Játékprogramozás az SDL programkönyvtárral

Saját játékainkhoz is felhasználhatjuk a Tux Racer és a Civilization mögött megbújó programkönyvtárat.

Az SDL (Simple DirectMedia Layer, ➲ http://www.libsdl.org) egyszerű, de hatékony felületfüggetlen játék- és multimédia-fejlesztő programkönyvtár, amelyet ***Sam Latinga*** fejlesztett ki, amikor a Loki Software játékkészítő cégnél dolgozott. A Loki az SDL-t használta kereskedelmi játékainak fejlesztésekor. Az SDL-t a több operációs rendszerrel dolgozó játékkészítők igényeit szem előtt tartva fejlesztették ki, és többek között a következő játékok linuxos változatához használták: Maelstrom, Hopkins FBI, Civilization: Call to Power, Descent 2, MythII: Soulblighter, Railroad Tycoon II és Tux Racer. Az SDL honlapján több száz SDL-t használó játék és alkalmazás van felsorolva.
Az SDL hivatalosan támogatja a Linux, Windows, BeOS, Mac OS, Mac OS X, FreeBSD, OpenBSD, BSD/OS, Solaris és Irix operációs rendszereket. Működik Windows CE, AmigaOS, Atari, QNX, NetBSD, AIX, Tru64 Unix és SymbianOS környezetben is, de ezek az operációs rendszerek hivatalosan még nem támogatottak. Ebből következően az SDL segítségével megírt alkalmazás csekély erőfeszítéssel az összes felsorolt operációs rendszer alá átvihető. Az SDL a hordozható játék- és multimédiás alkalmazások fejlesztését minden ma használatos nagy operációs rendszerre támogatja.

Az SDL telepítése

A nem túl régi Linux-terjesztéseknek eleve része a teljes SDL. Például a saját Red Hat 8.0 rendszeremen nyolc programot is találtam a */usr/bin* könyvtárban, amelyek az SDL-től függtek. A következő parancsok segítségével megállapíthatjuk, hogy az SDL programkönyvtárak és a C/C++-fejlécállományok telepítve vannak-e a rendszerünkön:

```
locate SDL.h
locate libSDL
locate sdl-config
```

Ha a parancsok mindegyike azt jelzi, hogy az állomány megtalálható, akkor nagy valószínűséggel teljes SDL-telepítéssel bírunk, és csak arról kell meggyőződnünk, hogy elég friss-e. Az ***sdl-config*** program visszaadja az SDL változatszámát és a fordítóprogramnak megadandó kapcsolókat. Ha az ***sdl-config*** program telepítve van, adjuk ki az `sdl-config --version` parancsot, hogy megtudjuk a telepített SDL változatszámát. Ha az eredményül kapott változatszám 1.2.4-nél kisebb, telepítsük a programkönyvtár újabb változatát. A legtöbb nyílt forrású projekthez hasonlóan az SDL is folyamatos fejlesztés alatt áll, így ha felhasználjuk a saját fejlesztéseinkhez, érdemes rendszeresen figyelni az új változatok megjelenését, vagy az SDL-es levelezőlisták egyikéhez csatlakozva követni a frissítéseket.
Ha az SDL nincs telepítve, töltsük le és telepítsük. A terjesztések általában tartalmazzák az előre lefordított SDL-csomagokat, ezért a keresést itt kezdjük. Ha a csomagok kellően frissek, az a legegyszerűbb, ha a terjesztésünkhöz való ***dev*** vagy ***devel*** végződésű SDL-csomagokat telepítjük.
A cikkben bemutatott forrás mellett megtalálható az `sdl-install.sh` parancsállomány, ami letölti és telepíti az SDL 1.2.5-ös változatát és az összes bővítmény-programkönyvtárat. A parancsállományt rendszergazdaként kell futtatni abban a könyvtárban, amibe az SDL forrását tartani szeretnénk. Ha nem az `sdl-install.sh` parancsállományt használjuk, akkor a megadott weboldalakról le kell töltenünk a fájlokat, és kicsomagolás után a ***README*** állományokban leírt módon telepítenünk kell őket. Az új telepítést a következő paranccsal ellenőrizhetjük:

```
sdl-config --version
```

