

## GRAFIKA C++ NYELVEN

Ezen az oldalon egy nagyon egyszerű C++ nyelvű grafikus könyvtár leírását olvashatod. A könyvtár célja, hogy a programozás tanuláshoz egy érdekes, ugyanakkor könnyen használható környezetet nyújtson.

Letöltés:

[kezdők csomagja Windows + Code::Blocks környezethez, verziófüggetlen](#)

[kezdők csomagja Windows + Code::Blocks 20.03 \(64bit\) környezethez](#)

[Font csomag](#)

[GitHub link](#)

(utolsó módosítás: 2020.02.07.)

### Használat

#### *A kezdőcsomaggal*

A kezdőcsomag Code::Blocks környezetben működik. Első nekifutásra mindenkinek ezt ajánljuk, és később, ha másik környezettel, vagy linuxon szeretnél fordítani, nézd meg a többi leírást.

Letöltés után csomagold ki egy könyvtárba, ahol a Code::Blocks project fájlra kattintva bejön a környezet, és azonn mód futtatható a példaprogram. A csomagban egy Hello World példaprogram, a szükséges fejléc, könyvtár és dll fájlok, és a project fájl van. A project fordítható debug és release módokban.

#### *Máshol, Windows/MinGW környezetben*

- Töltsd le az [SDL könyvtárat](#), és csomagold ki valahova.
- Töltsd le a csomaggal a graphics.hpp libgraphics.a fájlt valahova.
- Állítsd be a környezetedet úgy, hogy a linker könyvtárai között legyen mind az SDL lib könyvtára, mind a libgraphics.a könyvtára.
- Állítsd be, hogy az include könyvtárak között legyen a graphics.hpp könyvtára.
- Állítsd be, hogy a linker használja a graphics és az SDL librarykat (fontos ez a sorrend!).
- Ezek után le kell tudnod fordítani a programjaidat. A jó beállítások ilyen parancssori kapcsolóknak felelnek meg:  
g++ -L *SDL-lib-dir* -L *genv-dir* -I *genv-dir* -lgraphics -lsdl
- Ezek után le kell tudnod fordítani a programjaidat.

#### *Egyéb környezetben*

- Töltsd le az [SDL könyvtárat](#), és csomagold ki valahova, esetleg telepítsd csomagból (libsdl-dev, libsdl\_ttf2.0-dev).
- Töltsd le Linuxos csomagot és csomagold ki valahova.
- Állítsd be a környezetedet úgy, hogy a linker könyvtárai között legyen az SDL lib könyvtára.
- Állítsd be a környezetedet úgy, hogy az include könyvtárak között legyen az SDL include könyvtára és a graphics.hpp könyvtára.
- Állítsd be, hogy a linker használja az SDL libraryt.
- Add hozzá a projectedhez a graphics.cpp fájlt.
- Ezek után le kell tudnod fordítani a programjaidat.

### Debug és Release mód

A félev során felmerülhet, hogy a hibakeresést valaki esetleg nyomkövetéssel szeretné megoldani. Ehhez a könyvtárnak egy olyan példányára van szükség, amelyik tartalmazza az azonosítók neveit, ezért viszonylag nagyméretű, és emellett nem optimalizálással lett lefordítva. A kezdőcsomag is tartalmazza a libgraphicsd.a könyvtárat, ahol a "d" a debug-ot jelenti. Ezzel fordítva lehetséges lesz a nyomkövetés. Ugyanakkor a release módú könyvtár tartalmaz optimalizálást, ami gyorsabb működést jelent.

Ha még nem próbáltad a nyomkövetést, Code::Blocks alatt fordíts a bármít Debug módban, és az F7 gomb nyomásával soronként tudsz haladni a program végrehajtásában, akár menet közben a változók értékeinek nyomon követése mellett. Sok platformon sokféle nyomkövető felület van, még nem tudjuk, hogy a félev során lesz-e alkalmunk egyet komolyabban tárgyalni.

### Programozói felület

A könyvtár összes eleme a genv névtérben található. Ezt a leggyakrabban a using namespace genv; direktíva használatával lehet elérni.

