Fehér Krisztián

Grafikus és játékalkalmazások programozása

Fehér Krisztián

Grafikus és játékalkalmazások programozása

BBS-INFO Kiadó, 2017.

A könyv nagyobb mennyiségben megrendelhető a kiadónál:
BBS-INFO Kiadó, 1630 Bp. Pf. 21. Tel.: 407-17-07

Papírkönyv ISBN 978-615-5477-51-5
E-book ISBN 978-615-5477-52-2

Kiadja a BBS-INFO Kft.
1630 Budapest, Pf. 21.
Felelős kiadó: a BBS-INFO Kft. ügyvezetője
Nyomdai munkák: Biró Family Nyomda
Felelős vezető: Biró Krisztián

TARTALOMJEGYZÉK

1. BEVEZETÉS

Könyvünk számítógépes grafikát megjelenítő alkalmazások elkészítését mutatja be, elsősorban különböző játékprogramok készítésére fókuszálva PC-s, androidos és webes plattformokra.

Alapvető célunk, hogy az egyes témákat programozási szempontból a lehető legáltalánosabban tárgyaljuk, így a megszerzett ismeretek könnyedén átvihetővé válnak bármilyen, grafikus megjelenítést támogató programozási nyelvre.

A kor kihívásainak megfelelően már nem csupán egyetlen platformra fókuszálunk, hanem asztali- mobil- és webes alkalmazásokra is mutatunk példákat.

Mit tanulhatunk meg a könyvből?

A könyv segítségével az olvasó elsajátíthatja a grafikus megjelenítést használó játékok készítésének alapjait és képessé válik önállóan játékprogramokat megtervezni és kivitelezni.

Négy játéktípust mutatunk be: táblás-, akció-, ügyességi-platform-, és 3D akciójátékot fogunk készíteni Visual C/C++, ActionScript és JavaScript nyelven megírva, de a leírtakat felhasználva bármi más is elkészíthető, ez csupán az olvasó fantáziáján múlik.

Miben más ez a könyv?

Ez a könyv nem meglévő, magasszintű grafikus keretrendszerekhez ad leírást, hanem alacsony szinten elkészíthető grafikus játékalkalmazások programozását mutatja be.

Nem kell heteket, hónapokat azzal töltenünk, hogy egy meglévő keretrendszer alapvető kezelését megtanuljuk. Mi nem ezt az utat követjük.

Könyvünk azonnal kipróbálható válaszokat ad, ennek megfelelően erősen gyakorlati szemléletű. Elméletek és matematikai tételek, képletek hosszas taglalása helyett a használhatóságot tartjuk szem előtt. Így azon

olvasóink is haszonnal forgathatják, akiknek nem feltett szándékuk matematikából, vagy számítástechnikából ledoktorálni. Minden algoritmus azonnal felhasználható saját programokban.

Noha manapság már egy lépést sem lehet megtenni az objektumorientáltság mantrája nélkül, mi alapvetően ún. procedurális megközelítéssel készítjük programjainkat: objektumok készítése helyett, ahol csak lehet, egyszerű változókkal és függvényekkel dolgozunk. Ez a szemlélet vihető át legkönyebben más programozási nyelvekre is, de máskülönben az osztályok világában is hasznát vehetjük.

Célunk tehát az egyszerűség és az érthetőség, nem pedig a programozási virtuozitás erőltetése.

Példaprogramjainkat a lehető legegyszerűbb szinten igyekszünk tartani. Példaprogramjaink hossza a legritkább esetben haladja meg az 1000 sort, jellemzően ennek csak a töredékéről beszélhetünk.

Teljes forráskódokat közlünk, nem csupán kódrészleteket.

Miről szól könyvünk?

Könyvünk az alábbi területeket tárgyalja és mutatja be:

- fejlesztőrendszerek gyors üzembehelyezése
- programozási gyorstalpalók: JavaScript, Visual C/C++, ActionScript/Flex
- GDI, GDI+, HTML Canvas, Flash, OpenGL rajzrendszerek programozása
- platformspecifikus programozási módszerek bemutatása
- keresztplatformos szemlélet bemutatása, kialakítása
- játékprogramozási alafogalmak és alaptechnikák
- 2D vektoros megjelenítés alapjai
- 3D vektoros megjelenítés alapjai
- 3D mélységi rendezés
- raszteres képkezelés alapjai
- 2D táblás és ügyességi-akciójátékok készítése
- 3D alapú játékok készítése
- VR alkalmazások alapjai, példaprogrammal.

Milyen programokat tárgyalunk?

A legfontosabb, hogy könyvünk négy konkrét játékprogram kódját tartalmazza, három programozási nyelven, négy változatban: Visual C/C++ (Windows desktop), JavaScript (Web), ActionScript/Flex (Windows desktop, Android). Ezek a játékok teljesen működő alkalmazások,

azonban nincsenek a végletekig lecsiszolva, ezáltal lehetőséget kívánunk adni arra, hogy az olvasó testre szabhassa őket, így a könyv olvasása és kipróbálása során mindenki teljesen egyedi programokat készíthet.

A játékprogramok mellett sok ún. tanulmányprogramot is bemutatunk. Ezek inkább technikai prototípusok és arra ösztönzik az olvasókat, hogy játékprogrammá fejlesszék őket, vagy felhasználják a bemutatott technikákat saját programjaikban.

Különböző segédprogramokat és nem mellesleg azok elkészítését is bemutatjuk az olvasónak. Ezekkel a játékkészítés egyes fázisait tehetjük egyszerűbbé, gyorsabbá és nem mellesleg rengeteget tanulhatunk általuk.

Összesen 5 különböző rajzrendszer alapvető programozását lehet elsajátítani a könyv segítségével.

Miről nem szól a könyv?

Könyvünk a játékprogramok készítéséhez szükséges legfontosabb módszereket tárgyalja és többféle játéktípust is bemutat. Ez a terület azonban rendkívűl szerteágazó, természetes, hogy egy könyvben nem tudunk mindent megmutatni. Egyebek mellett, ez a kötet nem tartalmazza:

- a különféle útvonalkeresési technikákat és ezzel összefüggő megoldásokat
- izometrikus megjelenítésű játékokat
- az ún. csempézési technikákat
- nagyméretű keretrendszerek (Unity, Ogre stb.) programozását
- 3D textúrák programozását
- 3D nem látható poligonok keresését és eldobását
- 3D fénykezelési és árnyékolási technikákat
- 3D részletes figuraanimációt.

A fentiek jelentős része azonban a könyv felhasználásával, egy kis kreativítással és utánajárással szintén megoldhatók. Könyvünk bőségesen tartalmaz hivatkozásokat, melyek segítségével utat kívánunk mutatni a további fejlődéshez is.

Kinek szól a könyv?

A könyv a grafikus és játékrogramozást megismerni vágyó programozóknak szól. Alapvető C, ActionScript és JavaScript / HTML programozási ismeretek szükségesek a könyv ismeretanyagának teljes elsajátításához, de ebben külön fejezet is segítségére van az olvasónak.

Továbbá minden fejezet tartalmaz részletes magyarázatokat kapaszkodó gyanánt, így kezdők is képesek lesznek megérteni az egyes fejezeteket, elképzelhető azonban, hogy számukra ez egy hosszabb folyamat lesz.

A könyv felépítése

Az egyes fejezetek egymásra épülnek, ezért célszerű egymás után haladni velük, mindazonáltal, ha az olvasó otthon van egy-egy témában, nyugodtan át is lehet ugrani egy-egy szakaszt.

A könyv kb. első negyedében nem a programozásé a főszerep, sokkal inkább az előkészületeké. Mindent nagyon egyszerű szinten, érthetően tárgyalunk.

A könyv áttekintéssel kezdődik a számítógépes grafika fejlődéséről, a teljesség igénye nélkül, mégis bevezetve az olvasót a témába. A cél az, hogy egy alapszintű kép alakuljon ki arról, miről szól ez a terület. Ezek az ismeretek segítenek az egyes technológiák, területek közti eligazodásban is.

A második fejezet a fejlesztőeszközök bemutatásának van szentelve. Könyvünkben C, ActionScript és JavaScript programozási nyelveket fogunk használni. Itt megjegyezzük, hogy itt Visual C++ nyelv lesz használatban, viszont alapvetően C szintaktikájú kódokat fogunk írni, ahol a Visual C++ a hátteret adja csupán. Ezért minden esetben, amikor C nyelvről írunk, a gyakorlatban valójában Visual C++-t fogunk használni!

