek indult.

Az SDL egy általános célú multimédia-programozási könyvtár, ami gyors és hordozható elérést szolgáltat a grafikához, a hanghoz, a bemeneti eszközökhöz, a szálkezeléshez és az OpenGL megjelenítéshez. Az SDL könyvtár magja, több Unix-változat mellett átvihető BeOS, MacOS és Win32 rendszerekre is. Ezáltal kitűnő választás lehet azok számára, akik a hatékonyság feláldozása nélkül szeretnének felületfüggetlen játékokat készíteni.

Más multimédia eszköztárakkal szemben, az SDL nem közvetlenül a géppel tart kapcsolatot, inkább egy réteget képez az alkalmazás és az alatta lévő rendszer között. Például az SDL grafikus rendszere képkockatárat (frame buffer) vagy az X11-et használ Linux alatt, viszont DirectDraw-t Windows alatt. Az SDL API-felülete egyik esetben sem változik, az alkalmazásnak nem kell törődnie azzal, ami a háttérben zajlik. Egy gondosan felépített SDL-alkalmazást egyetlen gyors újrafordítással át lehet vinni más felületre.

Ebben a cikkben az SDL video-API világában teszünk egy körutat, egészen az alapoktól kezdve. Azt is bemutatjuk, miképpen kell adatokat fogadni a billentyűzetről. E cikk legnagyobb része kivonat a *John Hill* Programming Linux Games című, készülő művéből (No Starch Press and Loki Entertainment Software, 2001 elején várható).

## Az SDL beszerzése

Az SDL ingyenes csomag (az LGPL alá tartozik), és letölthető a csoport honlapjáról ➲http://www.linsdl.org. A jelenlegi SDL könyvtáron kívül, ez a honlap számtalan példaforráskódot, bemutatópéldányt, játékot és kiegészítést is tartalmaz. Az SDL-t könnyű telepíteni, de ennek megkönnyítésére a honlap futtatható állományokat is kínál a legismertebb felületekhez.

## Az SDL tervezési alapgondolata

Ha valaki dolgozott már a Microsoft DirectX eszköztárával, észrevehette, hogy ahhoz képest az SDL egy aprócska könyvtár.

A rendszermaghoz tartozó könyvtár (kernel library; magkönyvtár) forráskódja hat megabájt alatt van, ebbe pedig már sok olyan kódot is belevettünk, amelyek Linux alatt soha nem kerülnek fordításra. De ez ne tévesszen meg senkit. Ez a hat megabájt jól kihasznált, és a SDL magkönyvtár szinte mindent elérhetővé tesz számunkra, ami csak egy kiemelkedő minőségű linuxos játékhoz vagy médialejátszóhoz kellhet. Továbbá, a honlap otthont ad számos kiegészítő könyvtárnak is, amelyek olyan további szolgáltatásokat nyújtanak, mint például a képek betöltése és a fejlett hangkeverés. Ezeket a lehetőségeket a magkönyvtártól elkülönítve tartják, ennek köszönhetően az SDL kicsi és könnyen megismerhető marad. Az SDL könyvtár több rész-API-t tartalmaz, ami által felületfüggetlen támogatást nyújt a videó-, hang-, bemenet-, szálkezelés, OpenGL-leképezés és számos olyan további képesség eléréséhez, amit a játékírók igazán értékelni tudnak. Sajnos, nincs elég helyünk, hogy mindent bemutassunk, ezért most csak a videoprogramozásra és a bemenetkezelésre szorítkozunk. Ez az a két dolog, amire leginkább szükség lehet, hogy az első lépéseket megtegyük az SDL-ben.

## Az SDL video-API

Az SDL video-API egyetlen célja, hogy megfelelő videoeszközt találjon, és beállítsa a program számára. Miután előkészítette a megjelenítőt (készített egy ablakot vagy átkapcsolta a videokártyát a megadott módba), az SDL félreáll az útból, mindössze néhány alapvető függvényt nyújt a képponttömbök mozgatásához. Az SDL nem rajzoló eszközkészlet: az már nem az SDL dolga, hogy az előkészítés után mit teszünk a megjelenítőeszközzel.