Ha a parancs nem fut le, vagy az éppen telepített változatnál régebbit jelez, akkor a telepítés sikertelen. Tapasztalataim szerint ez akkor következik be, ha nem követjük az utasításokat, vagy egy régi SDL-változat fenn van egy másik helyen már . Ha az ***sdl-config*** egynél több helyet sorol fel, akkor vagy töröljük a régi SDL-telepítést, vagy az újat telepítsük a régi helyére. Az `sdl-install.sh` állomány bemutatja, hogy a `./configure --prefix` használatával az SDL tetszőleges helyre telepíthető, de a legkönnyebb és legbiztonságosabb az alapértelmezett hely használata.
Az SDL leírása a ➲ http://www.libsdl.org/docs.php címen található meg. Az online leírás helye pedig az ➲ http://sdldoc.csn.ul.ie. A támogató programkönyvtárak leírása vagy a letöltési oldalukról érhető el, vagy a forráskódhoz mellékelik, esetleg a ***.h*** állományokba van beágyazva. Az SDL-hez példaprogramok is tartoznak, és a támogató programkönyvtárak jól használhatók a saját fejlesztéseink elindításához.

SDL-példaprogram

A ***bounce.cpp*** (elérhető az ➲ ftp.ssc.com/pub/lj/listings/issue110/6410.tgz címen, illetve a 49. CD Magazin/SDL könyvtárában) állomány egy játékprogram forráskódját tartalmazza. A bevitelt és a grafikát az SDL, a TrueType betűkészletek betöltését az ***SDL_ttf*** programkönyvtár intézi. Maga a játék egy kicsit több mint 1300 sor C++-kód. A csomag tartalmazza a forráskódot, a képeket, a TrueType betűkészletet, a `Makefile`-t, az `sdl-install-sh` állományt, valamint a játékban felhasznált betűkészlethez és a képekhez tartozó felhasználói szerződést. Több ideig eltarthat a játékhoz jogtisztán felhasználható betűkészleteket, képeket és hangokat találni, mint magát a játékot megírni.
Kezdjük meg az SDL tanulmányozását a Bounce játék forrásának letöltésével és kicsomagolásával (`tar -xzvf bounce.tar.gz`). Ezután futtassuk a `make` parancsot. A lefordított programot a `bounce` parancs kiadásával indíthatjuk.
A `bounce -fullscreen` parancs hatására a program teljes képernyős üzemmódban indul. A játék lényege, hogy a Föld kóborol a Naprendszerben, és fennáll a veszély, hogy a Napba zuhan. Feladatunk, hogy a Földet távol tartsuk a Naptól, amit úgy érhetünk el, hogy nekiütközünk a Holddal. Minden alkalommal pontot kapunk, ha a Földet eltaláljuk a Holddal, és a játék magának könyvel el egy pontot, ha a Föld összeütközik a Nappal. A játék célja az SDL képességeinek a bemutatása, nem pedig a világ legérdekesebb játékának a megalkotása.

Az SDL indítása

Az SDL-t el kell indítani, mielőtt használhatnák a képességeit. Erre szolgál az `SDL_Init()` függvény:

```
if (-1 == SDL_Init((SDL_INIT_VIDEO |
                    SDL_INIT_TIMER |
                    SDL_INIT_EVENTTHREAD)))
{
  ...
}
```

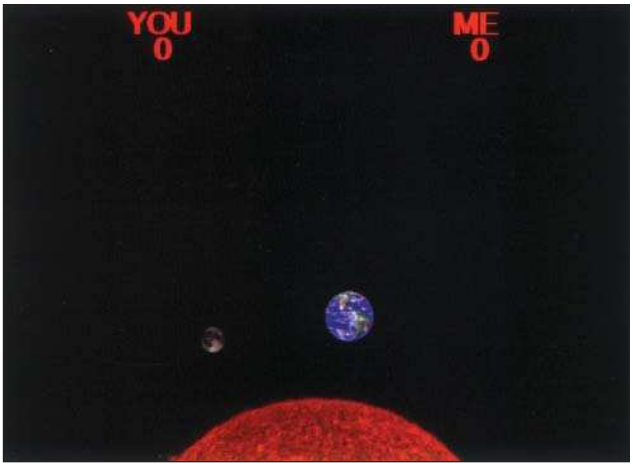
Az `SDL_Init()` függvénynek átadott érték adja meg az alrendszert, amit el kell indítani. Itt a képmegjelenítést, az időzítést és a szálakon alapuló eseménykezelést kapcsoltuk be. Használhattuk volna az `SDL_INIT_EVERYTHING` értéket is, ami az összes SDL-részt bekapcsolja, de célszerű csak azokat a részeket elindítani, amiket ténylegesen használunk a programban. Nincs értelme elindítani a botkormányt vagy a CD-ROM-ot kezelő részt, ha úgysem használjuk. Bármikor elindíthatjuk és leállíthatjuk az SDL egyes alrendszereit az `SDL_InitSubSystem()` és a `SDL_QuitSubSystem()` függvényekkel.
Fontos, hogy az SDL-t hívással leállítsuk az `SDL_Quit()`, még mielőtt a programunk leáll. Az `SDL_Quit()` az összes SDL-alrendszert leállítja, felszabadítja az SDL által használt rendszererőforrásokat, és helyreállítja a videomódot. Jó gyakorlat az `atexit()` használata, hogy biztosak legyünk benne, hogy az `SDL_Quit()` lefut a programunk befejeződésekor. Ha elmulasztjuk meghívni az `SDL_Quit()` függvényt, a számítógép furcsa videomódban maradhat.