A szerző saját tapasztalata alapján hangsúlyozza, mekkora gátat jelenthet, ha az ember nem tudja, melyik a megfelelő fejlesztőeszköz egy adott feladathoz, ezért a kommersz fejlesztőeszközök mellett megadunk ingyenes alternatívákat is. Részeletesen bemutatjuk a fejlesztőeszközök telepítését és alapvető konfigurálását, hogy az olvasó biztos háttérrel kezdhesse el kipróbálni a könyv példáit.

Ezt követően megnézzük mindazon programozási alapismereteket mindhárom tárgyalt nyelven, melyekre szükségünk lesz a későbbiekben, majd megismerjük, hogyan néznek ki a különböző programozási nyelveken a legegyszerűbb alkalmazások programkódjai. Ezt ugródeszkaként használva rátérünk a grafika programozásának alapjaira.

A játékprogramozás eszköztárában egy adott programnyelv speciálisabb eszközei is beletartoznak, ezért ezeknek külön fejezetet szenteltünk.

Ezután elkezdjük tárgyalni kifejezetten a kétdimenziós grafikával kapcsolatos technikákat.

A kilencedik fejezetben már konkrét játékprogramokat fogunk készíteni.

Ezután elkezdjük tárgyalni a térbeli megjelenítést, bemutatjuk, milyen programokkal és hogyan készíthetünk otthon komplex háromdimenziós objektumokat, majd egy játékprogramon keresztül ezt a gyakorlatban is demonstráljuk.

Érdekességképpen és egy példaprogram erejéig röviden kitérünk a VR alkalmazásokra is.

A könyv vége még egy lendületet ad a grafikus programozáshoz: leírást adunk a GDI+ és az OpenGL (rögzített csővezetékes technika) alapvető programozásához.

A könyv végül a DirectX és a CUDA programozás rövid bemutatásával zárul.

A függelék számos további hasznos technikai információt tartalmaz, melyeket ugyan nem illesztettünk be egy fejezetbe sem, mégis hasznos referenciaként szolgálhatnak a hétköznapokban és sok keresgélést spórolhatunk meg velük. Ugyanitt egy szakkönyv listát is találunk, melyek segítségével a könyvben tárgyalt, vagy érintett témákban tovább mélyíthetjük tudásunkat.

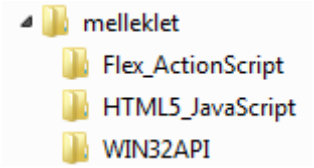
Hogyan használjuk a könyvet?

A legfontosabb a kísérletező kedv. Másodsorban pedig a könyv minden gyakorlati részét ajánott azonnal kipróbálni számítógéppel. Ha elakadunk, puskázhatunk a letölthető mellékletekből, ahol minden program teljes forráskódja megtalálható és nem utolsó sorban sok gépeléstől is megkímélhetjük magunkat. A forráskódokban igyekeztünk lehetőleg magyar elnevezéseket használni, de néhol angol elnevezésekre is fogunk példát találni.

Mellékletek

Könyvünk példaprogramjai letölthetőek a kiadó weboldaláról. (http://www.bbs.hu/letoltes/gjp.zip) A példaprogramok platformokra, ill. programozási nyelvekre lebontva vannak csoportosítva. Igyekeztünk minél több teljes példaprogramot összegyűjteni, hogy az olvasók minél gyorsabban kipróbálhassák azokat.

A melláklet fájljait három könyvtárba rendeztük, a könyvben tárgyalt három programozási nyelvnek megfelelően.



Mindhárom könyvtár tartalmazza a könyvben hivatkozott összes példaprogram teljes forráskódját és az esetleges kiegészítő fájlokat is.

A szerzőről

A szerző veterán szoftvertesztelő, minőségbiztosítási tanácsadó, a Magyar Térinformatikai Társaság (HUNAGI) egyéni szakértői tagja és diplomás német irodalmár.

Gyerekkorában autodidakta módon tanult meg programozni, az évek során számos programozási nyelvvel megismerkedett. Megszerzett tudását előszeretettel használja alternatív, kísérleti alkalmazások készítésére, melyek egy része ingyenesen elérhető, sőt vannak köztük nyílt forráskódúak is.

Sok időt fordít saját térinformatikai keretrendszerének fejlesztésére, a ZEUSZ-ra, melyet a NASA-nál is ismernek. A ZEUSZ saját 2D és 3D téképmegjelenítő grafikus motorral rendelkezik, mely saját, egyedi fejlesztés és fut asztali számítógépen, Android platformon és webböngészőben egyaránt (lásd a Függelék képernyőképeit). A könyv nagyban támaszkodik ezekre a technológiákra is, a szerző tehát valóban azt a tudást osztja meg az olvasókkal, amit saját maga is használ a hétköznapokban.

A szerző az Adobe Flash platform és az AIR keretrendszer, az Android operációs rendszer lelkes híve.

Több könyve is megjelent már a hazai könyvesboltokban az elmúlt években, nem egy közülük sikerlisták élére is került. Munkáiról bővebben a **http://feherkrisztian.atw.hu/** weboldalon is lehet olvasni.

2. SZÁMÍTÓGÉPES GRAFIKA EGYKOR ÉS MA

Könyvünk kezdetén megismerkedünk a számítógépes grafika főbb irányvonalaival, felhasználási területeivel és napjaink trendjeivel. Nem a teljességre törekszünk, célunk inkább az, hogy felkeltsük az érdeklődést a játékfejlesztés iránt és pontosítsuk az olvasó ismereteit e területen.

2.1. Történelmi visszatekintés

A számítástechnikai eszközökkel előállított grafikus technológiák hosszú fejlődésen mentek keresztül a **mikroszámítógépek** hőskora óta és ez a folyamat jelenleg is tart.

A számítógépes grafika eleinte csupán egy érdekes plusz volt, ami egyrészt annak tudható be, hogy a korai személyi számítógépek igen szerény grafikai megjelenítési képességekkel voltak felruházva, másrészt pedig annak, hogy ez a piac akkoriban még nem létezett.

A számítógépes játékipar hozta meg az igazi fellendülést, még ha erre sokáig várni is kellett. Ennek ellenére megjelentek korai grafikai megjelenítéssel készült játékok, mint amilyen a legendás Pong volt, amelyben egy ping-ponghoz hasonló játékot lehetett játszani.

A nehéz kezdet egyik oka a hardveres háttér lassú fejlődése volt, mivel a vizuális megjelenítésnél fontosabb volt az adatfeldolgozási- és adattárolási képesség.

A Commodore cég legendás 8 bites Commodore 64 számítógépe nem az egyetlen, de jelentős mérdföldkő volt ebből a szempontból. Ezek a gépek már dedikált grafikus móddal rendelkeztek és 16 szín megjelenítésére is képesek voltak, mai szemmel nézve szerény felbontás (320x200 ill. 160x200 pixel 16 színnel) mellett. Processzorának órajele 1MHz volt, ami sokféle feladat elvégzésére képessé tette. A legendás C64 egyik óriási vívmánya volt, hogy nemcsak elérhetővé tette bárki számára a programozást, hanem azt szerethető, emberi léptékkel kínálta. Ennek ellenére

a grafikát használó, igényes programok elkészítése nehézkes és lassú feladat volt ezekkel a gépekkel.

Megjegyezzük, hogy a dinamikusan fejlődő **személyi számítógépeken** a játékok mellett, a grafikus felületű felhasználói programok, később pedig komplett operációs rendszerek is megjelentek.

A Nintendo és SEGA cégek ún. **konzoljai** egy új irányt szabtak, mivel ezek a gépek kifejezetten szórakoztatás céljából készültek: játékok futtatására tervezték őket, TV képernyő használata mellett. Ezek a gépek speciális hardverrel rendelkeztek, melyek programozását külön-külön meg kellett tanulni és (nagy különbség!) az átlagember számára nem voltak elérhetőek programozási eszközök ezekhez a gépekhez. Ezek a gépek már 16 bites processzorral és fejlettebb, gyorsabb grafikai megjelenítéssel rendelkeztek, melyek sok tekintetben túlszárnyalták a kortárs PC-k teljesítményét is. A sorba jóval később állt be a Sony a Playstation és a Microsoft az XBOX termékvonalával, melyek mára azonban de facto a "konzolvilágot" jelentik. A konzolok egyik további hátránya, hogy szinte kizárólag csak játékra, esetleg filmnézésre lehet használni őket, általános feladatokra nem.