Az SDL úgynevezett felületeket (SDL_Surface típusú szerkezeteket) használ a grafikus adatok megjelenítéséhez. A felület egy egyszerű memóriatömb, ami téglalap alakú, képpontokból álló terület tárolására használható. Minden felületnek van szélessége, magassága és egy adott képpontformátuma (erről később bővebben is szó lesz). Az SDL a képállományokat közvetlenül felületekbe tölti, és a képernyő is felületnek tekinthető (jóllehet meglehetősen kivételes felületnek).

A felületek legfontosabb tulajdonsága, hogy nagyon gyorsan lehet őket egymásra másolni. Az egyik felület képpontjait át lehet helyezni egy másik felület azonos méretű téglalap alakú területére. Ezt a műveletet blitnek (block image transfer; képrészletátvitel), vagy részletátvitelnek nevezik. A részletátvitel igen fontos eszköze a játékprogramozásnak, mivel lehetővé teszi, hogy teljes képeket előre rajzoljanak meg (ezt gyakorta egy művész készíti el valamilyen rajzolóprogram segítségével). Mivel a képernyő ugyanolyan felület, mint bármely másik, egész képeket lehet a képernyőre küldeni egyetlen részletátvitel segítségével.

Az SDL általános függvényeket szolgáltat a felületek közti gyors részletátvitelhez, sőt, a különböző formátumot használó felületek közötti átalakítást is futás közben végzi el.

## A megjelenítés beállítása

Mielőtt tömböket dobálhatnánk a képernyőre, előbb be kell állítanunk az SDL könyvtár alapértékeit, és át kell váltanunk a megfelelő módba. Vessünk egy pillantást az első listára, ami a „Szia Világ!” SDL-es megfelelője.

Ez a program behívja az `SDL.h` fejlécfájlt, ami az SDL mesterfejléce. Minden SDL-alkalmazásnak hivatkoznia kell erre a

fejlécre. A program ezek után beolvas még két szabványos fejlécfájlt a printf és az atexit függvények eléréshez. Az `SDL_Init` meghívásával kezdünk, amivel az SDL-t alaphelyzetbe állítjuk *(lásd 1. listát)*. Ez a szolgáltatás értékként egy `VAGY` művelettel összeállított értéklistát vár, ami megmutatja számára, hogy mely alrendszereket kell üzembe állítania. Mivel bennünket jelenleg csak a videoalrendszer érdekel, az `SDL_INIT_VIDEO` értéket adjuk át (ha például hangot is szeretnénk, a szolgáltatást ezzel az értékkel kellene meghívni: `SDL_INIT_VIDEO | SDL_INIT_AUDIO`). Hacsak valamilyen végzetes hiba nem történik, ez a függvény nullával tér vissza. A C atexit rendszerét használjuk arra, hogy kilépéskor meghívjuk az `SDL_Quit` függvényt. Ez adja meg az esélyt arra, hogy az SDL megfelelően leálljon (ami különösen akkor fontos, ha egy teljes képernyőn futó alkalmazás omlana össze).

Ezután a `SDL_SetVideoMode` függvényt használjuk arra, hogy a megjelenítő tudomására hozzuk az alkalmazandó felbontást (ez jelen esetben 640×480 képpont) és színmélységet (16 bites csomagolt képpontok). Azonban van itt egy kis csalafintaság: az SDL megpróbálja ugyan beállítani a kért videomódot, de előfordul, hogy nem jár sikerrel. Ilyenkor az SDL nem szól semmit, hanem saját maga emulálja a kért módot. Ez többnyire elfogadható, hiszen az emulációs kód viszonylag gyors, és általában nekünk sem feladatunk a különféle videomódokkal foglalkozni. Az `SDL_SetVideoMode` egy felületmutatót ad vissza, ami a képernyőnek felel meg. Ha valami hiba történt, a szolgáltatás `NULL`-t ad vissza.