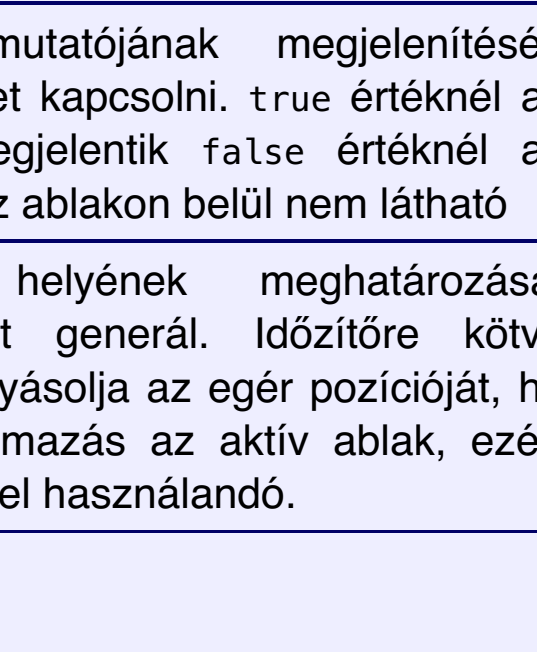
A könyvtárnak két fő eleme van, a rajzoló és az eseményfogadó rész. Meg tud jeleníteni egy ablakot a képernyőn, a rajzoló műveletek ebbe az ablakba rajzolnak, és az ebbe az ablakba érkező eseményeket lehet fogadni.

A rajzfelületen egy derékszögű koordinátarendszer segítségével tudunk pontokat megadni. A koordinátarendszer origója az ablak bal felső sarkában van, és lefelé illetve jobbra nőnek a koordináták értékei. A felületen kívül eső pontok használhatóak. A koordináták nulláról indulnak, és minden egész értékű koordinátpár a rajtfelület egy-egy pixelének felel meg.

#### *Rajzoló műveletek*

A rajzolás a groutput típuson keresztül történik, ennek egyetlen értéke van, aminek gout a neve, ezen keresztül lehet elérni az ablakot. A következő műveleteket lehet rá használni:

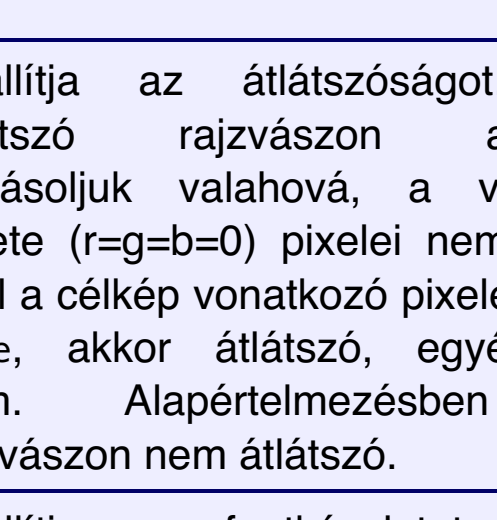
<code>gout.open(width, height [, fs]);</code>	Létrehozza az ablakot, width pixel széles és height pixel magas lesz a rajzolásra alkalmas felülete (mindkettő pozitív egész szám lehet). Ha a harmadik paraméter true, és a megadott felbontást a rendszer teljes képernyőn meg tudja jeleníteni, úgy a kijelző teljes képernyős módban indul. Az átváltás bizonyos rendszerekben akár több másodpercig is eltarthat.
<code>gout.save(filename);</code>	Lementíti az ablak tartalmát egy filename nevű Windows BMP típusú fájlként.
A rajzfelületen nyilvántartunk egy pontot, a rajzoló műveletek ebből a pontból indulnak ki, és a rajzolás végén ott hagyják, ahol a rajzolás véget ért. Ez a pont sosem hagyja el az open műveletnél megadott rajzfelületet.	
<div><div><code>gout.x()</code></div><div><code>gout.y()</code></div></div>	<div><div>A rajzpont aktuális x koordinátáját adja vissza (egész szám)</div><div>A rajzpont aktuális y koordinátáját adja vissza (egész szám)</div></div>
<code>gout &lt;&lt; move(x, y);</code>	Eltolja a rajzpontot vízszintesen x, függőlegesen y pixellel, rajzolás nélkül. Ha a végpont kívül esik a rajzolási felületen, nem mozgatja a pontot.
<code>gout &lt;&lt; move_to(x, y);</code>	Eltolja a rajzpontot az (x,y) koordinátájú pontba (amennyiben az a rajzolási felületre esik, különben nem csinál semmit).
<code>gout &lt;&lt; color(r, g, b);</code>	Megváltoztatja a további rajzolások színét. Az új színt vörös (r), zöld (g) és kék (b) komponensekből állítja elő, mindhárom komponens egy 0 és 255 közötti egész szám lehet.
<code>gout &lt;&lt; refresh;</code>	Frissíti a képet: megjeleníti a legutóbbi frissítés óta történt rajzolások eredményét. Frissítés nélkül a rajzoló műveletek hatása nem jelenik meg!
<code>gout &lt;&lt; dot;</code>	Az aktuális rajzpont színét megváltoztatja az aktuális rajzolási színre. A rajzpontot nem mozgatja.
<code>gout &lt;&lt; line(x, y);</code>	Egyenes szakaszt rajzol a rajzpontból kiindulva az aktuális színnel, a végpont vízszintesen x, függőlegesen y pixel távolságra lesz a kiindulási ponttól. A rajzolási pontot a végpontban hagyja.
<code>gout &lt;&lt; line_to(x, y);</code>	Egyenes szakaszt rajzol a rajzpontból kiindulva az aktuális színnel az (x,y) koordinátájú pontba. A rajzpontot a végpontban hagyja.
<code>gout &lt;&lt; box(x, y);</code>	A rajzpontból kiindulva x szélességű és y magasságú, vízszintes alapú téglalapot rajzol aktuális színnel kiszínezve. A rajzpontot a kiindulási pontból induló átló tulsó végén hagyja.
<code>gout &lt;&lt; box_to(x, y);</code>	Az aktuális színnel kiszínezett, vízszintes alapú téglalapot rajzol, amelynek az egyik átlója a rajzpontból indul és az (x,y) pontban ér véget. A rajzpontot az (x,y) pontban hagyja.