Az IBM PC típusú személyi számítógépeken a grafikus alkalmazások készítése a Windows termékek elterjedéséig kifejezetten hardverközeli, alapvetően a CPU bevonásával végzett feladatnak számított. Ennek előnye a viszonylag jó kompatibilitás volt, hátránya viszont a visszafogottabb fejlődés. Jó darabig ebben a konzolok sokkal jobbak voltak, a már ismertetett hátrányaik mellett. Ezen alapvetően az sem változtatott, hogy ekkorra már kifejezetten grafikus kártyákat is gyártottak.

A Windows 95 és az Intel Pentium (és az első kommersz AMD) processzorok idejére a játékipar már határozottan felemelkedőben volt. Ekkoriban már látszott, hogy a háromdimenziós megjelenítésre építő játékok igényei messze meghaladják a rendelkezésre álló hardverek teljesítményét.

Megjelentek az első „3d gyorsítókártyák”. Ezek olyan grafikus bővítőkártyák voltak, melyek speciális hardveres részegységekkel rendelkeztek, amelyeket a háromdimenziós megjelenítéshez szükséges bizonyos számítások és megjelenítések elvégzésére alkottak meg. Az első ilyen kártyák a 3dfx kártyák voltak és erősen hardverfüggő technológiát kínáltak.

A filmipar a számítógépes grafikát hosszú évek múltán kezdte alkalmazni. Ebben a folyamatban az Apple-höz köthető Pixar cég úttörő

munkát végzett. A cég neve csakhamar a számítógéppel előállított animációs filmekkel forrt egybe.

Szintén ekkoriban szabadult el igazán a számítógéptulajdonosok körében az a fajta "fegyverkezési verseny", melyben mindenki a legújabb, leggyorsabb hardverek birtoklásával vívott ki magának tiszteletet baráti társaságokban. (Ez ma is megfigyelhető érdekes viselkedés egyébként.)

A Microsoft DirectX technológiája a Windows operációs rendszeren végül sikeresen kanalizálta a hardverfüggőségeket és a hardver közvetlen programozása helyett a programozók figyelme egyre inkább az absztraktabb szoftveres, de széleskörű kompatibilitással kecsegtető DirectX felé fordult. A DirectX egyébként már nem csupán a grafikus megjelenítésért felelő alrendszer, hanem az audiorendszer és úgy általában a számítógép multimédiás képességeinek a programozását is lehetővé teszi. Tehát egy komplex dologgá vált.

Megjegyezzük, hogy a grafikus megjelenítőipar résztvevői OpenGL néven egy nyílt szemléletű, a DirectX-hez hasonló programozási felületet is megalkottak, mely mind a mai napig él és virul, bár elsősorban a professzionális (nem játék-) alkalmazásokban használják.

A CD, DVD, majd Blu-ray lemezek elterjedése és a szárnyra kapó processzorfejlődés a konzolokat sem hagyta érintetlenül, bár a jövő előreláthatóan az online tartalomvásárlás lesz ezen a területen is.

2.2. A jelen

2.2.1. A párhuzamosítás diadala

A számítógépes grafika világában manapság minden a párhuzamosításról szól. A grafikus kártyák saját feldolgozó egységgel, ún. **GPU-val (Graphical Processing Unit = Grafikus Feldolgozó Egység)** rendelkeznek. Ez tulajdonságaiban hasonlít egy hagymányos CPU-hoz: ennek is vannak feldolgozó magjai, órajele stb. A különbség viszont, hogy ezek az egységek nem programozhatóak általánosan, hanem csak bizonyos feladatok elvégzésére, mint például nagy mennyiségű adaton végzett matematikai műveletek végrehajtására, háromszögek megjelenítésére, takarási számítások elvégézésére stb. További különbség, hogy sokkal több processzormaggal rendelkeznek, mint egy hagyományos CPU. Egy 8-16 magos CPU már igen jónak számít, egy komolyabb GPU viszont akár több ezerrel is rendelkezhet.

A trükk a GPU-k esetében az, hogy ezek magjai egyszerűbb elemi műveleteket képesek nagy mennyiségben és párhuzamosan végrehajtani. Ilyenek például a tömbműveletek.

A GPU-kban rejlő erő kiaknázásához a vonatkozó algoritmusokat a programozás során ennek szellemében, azaz "párhuzamosított módon" kell megírni.

A párhuzamosításban az NVIDIA cég bővítőkártyái viszik a prímet, a CUDA technológiával. A kommerszebb (általános alkalmazások és játékok) felhasználásra a GeForce márkanevű videokártyákat gyártják, míg professzionális feladatokra a Quadro és a Tesla kártyákat. A professzionális termékvonal jellemzője az erősebb hardveres integráltság (= nagyobb teljesítmény) és a jóval alacsonyabb fogyasztás, no és persze a magasabb ár is. Egy belépő szintű Quadro kártya ára hazánkban 40-60.000 forint körül mozog, a komolyabbaké pedig egészen több millió forintig terjedhet.

Quadro kártyákat használnak például a filmipari szoftvereknél, ahol gigantikus mennyiségű forgatási anyagot kell feldolgozni a lehető legrövidebb idő alatt.

Elterjedt tévhit, hogy a Quadro kártyák nem alkalmasak játékprogramok futtatására. Megnyugtatásul közöljük: nagyon is alkalmasak.

A Tesla kártyák lényegében bővítőkártyába gyömöszölt szuperszámítógépek, melyeket kifejezetten tudományos célú számításokra ajánlanak.

Az NVIDIA legújabb GPU architektúrája a VOLTA, amit kifejezetten mesterséges intelligenciát igénylő alkalmazásokra szánnak.

Megjegyezzük, hogy az NVIDIA jelenleg legnagyobb teljesítményű (és legdrágább) mini-szuperszámítógépe a DGX kódnévre hallgat és lényegében egy közepes méretű dobozka, amit telepakolnak Tesla kártyákkal és pár Intel Xeon CPU-val. Egy ilyen gép akár 40.000 CUDA magot is képes kezelni, ami páratlan teljesítményhez enged hozzáférést.

2.2.2. Hibrid teljesítménynövelés

Mivel a kvantumprocesszorok egyelőre még csak kísérleti stádiumban léteznek, egyre többen igyekeznek kihasználni a GPU-kban rejlő potenciálokat általánosabb célú feladatokra is, mivel nagyságrendekkel, a legrosszabb esetekben is minimum 2-10 szeresére képesek növelni egy program számítási teljesítményét.

A trend egyértelmű, a rendkívül magas költségek miatt azonban ezek a megoldások egyelőre erősen feladatfüggőek és persze borzasztóan drágák.

Ennek egyik vállfaja a felhőalapú számítási kapacitások (angol szakkifejezéssel: "cloud computing") elérhetősége, akár bérleti alapon is, amennyiben csak korlátozott ideig van szükség kiugróan nagy kapacitás elérésére.

Ennek egyik érdekes formája az ún. streaming alapú játék, ahol egy konzolon keresztül egy felhőben levő grafikus feldolgozó teljesítményéhez lehet hozzáférni, sőt a komplett játékprogram is a távolban fut. Az ilyen konzolok így csak (sután és pontatlanul megfogalmazva) a képernyőképeket töltik le. Ilyen konzol például az NVIDIA Shield. A dolog óriási előnye, hogy nem kell beruházni méregdrága számítógépbe és videokártyába.

2.2.3. Filmipar

A filmipar számítógépek nélkül a mai formájában képtelen lenne működni. Már csak azért sem, mert a forgatott nyersanyagok szinte kivétel nélkül csak digitálisan jönnek létre, feldolgozásuk pedig emiatt számítógépekhez kötött.

A nyersanyagokon végzett utómunkálatok óriási számítási kapacitást igényelnek. Ezekkel az igényekkel csak a nagyteljesítményű grafikus gyorsítók képesek lépést tartani.

A számítógépes animációs filmek közkedveltek. Ezek elkészítéséhez gyakorta ún. számítógépfarmokat alkalmaznak, mert a grafikus gyorsítókkal felszerelt gépek egész láncolata képes csak olyan teljesítményt biztosítani, ami a képkockák elkészítéséhez szükséges.