Utolsó lépésként hírt adunk a sikerről, és kilépünk. A C könyvtár automatikusan meghívja a `SDL_Quit` függvényt (mivel már bejegyeztük az atexit segítségével), és az SDL az eredeti helyzetbe állítja vissza a videomegjelenítőt. (Közvetlenül is meghívhatjuk az `SDL_Quit` függvényt, amennyiben az alkalmazásból történő kilépés előtt szeretnénk lezárni a rendszert. Nem okoz gondot, ha egynél többször hívjuk meg.) Elkészítettünk egy SDL-alkalmazást, most már csak le kell fordítanunk. Helyes telepítést feltételezve, az SDL-alkalmazásokat könnyű felépíteni, mindössze néhány jelzőre (flag) és könyvtárra van szükségünk. Az alap SDL-változat tartalmaz egy sdl-config nevű programot (hasonlóan a gtk-config vagy a glib-config programokhoz, amelyek a GTK+ eszközkészlettel érkeznek), ezzel tudjuk előállítani a gcc-hez szükséges megfelelő parancssori kapcsolókat. Az `sdl-config -cflags` egy kapcsolólistát készít, amelyet aztán átadhatunk a fordítónak. Az `sdl-config --libs` pedig a használandó könyvtárak listáját állítja elő. A parancssorba helyezéshez akár fordított aposztrófos behelyettesítést (`` `parancs` ``) is alkalmazhatunk. Amennyiben az SDL már telepítve van rendszerünkön, a példát a következő parancssorral fordíthatjuk le:

```
$ gcc sdltest.c -o sdltest `sdl-config --cflags
--libs`
```

## Képpontok közvetlen kirajzolása

Egy SDL-felületre adatokat tenni igen könnyű feladat. Minden `SDL_Surface` szerkezet tartalmaz egy `pixel` tagot. Ez egy void mutató a nyers, grafikus képre, ahová akár közvetlenül is írhatunk, amennyiben tudjuk, milyen típusú képpontok alkotják a felületet. Mielőtt hozzányúlnánk az adatokhoz, először meg kell hívnunk az `SDL_LockSurface` függvényt (hiszen néhány felület különleges memóriaterületeken helyezkedik el, és éppen ezért különleges elbánást igényel). Amikor végeztünk a felület módosításával, a felszabadításához meg kell hívnunk az `SDL_UnlockSurface` függvényt. A kép szélességét és magasságát a szerkezet `w` és `h` eleme adja meg, a képpontformátumot pedig az `SDL_PixelFormat` típusú elem. Az SDL gyakorta emulál nem szabványos képernyőfelbontásokat magasabb felbontású módban, az `SDL_PixelFormat` szerkezet `pitch` eleme viszont mindig a képkockatár (frame buffer) valódi szélességét adja meg. Az eltolások számításához minden esetben a `pitch` elemet használjuk a `w` elem helyett, különben a program nem fog helyesen működni bizonyos megjelenítőkön.