A videomód beállítása

A videomód kiválasztásakor el kell döntenünk, hogy ablakban vagy a teljes képernyőn szeretnénk-e futtatni az alkalmazást. Azután meg kell adnunk az ablak vagy a képernyő méretét. Ha az ablak mellett döntünk, meg kell adnunk, hogy a felhasználó átméretezheti-e. Ezután ki kell választanunk a képernyő színmélységéhez való alkalmazkodás módját. A Bounce programban valami ilyesmit használtam:

```
options = SDL_ANYFORMAT |  SDL_FULLSCREEN;
screen = SDL_SetVideoMode(640, 480, 0, options);
```

Az első két érték a program ablakának vagy képernyőjének a szélességét és a magasságát adja meg képpontokban. Az adott szélességet és magasságot csak akkor használhatjuk teljes képernyős módban, ha az ***XF86Config-4*** állomány (bizonyos X-változatokban XF86Config) „screen” fejezetében szerepel a megadott méret. Ha a Bounce nem fut teljes képernyős üzemmódban a gépünkön, akkor a 640×480-as képernyőfelbontás valószínűleg nincs beállítva az ***XF86Config-4*** állományunkban. A harmadik érték adja meg a képpontokat leíró bitek számát. Ha nullára (0) állítjuk, akkor az SDL a pillanatnyi színmélységet használja. Jobban járunk, ha a játék a pillanatnyi színmélységhez alkalmazkodik, mert ekkor nem kell minden egyes géptípusról, amin a játékunk futhat, számon tartanunk, hogy támogatja-e a kívánt képpontformátumot.
Az utolsó érték segítségével részletes utasításokat adhatunk az SDL-nek a videomód beállításával kapcsolatban. Közel egy tucat lehetőség közül választhatunk. A Bounce programban az `SDL_ANYFORMAT` lehetőséget választottam, ami az SDL-t a lehető legjobb mód választására utasítja. Ez a lehetőség arra kényszeríti, hogy a kód minden lehetséges színmélységhez alkalmazkodjon, de egy kis többletkódolás árán még jobb



A Bounce játék

teljesítményt nyújt. Az `SDL_FULLSCREEN` lehetőség az SDL-t teljes képernyős üzemmódba kapcsolja.
Az `SDL_SetVideoMode()` által visszaadott érték mutató egy `SDL_Surface` szerkezetre. Ez a szerkezet írja le a képernyőt teljes részletességgel. De a `NULL` érték visszakapása nem azt jelenti, hogy mindent megkaptunk, amit akartunk. Ellenőrizzük a szerkezet jelzőbitmezőjét, hogy a kért lehetőségek működnek-e. Úgy tapasztaltam, hogy érdemesebb keveset kérni, és azzal dolgozni, ami van, mert így elkerülhető, hogy operációs rendszertől függő kódot építsünk a programba.
A videomód beállítása után az ablakcím és az ikonnév beállítására használjuk az `SDL_WM_SetCaption()` függvényt. Ez nem kötelező, de a program használata egy kicsit könnyebb lesz tőle:
`SDL_WM_SetCaption("Bounce", "Bounce")`