A televíziós műsorok szintén igénylik a kimagasló számítási kapacitást, mivel ott a valósidejű adatfeldolgozási- és megjelenítési képesség is alapkövetelmény.

2.2.4. A videojátékok és a filmipar

Ide kívánkozik, hogy napjainkra a videojátékokhoz készült filmes betétek, sőt egyes játékok érdemi cselekményének ábrázolása már de facto elérte a játékfilmek minőségét.

Egy-egy bemutató (trailer), átvezető jelenet megalkotásán már ugyanúgy színészeket, rendezőket, operatőröket stb. használnak, mint a mozifilmeknél. Elmondható, hogy a videojátékoknak is kezd kialakulni a maga kis Hollywoodja.

2.2.5. Otthoni felhasználás és játékipar

A játékipar a felhasználói szemmel leginkább követhető területe a számítógépes grafikai fejlődésnek. A legújabb játékok szemkápráztató módon tálalnak elképzelt világokat akár térbeli megjelenítésssel, vagy virtuális valóság beiktatásával (lásd lentebb) is.

Természetesen egy e-mail megírása, vagy a születésnapi fotók rendszerezése is grafikus felületű operációs rendszereken és alkalmazásokkal történik, noha ezt már annyira megszoktuk, hogy nem is tudatosul bennünk.

2.2.6. Dizájn és reklámipar

A számítógépes grafika egyik triviális felhasználási módja művészeti célú. Ki ne ismerné például az Adobe cég Photoshop, vagy Illustrator termékeit?

A reklámipar, nyomdaipar, **számítógéppel támogatott tervezés (Computer Aided Design - CAD)** napjainkban elképzelhetetlen a legmagasabb színvonalú számítógépes grafikai megoldások nélkül.

2.2.7. Tudomány, kutatás

A fentieken kívül számos olyan professzionális felhasználási terület létezik, melyek nagyteljesítményű grafikai megjelenítés nélkül elképzelhetetlenek lennének.

A teljesség igénye nélkül ilyen területek az orvosi képalkotás, energiakutatás, térinformatika, közbiztonsági / védelmi rendszerek, virtuális valóság, mesterséges intelligencia, kiterjesztett valóság, önjáró autók alkalmazása, meteorológiai szimulációk, űrkutatási elemzések elvégzése stb. A sort hosszan folytathatnánk.

2.3. Nagyteljesítményű grafikus API-k

A különböző **API-k (Application Programming Interface = Alkalmazásprogramozási felületek)** megszabadították a fejlesztőket attól, hogy hardverközeli, akár kártyánként is különböző, testreszabott programokat írjanak.

A grafikus API-k ezt a terhet levették ugyan a programozókról, ám cserébe a videokártyák így korlátozottan voltak programozhatóak, jellemző volt az ún. **rögzített csővezetékes (Fixed Function Pipeline, FPP)** programozási gyakorlat. Ez azt jelenti, hogy a videokártya szolgál-

tatásait programozási szempontból meghatározott, "előre rögzített" módon és sorrendiségben lehett igénybe venni. Például egy 3d-s objektum megjelenítéséhez rögzítve voltak azok a műveletsorok, melyeken keresztül az elvégezhető volt. Bár ez a módszer mind a main napig elérhető, használatuk nem javasolt. Helyette az ún. **shaderprogramozás** hódít, ami a grafikus kártyák még nagyobb szabadságfokú programozását teszi lehetővé, ám ezért ismét az egyszerűséget kell feláldozni: ez a programozás nehezebben tanulható. Megemlíthető azonban, hogy a rögzített csővezetékes gyakorlatot sem kell lenézni, sőt, ugyanúgy használható napjainkban is, mint régen. Ízlések és pofonok...

A különféle grafikus programozási felületek (API-k) manapság nagyon jó minőségűek és leginkább támogatottságukban térnek el egymástól. A támogatottság mögött többnyire kemény üzleti érdekek húzódnak, semmint használhatósági megfontolások.

Megjegyezzük, hogy bár a legtöbb API keresztplatformos megoldásként hirdeti magát, ezt a gyakorlatban ismereteink szerint csak az Adobe Flash platformra épülő AIR keretrendszernek sikerült megugrania abban a formában, ahogyan azt a keresztplatform kifejezés alatt érti az ember, azaz: "egyszer megírni, több platformra szállítani".

Másrészt azonban el kell fogadnunk, hogy minden platformnak megvannak a maga sajátosságai (előnyeikkel és hátrányaikkal együtt), így bizonyos fokú testreszabás sok esetben teljesen elfogadható. Erre a könyvünkben is látni fogunk példát.

Fontosnak tartjuk megemlíteni, hogy a könyvünkben tárgyalt Flash, Web/JavaScript és Windows platformok programozása nagyon jól támogatott dokumentáció tekintetében. Nemcsak sok háttéranyag, hanem rengeteg példakód is található a kapcsolódó oldalakon. Ez nem utolsó szempont a tanulásnál, ezért a következő fejezetben külön meg is említjük ezeknek az oldalaknak az elérhetőségét.

Most pedig tekintsük át madártávlatból napjaink legelterjedtebb grafikus API-jait!

2.3.1. OpenGL

Az OpenGL a Khronos Group által fejlesztett, kiforrott API, mely keresztplatformos működést ígér. Ez a gyakorlatban, ebben a formában nem teljesen így van, mert bár ugyan elérhető Windows, Linux és MAC OS X környezetekhez is, az OS X már nem is támogatja a legfrissebb megoldásait (hanem tipikus Apple megoldásként az Apple saját, Metal

felületét kezdte el erőltetni). Mobileszközökre pedig egy külön implementációja létezik.

Az OpenGL-t korábban játékfejlesztésre is használták (például Quake4), ám napjainkban elsősorban a professzionális tervező- és grafikus szerkesztőalkalmazásokban alkalmazzák, melyek igénylik a nagyteljesítményű térbeli megjelenítést.

Az OpenGL weboldala: https://www.opengl.org/

2.3.2. Vulkan

A Vulkant elsősorban játékfejlesztésre találták ki és meglehetősen új kezdeményezés. Szintén a Khronos Grouphoz köthető.

A Vulkan weboldala: https://www.khronos.org/vulkan/

2.3.3. OpenGL ES

Az OpenGL ES beágyazott rendszerek grafikus gyorsítóinak programozását hivatott támogatni. Elsősorban itt az Androidot futtató eszközökre kell gondolnunk. Ez a "sima" OpenGL-től eltérő API, noha nevükben hasonlóak.

Az OpenGL ES weboldala: https://www.khronos.org/opengles/

2.3.4. WebGL

A WebGL a HTML5 Canvas komponensét megjelenítésre felhasználó, grafikus gyorsítást kínáló programozási felület, mely JavaScript nyelven programozható. Elsősorban a Chrome, Firefox, Safari és Opera böngészők támogatják, tehát csak ott használható, ahol a HTML5 is támogatott.

A WebGL weboldala: https://www.khronos.org/webgl/

2.3.5. Stage 3D

A Stage3D az Adobe Flash platformjára épülő háromdimenziós programozási felülete, mely lehetővé teszi a videokártya alacsony szintű programozását, egy Assembly-hez hasonló nyelven. Ez valódi keresztplatformos megoldás is egyben.

Létezik például a Quake játéknak is Flash átirata, vagy a közismert Epic Citadel demo is évekkel korábban elérhető volt Flash platformon, mint a WebGL verzió.

Programozása picit bonyolult, ezért számos további API-t készítettek, melyekkel egyszerűbben programozható.

A Stage3D weboldala:
http://www.adobe.com/devnet/flashplayer/stage3d.html

2.3.6. DirectX

A DirectX a Microsoft multimédiás platformja, mely nem csupán grafikus programozást tesz lehetővé, hanem teljes multimédiás fejlesztést is (például audio, videofeldolgozás stb.).

Előnye, hogy a Windows-ra épül, ami lényegében uralja az asztali számítógépes játékpicaot. Hátránya viszont, hogy kizárólag Windows platformon használható, azaz ebben az esetben le kell mondanunk a keresztplatformos álmainkról.

Programozásához külön SDK-t is le kell tölteni és telepíteni.

A DirectX weboldala:
https://msdn.microsoft.com/en-us/library/windows/desktop/ee663274(v=vs.85).aspx

2.4. A Microsoft grafikus programozási felületei

A Microsoft többféle grafikus programozási felületet kínál. Ezeket tekintjük át a következőkben röviden.