A *2. listán* bemutatott program az `SDL_PixelFormat` adatot használja arra, hogy önálló képpontokat rajzoljon a képernyőre. A bemutató céljára tizenhat bites (hicolor) módot választottunk, de más képernyőmódokat is éppoly könnyű lenne programozni.
A programot a megjegyzések magától értetődővé teszik, de néhány dolog talán nem annyira nyilvánvaló. Ez a program egy nagyon általános eljárást alkalmaz az SDL által felismerhető hicolor képpontok előállítására. Természetesen írhattunk volna külön eljárást minden egyes hicolor adatformátumhoz (ami nyilván gyorsabb lenne), csakhogy ez sok különmunkával járna, ezzel szemben csak kis mértékben növelné a hatékonyságot. Ugyan a hicolor 565 (5 vörös bit, 6 zöld bit, 5 kék bit) a legszélesebb körben használt képpontformátum, és így ésszerű lenne erre kiélezni a kódot, mindazonáltal az 556 és az 555 formátum sem ritka. Ráadásul semmi sem biztosítja azt, hogy a bitmezők vörös, zöld, kék sorrendben következnek. A CreateHicolorPixel eljárásunk azonban az `SDL_PixelFormat` szerkezetben található adatoknak megfelelően kezeli ezt a kérdést. Például az eljárás a szerkezet `Rloss` elemét használja annak megállapítására, hány bitet kell eldobni a 8 bites vörös összetevőből, majd az `Rshift` elem segítségével állapítja meg, hol helyezkednek el a vörös bitek a 16 bites képpontértéken belül.
Egy másik fontos dolog, amiről beszélni kell, az `SDL_UpdateRect` függvény. Amint azt már korábban említettük, az SDL néha emulálja a videomódokat, ha a videokártya nem képes azokat megjeleníteni. Tegyük fel, a videokártya nem támogatja a kért 24 bites módot, akkor az SDL például 16 bites módba vált, és visszaad egy hamis 24 bites képkockatár-beállítást. Ez lehetővé teszi számunkra, hogy gond nélkül folytassuk a programot, az SDL pedig futásidőben fogja 24 bitről 16 bitre átültetni a képpontokat (ami némi teljesítménycsökkenést okoz).
Az `SDL_UpdateRect` függvény arról tájékoztatja az SDL-t, hogy az adott képernyőterület tartalma megváltozott, és végre kell hajtania a megfelelő átalakításokat az adott terület megjelenítéséhez. Ha a programban nem használjuk ezt a függvényt, még megvan az esély rá, hogy a működni fog. Jobb azonban biztosra menni, és meghívni ezt a függvényt, valahányszor a képkockatárnak használt terület megváltozik.
Végül, ha lefuttatjuk a programot, feltűnhet, hogy ablakban fut, ahelyett, hogy a teljes képernyőterületet birtokba venné. Ha ezt szeretnénk elérni, az `SDL_SetVideoMode` hívásban a nulla helyére az `SDL_FULLSCREEN` állandót kell írnunk. Mindazonáltal legyünk óvatosak. A teljes képernyős alkalmazásokban előforduló hibákat nehezebb nyomon követni, és többnyire csúnyán összekavarnak mindent, amikor összeomlanak.

## Rajzolás részletátvitel segítségével

Láthattuk, miképpen lehet képpontokat közvetlenül a képernyőre rajzolni, és nincs is semmi ok, ami miatt ne lehetne ilyen módon akár egy teljes játékot megalkotni. Mégis, nagy adatmennyiség képernyőre küldésére létezik egy sokkal jobb módszer is. A következő példánk egy teljes felületet tölt be egy fájlból, majd egyetlen SDL-függvénnyel a képernyőre rajzolja azt. A program a *3. listán* látható.
Amint láthattad, az `SDL_LoadBMP` függvénnyel töltöttük a memóriába a bitképet. A függvény vagy egy, a képet tartalmazó `SDL_Surface` típusú mutatót ad vissza, vagy NULL-t, amennyiben a képet nem sikerült betölteni. A fájl sikeres betöltését követően, a bitkép egyszerű SDL-felületként kezelhető, amit a program képernyőre vagy bármilyen más felületre rajzolhat. A bitképek dinamikusan foglalt memóriát használnak, amit fel kell szabadítani, ha a továbbiakban már nincs rá szükség. Az `SDL_FreeSurface` függvény felszabadítja a bitkép által lefoglalt memóriaterületet.
Az `SDL_BlitSurface` függvény feladata az egyik felület másikra vitele, miközben képpont-átalakítást hajt végre az egyes formátumok között, amennyiben szükséges. Ez a függvény négy bemenő értéket