Az erőforrások betöltése

Mielőtt a Bounce elindulhatna, be kell töltenie és kezdeti értékekkel kell ellátnia a használt erőforrásokat. A Bounce programnak be kell állítania a színeket, be kell töltenie egy pár képet és a szövegek megjelenítésére használt betűkészletet. Mivel a videomódot az `SDL_ANYFORMAT`-ra állítottuk, az összes erőforrást át kell alakítani olyanra, hogy tetszőleges képernyőformátumhoz megfeleljen. A következő két kódsor egy vörös képpontot hoz létre a szükséges formátumban:

```
SDL_PixelFormat *pf = screen->format;

int red = SDL_MapRGB(pf, 0xff, 0x00, 0x00);
```

Az `SDL_PixelFormat` szerkezet a képernyő képpontjainak leírása, és az `SDL_MapRGB()` a képpontot a szabványos 24-bites RGB színformátumból egy olyan képpontértékre alakítja át, ami a kívánt színt jeleníti meg a képernyőn.
A képek betöltése egy kissé bonyolultabb:

```
SDL_Surface *s0, *s1;

s0 = SDL_LoadBMP(name);
s1 = SDL_DisplayFormat(s0);
SDL_SetColorKey(s1,
                (SDL_SRCCOLORKEY |
                 ➥SDL_RLEACCEL),
                black);
SDL_FreeSurface(s0);
```

Az SDL magja tartalmazza az `SDL_LoadBMP()` függvényt, ami egy ***.bmp*** formátumú képet tölt be egy `SDL_Surface` szerkezetbe. Az `SDL_image` sok más képformátum betöltéséhez kínál függvényeket. A kép abban a formátumban lesz, amelyben létrehozták. A megjelenítési formátumra az `SDL_Display_Format()` függvény segítségével alakíthatjuk át. Az `SDL_SetColorKey()` arra szolgál, hogy az SDL-lel közöljük, a kép másolásakor ne vegye figyelembe a fekete képpontokat. Amikor a Föld képét mozgatjuk a képernyőn, a fekete képpontokat nem másoljuk, csak a kerek Föld belső képpontjai érintettek. Az `SDL_RLEACCEL` jelzőbit arra utasítja az SDL-t, hogy az RLE (run length encode) módszer szerint tömörítse a képet. Az RLE-vel kódolt képek másolása gyorsabb.
A Bounce egyetlen TrueType betűkészletet használ, de három különböző méretben és három különböző stílusban. Az ***SDL_ttf*** programkönyvtárat használva írtam egy függvényt, ami betölti a TrueType betűkészletet, 0-tól 127-ig minden ASCII-karaktert megjelenít egy `SDL_Surface` szerkezetben, minden karaktert átalakít a képernyőhöz, és menti a betű magasságát és szélességét, hogy a szövegek kirajzolhatók legyenek a képernyőre.

A főhurok

Az SDL eseményvezérelt beviteli rendszert használ, hasonlóan az X-hez, a Mac OS-hez vagy a Windowshoz. Amikor lenyomunk egy billentyűt vagy megmozdítjuk az egeret, egy esemény kerül a sorba. A program vagy vár az eseményekre az `SDL_WaitEven()` segítségével, vagy lekérdezheti, hogy történt-e esemény az `SDL_PollEvent()` használatával. A főhurok feladata az események feldolgozása, a játék állapotának frissítése, a következő képkocka kirajzolása, majd az egészet elölről kell kezdenie, amíg nem végez.
Az, hogy az eseményekre várunk vagy lekérdezzük őket, a játék egész szerkezetét befolyásolja. Én a várakozást választottam – ekkor a műveleteket szívverésszerű időzítés vezérli. Azért szeretem ezt a megoldást, mert a program akkor dolgozza fel az eseményeket, amikor történnek, és közben a processzorhasználat is kézben tartható. Mindkét jellemző igen fontos a hálózati játékokban.
Az időzítőt a következő kódsorral hozzuk létre:

```
timer = SDL_AddTimer(10, timerCallback, NULL);
```