2.4.1. GDI

A GDI a Windows legegyszerűbb grafikus programozási felülete. A közhiedelemmel ellentétben bizonyos mértékű grafikus gyorsítást tartalmaz, számos dolgot viszont nem támogat, például az élsimítást sem. Előnye, hogy nagyon gyorsan megtanulható és az ezzel elkészített programokhoz semmiféle kiegészítőt nem kell telepíteni, ráadásul a technológia egészen akár Windows 98-ig visszafelé is kompatibilis.
További információk:
https://msdn.microsoft.com/en-us/library/windows/desktop/dd145203(v=vs.85).aspx

2.4.2. GDI+

A GDI+ a GDI továbbfejlesztett, ha úgy tetszik: kibővített változata. A videokártyák több képességét használja ki, viszont programozása is bonyolultabb, mint az "alap" GDI-é.
További információk:
https://msdn.microsoft.com/en-us/library/windows/desktop/ms533798(v=vs.85).aspx

2.4.3. Direct2D

A Direct2D a DirectX része és kifejezetten 2D megjelenítésre van kihegyezve. Erőssége a 2D tartalmak nagyon jó minőségű megjelenítése.
További információk:
https://msdn.microsoft.com/en-us/library/windows/desktop/dd370990(v=vs.85).aspx

2.4.4. Direct3D

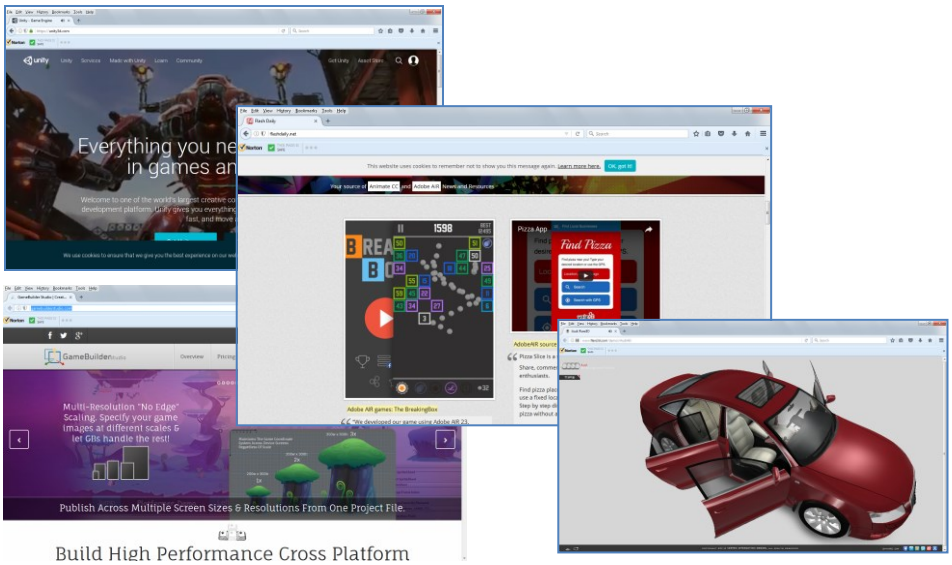
A Direct3D a DirectX legismertebb része, segítségével háromdimenziós megjelenítést alkalmazó programokat lehet írni.
További információk:
https://msdn.microsoft.com/en-us/library/windows/desktop/hh309466(v=vs.85).aspx

2.4.5. OpenGL implementáció

A Microsoft Windows beépített módon támogatja az OpenGL-t, így akár a Visual Studio fejlesztőeszköz segítségével is programozhatjuk.

2.5. Játékfejlesztő platformok

A játékfejlesztő platformok ingyenes, vagy fizetős szolgáltatásként lehetőséget kínálnak a programozóknak arra, hogy egy komplett eszközkészletet felhasználva egyszerűbben kihasználhassák egy-egy grafikus API képességeit, semmint a nulláról indulva fejlesztenék azt ki. Ide tartozik például a Unity, az Unreal Engine, az Ogre stb.



Unity https://unity3d.com/
Unreal Engine: https://www.unrealengine.com/
Google blocks: https://vr.google.com/objects
Ogre: http://www.ogre3d.org/
Adobe Gaming SDK: http://www.adobe.com/devnet/games/gaming.html
GameBuilder Studio: http://gamebuilderstudio.com/
Flare 3D: http://www.flare3d.com/
Feathers: https://feathersui.com/
Away 3D: http://away3d.com/
Starling: http://gamua.com/starling/
Flashdaily: http://flashdaily.net/

A kétségtelen előnyök mellett ezen platformok óriási hátránya, hogy jelentős időbe kerül, mire megtanulja az ember a használatukat, valamint az, hogy a működésüket teljes mélységében nem lehet átlátni, mivel hatalmas kódbázisra épülnek. Könyvünk témájához hozzátartoznak, ezért meg kell említenünk őket, de mi nem ezt az utat fogjuk követni, hanem saját, pehelysúlyú algoritmusokat fogunk használni játékaink elkészítéséhez.

2.6. OpenGL vs DirectX

A DirectX a Microsoft fejlesztése, míg az OpenGL egy nyílt szemléletű, többé-kevésbé független módon fejlesztett rendszer.

Az OpenGL és a DirectX ugyanazt csinálja, csak más programozási eszközökkel. Egyik sem jobb a másiknál e tekintetben.

A különbség egyrészt az, hogy a DirectX az idők során már nem csupán a grafikus megjelenítésről szól, hanem tágabb értelemben vett multimédiás programozási felületté vált. E tekintetben több, mint az OpenGL, de nem jobb.

A másik, szintén lényeges különbség, hogy az OpenGL keresztplatformos megközelítésre törekszik. Sajnos ez a gyakorlatban nem valósul meg száz százalékosan, mert a támogatottság adott esetben (például MAC OS X példája) mindezek ellenére mégis platformfüggő lehet.

Nagy általánosságban elmondhatjuk, hogy a DirectX-et Windows-on játékfejlesztésre, míg az OpenGL-t professzionális alkalmazásoknál használják inkább. Régebben még az OpenGL-t is használták komolyan játékfejlesztésre, lásd például a Quake játékokat.

3. FEJLESZTŐESZKÖZÖK BEMUTATÁSA

Ideje megismerkedni azokkal a fejlesztőeszközökkel, melyeket könyvünk példaprogramjainak az elkészítéséhez is felhasználunk. Igyekszünk lehetőleg ingyenes eszközöket bemutatni.

Megmutatjuk, honnan szerezhetjük be őket, hogyan telepíthetjük és vehetjük használatba ezeket a nagyszerű programokat.

3.1. Programozási nyelvek és online dokumentációk

Napjainkban kiemelt fontosságú, hogy egy adott programozási nyelvhez milyen online referencia leírása érhető el. Könyvünk mindhárom célnyelvéhez kiváló online dokumentációk érhetőek el, melyek segítségével a legmélyebb technikai kérdéseinkre is választ kaphatunk és rengeteg példaprogramot is találunk. Ezeket a helyeket mutatjuk be most röviden.

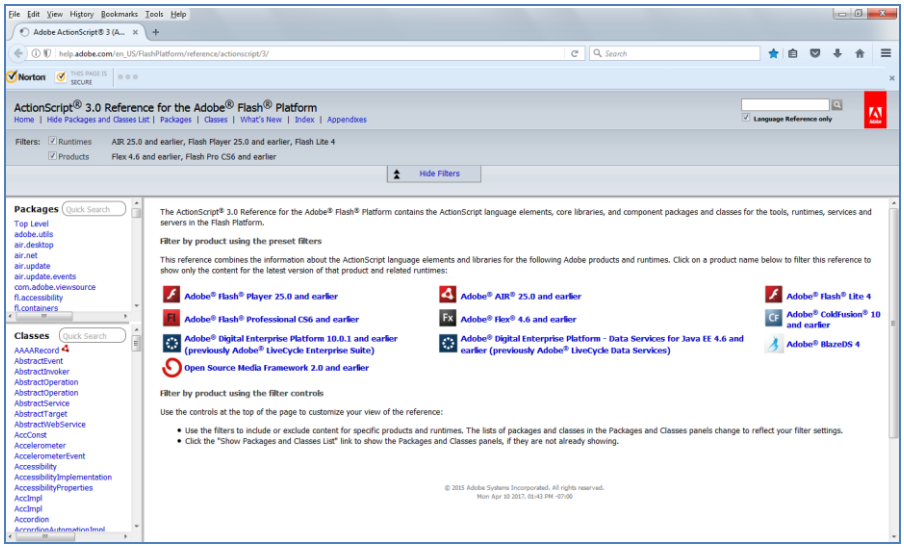
Megjegyezzük továbbá, hogy könyvünk függelékében nyomtatásban elérhető szakkönyvek listája is megtalálható.