*1. lista:* A megjelenítő felkészítése

```
#include "SDL.h"
#include <stdio.h>
#include <stdlib.h>
int main()
{
    SDL_Surface *screen;
    /* Az SDL videorendszerének üzembe
       helyezése, majd a hibák lekérdezése */
    if (SDL_Init(SDL_INIT_VIDEO) != 0) {
                printf("Az SDL üzembehelyezése
                        nem sikerült: %s\n",
                        SDL_GetError());
                return 1;
    };
    /* Bizonyosodjunk meg, hogy az SQL_Quit
    meghívásra kerül a program futása végén! */
    atexit(SDL_Quit);
    /* kisérlet a 640x480-as módba váltásra */
    screen =
SDL_SetVideoMode(640,480,16,SDL_FULLSCREEN);
    if (screen == NULL) {
        printf("Nem tudunk a megfelelő módba
                váltani: %s\n",
                SDL_GetError());
        return 1;
    };
    /* Ha eddig eljutottunk, akkor minden
    működött */
    printf("Siker!\n");
    return 0;
}
```

vár: a forrásfelületet (ahonnan a képet másolni kell), egy `SDL_Rect` szerkezetet, ami a forrásból ténylegesen átvitelre kerülő téglalap alakú területet határozza meg, a célfelületet (ahová a kép kerül), és egy másik `SDL_Rect` szerkezetet, amely a célterületet adja meg. Ennek a két területnek azonos szélességűnek és magasságúnak kell lennie (mivel az SDL jelenleg még nem támogatja a torzítást), a területek kezdő x és y koordinátái azonban különbözhetnek.

## Színkulcsok és átlátszóság

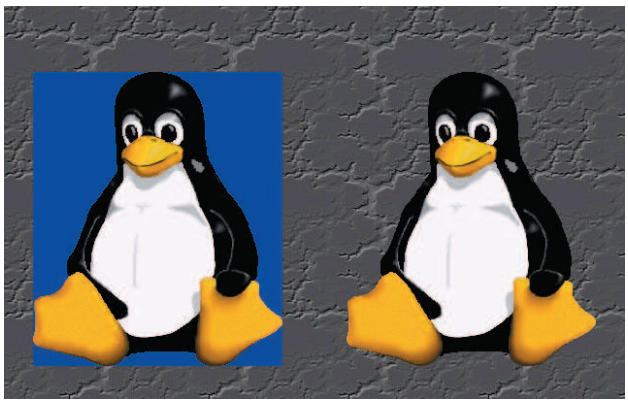
A játékokban gyakran van szükség az átlátszóság látszatára. Például van egy játékfigurát ábrázoló bitképünk valamilyen egyszínű háttér előtt, és szeretnénk kirajzolni a játék pályájára. Elég szörnyen nézne ki, ha a hátteret is kirajzolnánk, és a figura egyszínű dobozba zárva jelenne meg. Sokkal jobb lenne, ha csak azokat a képpontokat rajzolnánk ki, amelyek valóban a figura részei, nem pedig a háttéré. Ezt színkulcsos részátvitellel tehetjük meg. Az SDL ezt is támogatja, sőt, még a színkulcs tömörítéses gyorsítását (run-length colorkey acceleration) is lehetővé teszi számunkra (ez egy ügyes trükk a rajzolás meggyorsítására). Az RLE gyorsítás óriási sebességnövekedéshez vezethet színkulcsos munkáknál.
Alkalmazásuk azonban csak olyan bitképek esetében célszerű, amelyek a futás alatt nem változnak (mivel az RLE szerint kódolt kép megváltoztatásához előbb ki kell bontani, majd a változtatás után újra be kell csomagolni).
A színkulcs egy olyan képpontérték, amit a program átlátszó színnek határoz meg. Az SDL-ben ezt a SDL_SetColorKey függvénnyel lehet megadni. Azok a képpontok, amik megegyeznek a színkulccsal, nem másolódnak át a kép átvitelekor. A mi játékfigurás példánkban, a tömör hátteret kell színkulcsként megadnunk, így az nem jelenik meg. A színkulcsok révén egyszerűvé válik a téglalap alakú területen tárolt, de nem téglalap formájú tárgyak használata. A következő példánkban színkulcsos átvitelt fogunk használni arra, hogy Tux képét egy másik képre rajzoljuk rá (a lista az ➲ftp.scc.com/pub/lj/listings/issue81/ címről tölthető le). Tuxot egyöntetű kék szín előtt tároltuk, így a kék színt (RGB 0, 0, 255) fogjuk színkulcsként használni. Az összehasonlítás kedvéért színkulcs nélkül is kirajzoljuk ugyanazt a képet.