Szövegek megjelenítése következőképpen történik. Egy egy soros szövegnek van egy ún. <i>alapvonala</i> (angolul <i>baseline</i> ), ez egy láthatatlan vonal, amire „ráülnek” a karakterek. A karakterek képeinek nagy része az alapvonal fölött van, és valamennyire az alapvonal alá is nyúlhat. Az alapvonal fölötti rész maximális magasságát <i>ascenknek</i> hívjuk, az alapvonal alatti rész maximális magasságát <i>descenknek</i> hívjuk. Ha több sort akarunk egymás alá írni, az alapvonalak normál távolságát az ascent és a descent összege adja meg.	
<div><div><code>gout.ascent()</code></div><div><code>gout.cdescent()</code></div><div><code>gout.twidth(s)</code></div></div>	<div><div>Egy egész számot ad eredményként, az aktuális betűkészlet ascent értékét pixelekben.</div><div>Egy egész számot ad eredményként, az aktuális betűkészlet descent értékét pixelekben.</div><div>Visszaadja az s szöveg szélességét pixelekben, az aktuális betűkészlettel megjelenítve.</div></div>
<code>gout &lt;&lt; text(t);</code>	Kirajzolja a t szöveget a rajzpontból vízszintesen induló alapvonalra. A szövegben található '\n' karaktereket sorvege jelként értelmezi, és a szöveget a kiindulási pont vonalában, de a következő sorban folytatja. A rajzpontot a szöveg végén, az alapvonalon hagyja. A t típusa lehet string és char is.
<code>gout.showmouse(bool);</code>	Az egér mutatójának kiszínezését, elrejtését lehet kapcsolni, true értéknél az egérmutató megjelenítik false értéknel az egérmutató az ablakon belül nem látható
<code>gout.movemouse(x, y);</code>	Az egér helyének meghatározása. Egéreseményt generál. Időzítőre kövte akkor is befolyásolja az egér pozícióját, ha nem az alkalmazás az aktív ablak, ezért körültekintéssel használandó.

#### *Rajzvaszon műveletek*

Lehetőség van olyan felületre is rajzolni, ami nem a képernyő, hanem egy másik rajzvaszon típus, a canvas. Ennek tartalmát gyorsan lehet rámásolni a gout-ra, vagy másik canvas-ra.

Technikailag a gout egy a kijelzőhöz kötött canvas leszármazott.