3.1.1. Adobe AIR, ActionScript, Flex

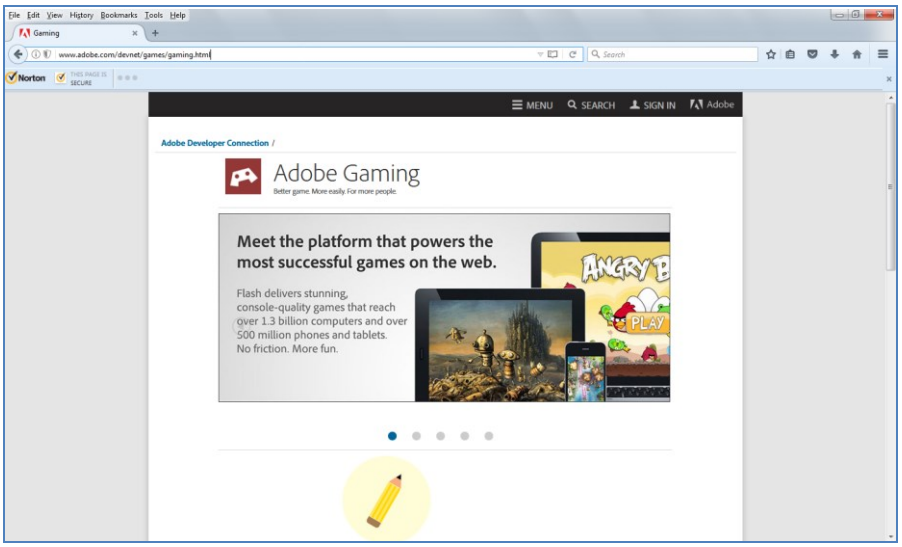
Az Adobe Flash platformjára épülő AIR keretrendszer igazi, vérbeli játékfejlesztői platform. Ismeretével egyszerre több operációs rendszerhez is készíthetünk alkalmazásokat, nagyon könnyen.

ActionScriptben programozható, mely egy EcmaScript szaványra épülő modern programozási nyelv.

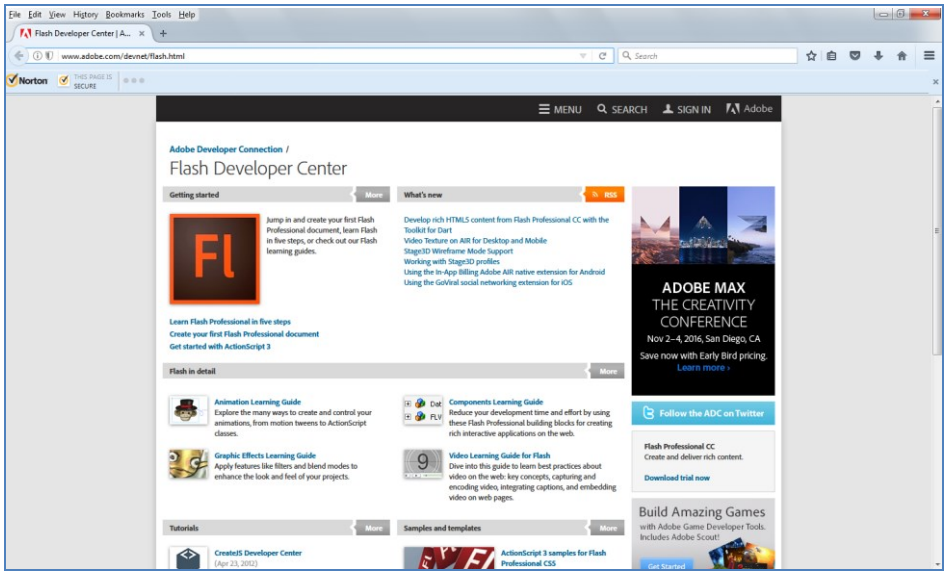
A Flash platform központi referencia oldala:
http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/



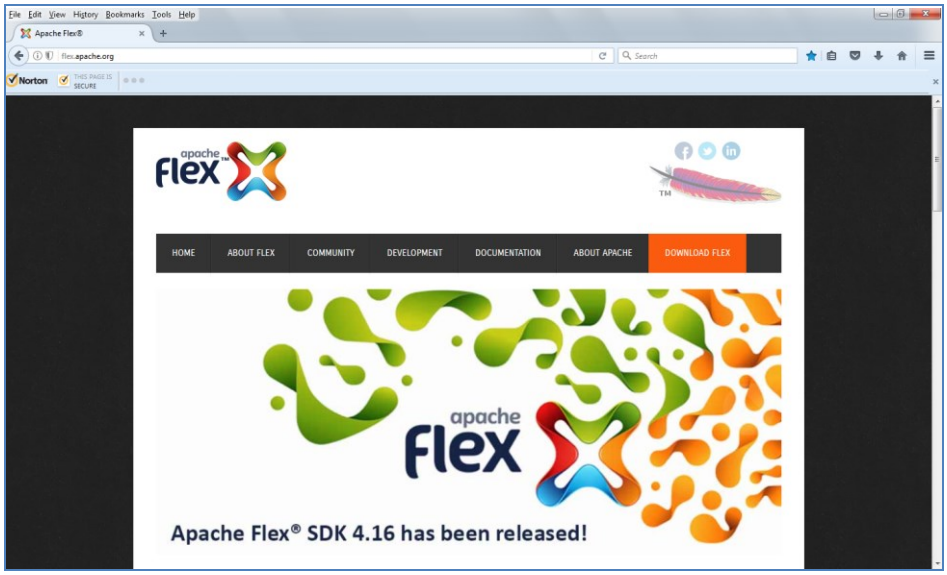
Az Adobe játékfejlesztési központi oldala:
http://www.adobe.com/hu/products/gaming-sdk.html
és
http://www.adobe.com/devnet/games/gaming.html



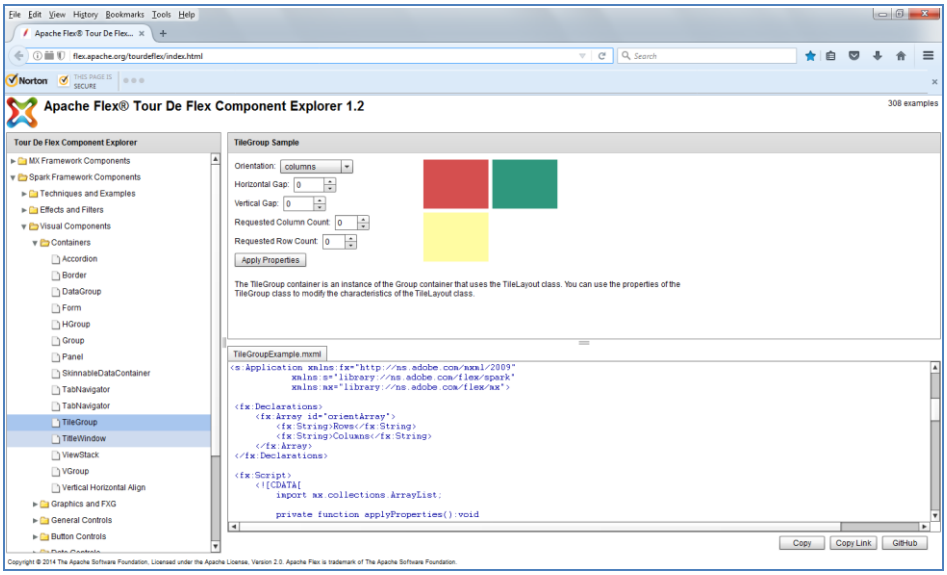
A Flash Developer Center oldala (hasznos linkekkel):
http://www.adobe.com/devnet/flash.html



A Flex keretrendszer központi oldala:
http://flex.apache.org/



Tour de Flex (óriási Flex példagyűjtemény):
http://flex.apache.org/tourdeflex/index.html



3.1.2. MSDN, WIN32 API, Visual C++

A **Microsoft Fejlesztői Hálózata (Microsoft Developer Network - MSDN)** valóságos aranybánya a natív Windows fejlesztések iránt érdeklődőknek. Tényleg csak szuperlatívuszokban beszélhetünk róla. Az alábbiakban néhány linket közlünk, melyekkel bátran elindulhatunk.