## Egyszerű billentyűzetkezelés

Az SDL a billentyűzeten található minden billentyűhöz „virtual keysym” kódokat rendel. Ezeket a számokat (valós egészeket) az operációs rendszer saját billentyűkódjaira alakítja (amelyek a billentyűzet által küldött kódokra hivatkoznak), az SDL azonban a színfalak mögött felügyel ezekre az átalakításokra. Minden virtuális keysymhez egy előfeldolgozó-állandót (preprocessor symbol) rendel. Például az ESCAPE billentyűnek az `SDLK_ESCAPE` állandó felel meg. Az érvényes állandók listája megtalálható az SDL leírásában.
Ezeket a kódokat használjuk, valahányszor csak közvetlenül szeretnénk lekérdezni valamely gomb állapotát (lenyomott, vagy felengedett), illetve az SDL ezeket adja át, amikor billentyűzeteseményt küld. A virtuális keysymeket az SDLKey adattípus határozza meg.
Mivel az eseménykezeléssel most nemigen foglalkozunk, közvetlenül kell lekérdeznünk a billentyűzet állapotát, valahányszor kíváncsiak vagyunk valamelyik billentyűre. A program pillanatfelvételt kérhet a teljes billentyűzetről egy vektor formájában. Az `SDL_GetKeyState` függvény egy mutatót ad vissza az SDL belső billentyűzet-állapot tömbjére, amit az `SDLK _keysym` állandók segítségével címezhetünk meg. Ezt a függvényt csak egyetlen egyszer kell meghívni, a mutató a teljes futásidő alatt érvényes marad. A vektor minden egyes eleme egy egyszerű Uint8-típusú jel, amely azt mutatja meg, hogy az adott billentyű lenyomott állapotban van-e. Időnként meghívjuk az `SDL_PumpEvents` függvényt, ezzel frissítve a vektor értékeit.

## Még, még, még!

Az induláshoz ennyi is elég az SDL-ről. Sok mindenen átsiklottunk, többek közt az animáción, az alfacsatorna használatán, valamint a hangkezelésen. Ha többet akarsz tudni ennek a könyvtárnak a programozásáról, a legjobb kiindulási hely az SDL Documentation