Ezután az SDL a `timerCallBack` függvényt tízezred-másodpercenként meghívja. Az időzítő visszahívó függvénye esetünkben az `SDL_PushEvent()` függvényt használva küld egy eseményt. Mivel az időzítő visszahívó függvénye külön szálban fut, akkor is tud eseményt küldeni, ha a játék eseményre várva áll. Az időzítő eseményét megkapva a Bounce ellenőrzi, hogy egy újabb képkockát kell-e kirajzolnia. Az időzítő biztosítja, hogy a program nem próbál meg másodpercenként száznál több képkockát kirajzolni, miközben lehetővé teszi, hogy a játék kisebb frissítési sebességgel is működhessen. A gépemen 85 képkocka/másodperc sebességgel fut, ami megfelel a monitorom képfrissítési gyakoriságának.
A Bounce több lapból áll. A főhurok kezeli azokat az eseményeket, amik minden lapnál közösek, például a kilépés az Esc billentyű vagy a szüneteltetés az F1 billentyű hatására. Miután a főhurok megvizsgálta az eseményt, továbbítja a pillanatnyi lapnak. Minden lap egy függvény, ami egy `SDL_Event` típusú értéket vár. Minden lap felelős az események kezeléséért, az idő nyilvántartásáért és a képernyő kirajzolásáért. Bár ez a megközelítés többször ismétlődő kódot eredményez, a programozónak nagyobb a szabadsága és objektumközpontú tervezést eredményez, amiben minden lap a laposztály egy példánya. Az 1. listában (49. CD Magazin/SDL könyvtár) látható példa a főhurok egyes részeit mutatja be, és jól megfigyelhető, hogy az eseményeket hogyan adjuk át az egyes lapoknak.
A `currentPage` teljes hatókörű változó a pillanatnyi lap megvalósítására mutat. Amikor egy lap új lapot akar indítani, létrehozza azt, és a mutatót arra a lapra állítja. A Bounce három lappal rendelkezik, az első lap az üdvözlőképernyő, ami a program indulásakor jelenik meg, a második lap magát a játékot kezeli, és a harmadik a „Nyertél/Vesztettél” üzenetet jeleníti meg.
Az üdvözlőképernyő eseménykezelője így fest (2. lista, 49. CD Magazin/SDL könyvtár).
Amikor ez a kód megkapja az időzítő eseményt, ellenőrzi, hogy mikor frissítette utoljára a képernyőt, és meghívja a `drawWelcome()` függvényt, ami animálja a képernyőt. Ha észleli az egérgomb lenyomását, az `initBounce()` meghívásával átvált a játék lapjára, és átállítja a `currentPage` változót, hogy az a játék lapjára mutasson. A következő alkalommal a főhurok `bounce()` függvénye fog meghívódni.

Animáció

Az objektumok mozgatása az úgynevezett piszkos képpontok módszerével történik, azaz minden képkockából mindig csak egy kis részt rajzolunk újra. Az objektum régi és új helyzetét egyaránt megjegyezzük. Amikor a Bounce a Földet rajzolja ki, először letörli a piszkos képpontokat, úgy, hogy háttérszínnek tölti fel azokat, ezután a Földet az új helyzetben rajzolja ki. Téglalapok kitöltésére és képek rajzolására ezeket használjuk:

```
SDL_FillRect(képernyő, téglalap, szín);
SDL_BlitSurface(kép, NULL,  képernyő ,
téglalap );
```

Az `SDL_FillRect()` egy `SDL_Surface` szerkezetben (például a képernyőn) egy téglalapot az adott színnel fest ki. A téglalapot az `SDL_Rect` szerkezet határozza meg, a színt az `SDL_MapRGB()` függvénnyel állítjuk elő.
Az `SDL_BlitSurface()` egy téglalapot az egyik felületről (surface) a másikra másol. Ha a forrástéglalap `NULL`, akkor az egész felület lemásolódik. Az `SDL_BlitSurface()` az a függvény, ami a színkulcsot alkalmazza, és kihasználja az RLE-kódolást.

Összegzés

Az SDL csökkenti a linuxos játékprogramok fejlesztésének az idejét. Elég kicsi ahhoz, hogy az elsajátítása ne vegyen el túl sok időt az életünkből, viszont elég sokat tud ahhoz, hogy akár kereskedelmi alkalmazásokat is fejlesszünk a segítségével. Remélem, hogy a cikk és a Bounce forráskódja elegendő lesz hozzá, hogy SDL-es játékok fejlesztésébe is bele merjünk vágni.

A kapcsolódó címek, valamint a listák megtalálhatóak a 49. CD Magazin/SDL könyvtárában.

Linux Journal 2003. június, 110. szám

Bob Pendleton (Bob@ Pendleton.com)
Az első programozási feladata az volt, hogy játékokat írjon át HP miniszámítógépekről az UNIVAC nagygépre. Azóta is a számítógépes játékok szerelmese. 1981 óta dolgozik a Unix különféle változataival és a Linuxszal. Bob független programfejlesztő és szakíró.