Könyvünkben a WIN32 API-ban történő programozást is bemutatjuk, mely a natív Windows alkalmazások készítésének kifejezetten technikai megközelítése és C/C++-ban történik. Az elnevezésben szereplő 32-es szám ne tévesszen meg senkit, a 64 bites alkalmazások is ide tartoznak! Ennek ellenére, ahol csak lehetséges, megmaradunk a sztenderd C nyelv szintaktikája mellett.

A WIN32 API programozásában történő kezdeti eligazodást segítik ezek az oldalak:
https://msdn.microsoft.com/en-us/library/aa271855(v=vs.60).aspx

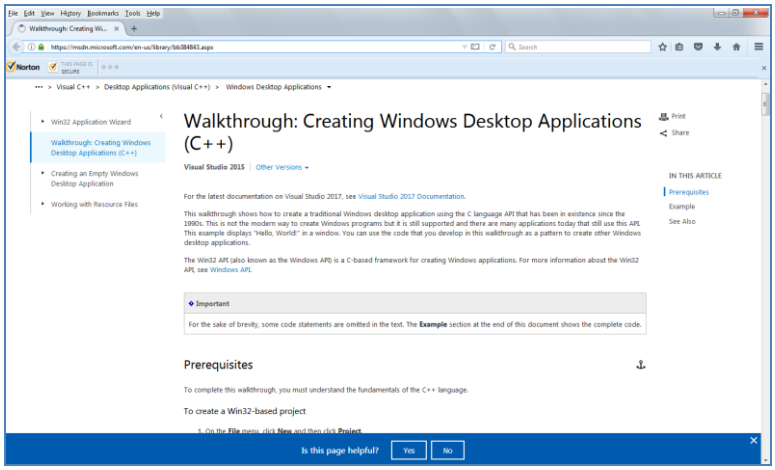
https://msdn.microsoft.com/en-us/library/bb384843.aspx

https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516(v=vs.85).aspx

A kifejezetten Visual Studio-val történő fejlesztéshez adnak remek bevezetőt ezek a cikkek:

https://docs.microsoft.com/en-us/cpp/windows/desktop-applications-visual-cpp

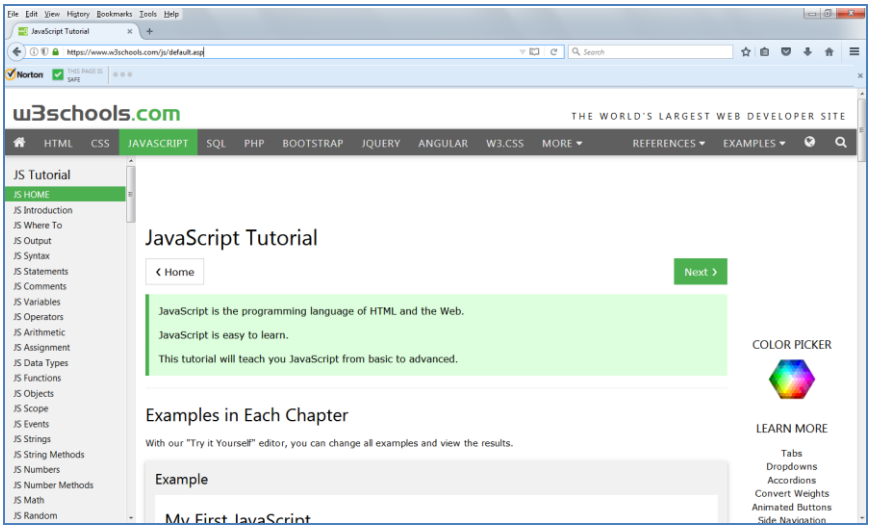
https://msdn.microsoft.com/en-us/library/60k1461a(v=vs.140).aspx



3.1.3. HTML5 / JavaScript

Igen jó felépítésű és rengeteg példával megtűzdelt dokumentációt találunk a W3schools oldalán mind HTML5, mind JavaScripthez (és még számos más webes programozási eszközhöz is):

https://www.w3schools.com/js/default.asp

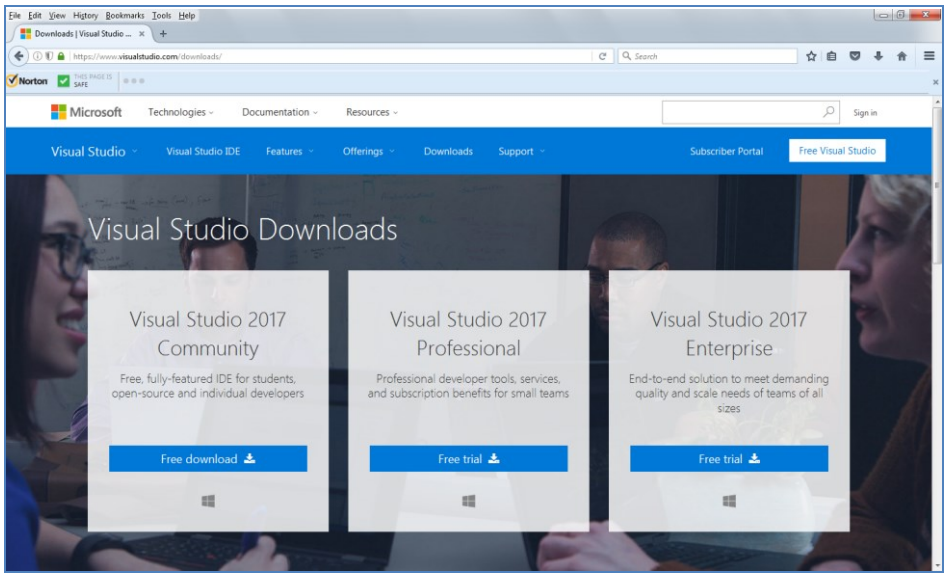


A példakódok óriási előnye, hogy a böngészőből közvetlenül kipróbálhatóak, kvázi fejlesztőeszközként is használhatjuk így webböngészőnket.

3.2. Microsoft Visual Studio Community Edition

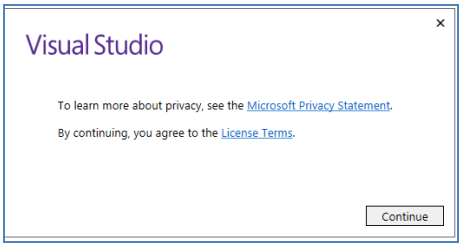
A Visual Studio a Microsoft hivatalos fejlesztőeszköze. Professional kiadása fizetős, ám egy ideje létezik nyílt forrású verziója is, a 'Microsoft Visual Studio Community 2017' ami teljesen ingyenes, mindössze egy ingyenes Microsoft fiókot kell megadnunk a korlátlan idejű használatához. A Community Edition a korábbi Express Edition felváltásának is tekinthető.

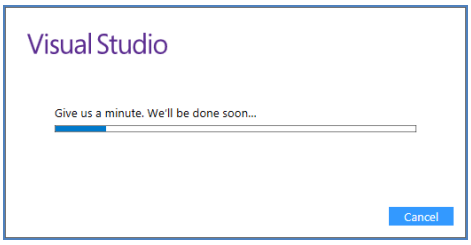
Az alábbi webhelyről tölthető le:
https://www.visualstudio.com/downloads/



A 'Free download' linkre kattintva először is le kell töltenünk egy online telepítőalkalmazást, melyet elindítva kezdhető meg a tényleges telepítés.

Először el kell fogadnunk a Microsoft liszensz feltételeit.