<code>canvas(width, height);</code> <code>canvas c;</code> <code>c.open(width, height);</code>	Konstruktor, egy adott canvas méretét beállítja, width pixel szélesre és height pixel magasra.
<code>gout &lt;&lt; stamp(pattern, x, y);</code>	Másoló művelet, ami a gout-ra másolja a pattern-t, aminek canvas & pattern a típusa. Az x, y egész számokkal lehet szabályozni, hogy a képernyőn hová kerüljön a másolat.
<code>gout &lt;&lt; stamp(pattern, sx, sy, mx, my, x, y);</code>	Másoló művelet, ami a gout-ra másolja a pattern-t, aminek canvas & pattern a típusa. Az x, y egész számokkal lehet szabályozni, hogy a képernyőn hová kerüljön a másolat. Az előzőhöz képest itt megadhatjuk azt a tartományt, amit ki akarunk másolni, nem az egész pattern kerül a képernyőre. 
<code>canvas C;</code> <code>C.save(filename);</code> <code>C &lt;&lt; move(x,y);</code> <code>C &lt;&lt; move_to(x,y);</code> <code>C &lt;&lt; color(r, g, b);</code> <code>C &lt;&lt; dot;</code> <code>C &lt;&lt; line(x, y);</code> <code>C &lt;&lt; line_to(x, y);</code> <code>C &lt;&lt; box(x, y);</code> <code>C &lt;&lt; box_to(x, y);</code> <code>C &lt;&lt; text(t);</code>	A rajzvaszonzonra működik minden művelet, ami a gout-ra, kivéve a refresh, aminek nincs hatása. Minden rajzvaszonnak saját kurzora és aktuális színe van, tehát a rajzműveleteknek a hatása csak azon a rajzvaszonzonon jelentkezik, amikre meghívjuk.
<code>c.transparent(bool);</code>	Beállítja az átlátszóságot. Az átlátszó rajzvaszonzon amikor átmásoljuk valahová, a vázron fekete (r=g=b=0) pixelei nem írják felül a célképf vonatkozó pixeleit. Ha true, akkor átlátszó, egyébként nem. Alapértelmezésben a rajzvaszon nem átlátszó.
<code>c.load_font(filename, fontsize, antialias);</code>	Beállítja a fontkészletet. TTF fontokat támogat. A ttf fájl nevét és elérési újtját kell megadni az első paraméterben. A betűméretet a másodikban. A harmadik paraméter nem kötelező, kikapcsolható vele az antialias, ami akkor fontos, ha kétszer ugyanazt a szöveget egymásra más színnel akarod rajzolni, ilyenkor csak antialias nélkül lesz tökéletes a fedés. Érdemes a kiadott fontokat használni, úgy, hogy a ttf fájl a program könyvtárában legyen, így elég fájlnevet megadni, egyéb esetben elérési úttal együtt kell. Ne felejtse el Windows alatt dupla backslash karaktert használni az elérési útban. Beadandóknak csak a kiadott fontokat használni, abban a könyvtárban, ahol a project van, és ne töltsd fel a font fájlt.
<code>c.antialias(bool);</code>	Beállítja az élsimítást TTF font esetében, a font újratöltése nélkül. Ha true, akkor az élsimítás be van kapcsolva, egyébként nem. Alapértelmezésben az élsimítás aktív. Akkor van értelme kikapcsolni, ha pontos fedést szeretnél, például ugyanazt a szöveget más színnel újíráim ugyanott. Élsimítással ez a betűk szélein problémás lehet. Ha átlátszó canvasra írsz szöveget, élsimítással esetleg sötét aurát kaphat a betű világos háttérre stampelve.

#### *Eseményfogadó műveletek*

Az események kezelése úgy történik, hogy van egy eseményfogadó művelet, ami addig várakozik, amíg valamilyen esemény nem történik a programban. Az első esemény bekövetkeztekor rögtön visszaadja az eseményt. Ha több esemény is történik, azokat egyesével adja vissza (ilyenkor tulajdonképpen nem is várakozik, hanem rögtön visszaadja a következő eseményt).