*2. lista:* Egyedi képpontok rajzolása a képernyőre

```
#include "SDL.h"
#include <stdio.h>
#include <stdlib.h>
Uint16 CreateHicolorPixel(SDL_PixelFormat *fmt,
                          Uint8 red, Uint8
                          green, Uint8 blue)
{
    Uint16 value;
    /* Az alábbi eltolások segítségével kiszá-
       mítjuk a szükséges 8 bites vörös, zöld és
       kék értékeket a 16 bites értékekből
    */
    value = ((red > fmt->Rloss) << fmt->Rshift) +
        ((green > fmt->Gloss) << fmt->Gshift) +
        ((blue > fmt->Bloss) << fmt->Bshift);
    return value;
}

int main()
{
  SDL_Surface *screen;
  Uint16 *raw_pixels;
  int x,y;

    /* A felkészítő kód jön. Létrehozunk egy
       256x256-os, 16 bites felületet, majd
       elmentjük a screen mutatóba. Lásd az előző
       példát. */
    /* A screen "zárolása", így közvetlenül raj-
       zolhatunk rá. */
     SDL_LockSurface(screen);
     /* Készítünk egy mutatót a videofelület
        i-memóriájához. */
     raw_pixels = (Uint16 *)screen->pixels;

    /* Most már nyugodtan írhatunk a videofelület-
       re. Egy szép átmenetes mintát rajzolunk, a
       vörös és a kék összetevők változtatásával.
    */
    for (x = 0; x < 256; x++) {
       for (y = 0; y < 256; y++) {
       Uint16 pixel_color;
       int offset;
       pixel_color =
         CreateHicolorPixel(screen->format, x,0,y);
       /* A módosítandó képpont memóriaeltolásának
          számítása. */
       offset = (screen->pitch/2 * y + x);
       raw_pixels[offset] = pixel_color;
       };
    };
   /* Készen vagyunk, úgyhogy megszüntetjük a
      zárolást. */
   SDL:_UnlockSurface(screen);
   /* értesítjük az SDL-t, hogy a képernyő tartal-
      ma megváltozott. Ez szükséges, ugyanis az
      SDL felülete nem a tényleges megjelenítő,
      hanem egy köztes réteg.
    */
    SDL_UpdateRect(screen,0,0,0,0);

    /* Pár másodpercig várunk, hogy a nézőnek
       legyen ideje ámuldozni. */
    SDL_Delay(3000);
    return 0;
}
```

*3. lista:* Nagy mennyiségű adat rajzolása a képernyőre

```
#include <SDL/SDL.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
   SDL_Surface *screen;
   SDL_Surface *image;
   SDL_Rect src,dest;
   /* Kezdeti kód, az előző példával azonos
      módon. */
   /* A bitkép beolvasása. Az SDL_LoadBMP egy
      mutatóval tér vissza, mely a képet tartal-
      mazó új felületre mutat. */
   image = SDL_LoadBMP("tux.bmp");
   if (image == NULL) {
       printf("Nem tudom betölteni a képet.\n");
       return 1;
   };
   /* Az SDL képátvitel (blitting) számára pon-
      tosan meg kell adni az átvinni kívánt adat
      mennyiségét. Ezt az src és dest  SDL_Rect
      szerkezetekkel adjuk meg. A két területnek
      azonos méretűnek kell lennie. Az SDL jelen-
      leg nem kezel torzításokat. */
   src.x = 0; src.y = 0;
   src.w = image->w; /* Az egész kép másolása */
   src.h = image->h;
   dest.x = 0; dest.y = 0;
   dest.w = image->w;
   dest.h = image->h;
   /* A bitkép kirajzolása a képernyőre. Hicolor
      módban vagyunk, ezért nem kell foglalkoz-
      unk a színpalettákkal. Képátvitelnél
      (blitting) nem kell zárolnunk a képernyőt,
      az SDL elintézi a zárolást. */
   SDL_BlitSurface(image,&src,screen,&dest);
   /* Utasítjuk az SDL-t, hogy frissítse az egész
      képernyőt. */
    SDL_UpdateRect(screen,0,0,0,0);
   /* Pár másodpercig várunk, hogy a nézőnek
      legyen ideje ámuldozni. */
    SDL_Delay(3000);
    /* A lefoglalt memória felszabadítása.. */
    SDL_FreeSurface(image);
    return 0;
}
```

Project honlapja a ➲http://www.libsdl.org. Esetleg benézhetsz a #sdl csatornára az ➲irc.openprojects.net-en, ahol valószínűleg szép számmal találsz majd több-kevesebb tapasztalattal rendelkező SDL-rajongókat.
Jó játékot és jó kódolást!

*John Hall* (overcode@lokigames.com) a Georgia Tech cégnél vezető kutatóinformatikus. Mindenféle linuxos játék érdekli. Amikor éppen nem önkívületben ül a billentyűzet előtt, gyakran lehet látni az egyetem környékén, amint pókszabású kedvenceit sétáltatja.