Egy eseményt egy event típusú rekord ír le. A rekord mezoí a következők:

keycode	Billentyű lenyomása vagy felengedése esetén ez a mező tárolja a billentyű kódját. Pozitív érték jelzi a lenyomást, negatív a felengedést. A billentyű kódja a következők lehet: <ul style="list-style-type: none"><li>A karakterekét jelentő billentyű kódja megegyezik a karakter kódjával (ISO Latin 2 kódolással)</li><li>A vezérlőbillentyűk kódjait a következő azonosítókon keresztül lehet elérni: key_backspace, key_capsl, key_delete, key_down, key_end, key_enter, key_escape, key_f1, key_f2, key_f3, key_f4, key_f5, key_f6, key_f7, key_f8, key_f9, key_f10, key_f11, key_f12, key_f13, key_f14, key_f15, key_home, key_insert, key_lalt, key_lctrl, key_left, key_lshift, key_lwin, key_menu, key_numl, key_pgdn, key_pgup, key_ralt, key_rctrl, key_right, key_rshift, key_rwin, key_scrl, key_space, key_tab, key_up</li></ul>
button	Egérgomb lenyomása vagy felengedése esetén ez a mező tárolja a gomb kódját, lenyomás esetén pozitív, felengedés esetén negatív értékkel. A lehetséges kódok a következők: btn_left, btn_middle, btn_right, btn_wheelup, btn_wheeldown
pos_x, pos_y	Egérrel kapcsolatos eseményekhez ezek a mezők tárolják az egér aktuális koordinátáit az ablakban.
time	Időztítési esemény esetén ez a mező tárolja, hogy a program indulása óta hány ezredmásodperc telt el.
type	Az esemény típusát tárolja, lehet ev_key, ev_mouse és ev_timer

Az eseményekkel kapcsolatos műveletek a grinput típuson keresztül érhetőek el, ennek a típusnak egyetlen értéke van, a gin. A műveletek a következők:

gin	A gin változó önmagában használható feltételként, ami addig igaz, amíg az ablak nyitva van. Az ablak bezárása után hamis lesz, ami azt jelzi, hogy több esemény nem következhet be a programban.
gin >> ev;	Az eseményfogadó művelet: várakozik egy eseményre, majd beteszi azt az event típusú ev változóba. Eredményként a gin-t adja vissza. Az ablak bezárása esemény kivételével mindig érvényes eseményt tesz ev-be.
gin.timer(wait);	Beállítja az időzítőt úgy, hogy minden wait ezredmásodperc elteltével küldjön egy időztítési eseményt. Ha a wait értéke nulla, kikapcsolja az időzítőt.

Végül egy összefoglaló táblázat arról, hogy milyen események következhetnek be a program futása során, és hogyan állítják be az eseményrekord mezoit:

Billentyű lenyomása	keycode: a billentyű kódja (pozitív szám) type==ev_key A többi mező értéke 0
Billentyű felengedése	keycode: a billentyű kódjának negáltja (negatív szám) type==ev_key A többi mező értéke 0
Egérgomb lenyomása	button: a gomb kódja (pozitív szám) pos_x, pos_y: a gomb megnyomásának helye type==ev_mouse A többi mező értéke 0
Egérgomb felengedése	button: a gomb kódjának negáltja (negatív szám) pos_x, pos_y: a gomb felengedésének helye type==ev_mouse A többi mező értéke 0
Egér mozgatása	button = 0 pos_x, pos_y: az egérkurzor új helye type==ev_mouse A többi mező értéke 0
Időztítő esemény	time: a program indítása óta eltelt ezredmásodpercek száma type==ev_timer A többi mező értéke 0

### Változások naplója

- 2020.02.07. 12:55 - SDL2 port, 64bit. Ismert jelenség a billentyűkezelés leegyszerűsödése.
- 2012.03.04. 18:01 - canvas constructor, copy constructor bugfix
- 2012.02.27. 11:05 - move(1,0) bug javitva
- 2011.02.13. 22:20 - font betöltés, kikapcsolható antialias támogatással

- 2009.03.18. 15:10 - memóriafolyás probléma kiküszöbölése, relatív mozgás **átalakítása** úgy, hogy a line(0,0), box(0,0), move(0,0) nem csinál semmit, és más esetekben a paraméter abszolútértékének megfelelő számú pixelt rajzol. Egér kurzorával kapcsolatos műveletek bevezetése.
- 2009.02.12. 21:10 - event type bevezetése, debug módban fordított lib kiadása
- 2009.02.10. 11:50 - Linux kompatibilitásis problémák kezelése
- 2008.02.12. 18:10 - rajzvaszon műveletek hozzáadása, gout fullscreen lehetőség, szövegek betűinek átmenetességének megszüntetése.
- 2007.03.21. 22:10 - színes szövegek megjelenítésének javítása
- 2007.02.15. 10:50 - első letölthető változat