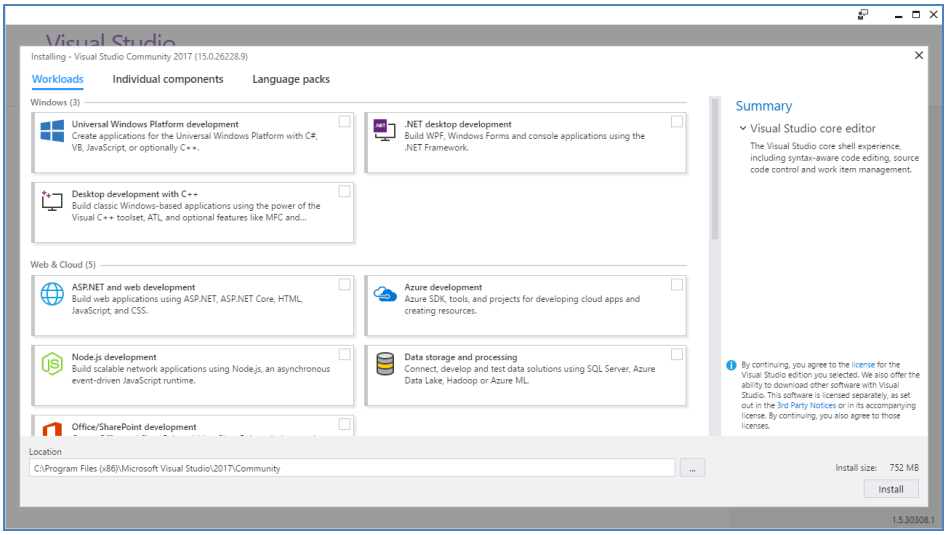




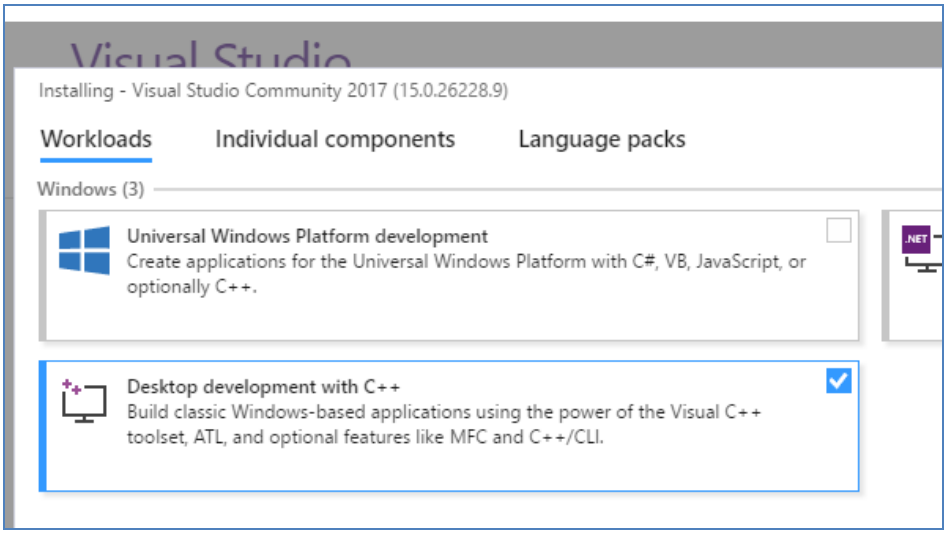
Ezt követően a telepítő felkészül a telepítés előkészítésére.

A Visual Studio Community 2017-es kiadása a megelőző, 2015-ös kiadáshoz képest jelentősen egyszerűsített telepítést tesz lehetővé.

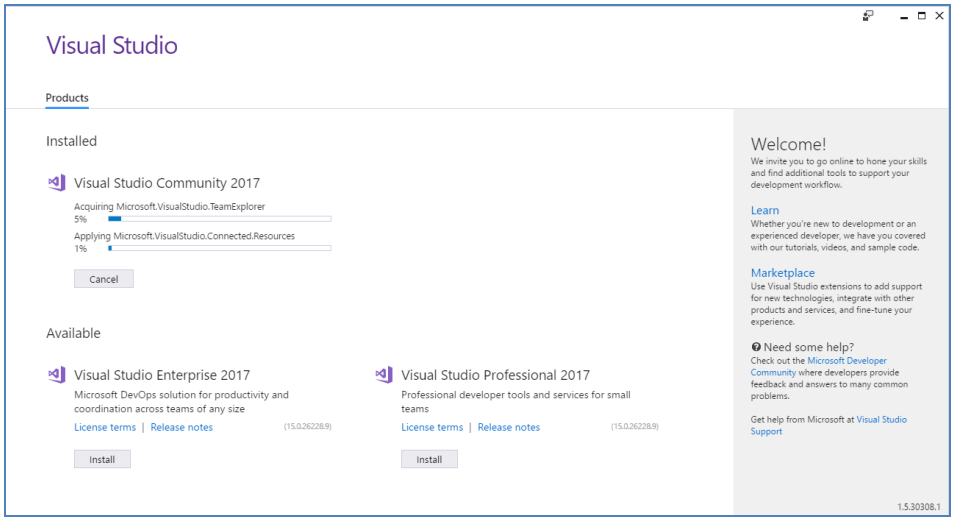
Néhány kategóriába foglalva választhatjuk ki, hogy mire szeretnénk használni az eszközt. Így nem kell részletekbe menően meghatározni és találgatni, mely komponensekre lehet szükségünk.



Könyvünk példaprogramjaihoz a "Desktop development with C++" beállításcsoportot elegendő kiválasztani.



Az 'Install' gombra kattintva kezdődik el maga a telepítés. A szoftver esetünkben kb. 6GB tárhelyet igényel. A telepítés néhány percig tart, egészen gyorsan befejeződik.



Telepítés közben figyelemmel kísérhetjük a folyamat állását.

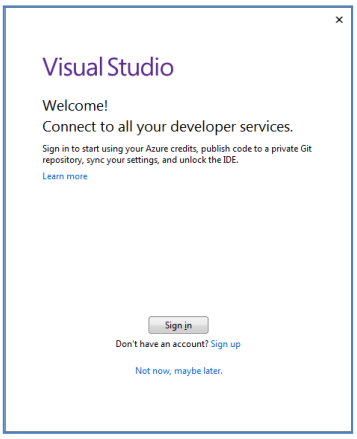
Mielőtt használatba vennénk a fejlesztőrendszert, jó eséllyel újra kell indítanunk a számítógépünket.

A számítógép újraindulása után befejeződik a telepítés. Gépünkön a fejlesztőrendszer indítóikonja nem került ki automatikusan az Asztalra, ezt manuálisan végeztük el.

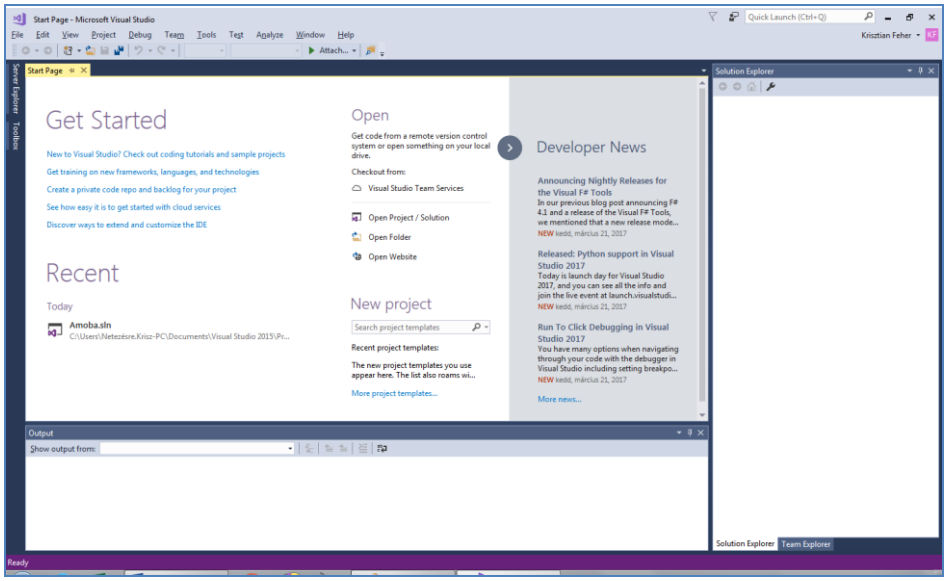
3.2.1. A fejlesztőeszköz kipróbálása

Ajánlatos a fejlesztőeszközt rendszergazdai jogosultságokkal elindítani, hogy ne legyen gond könyvtárakhoz történő hozzáféréssel, stb.

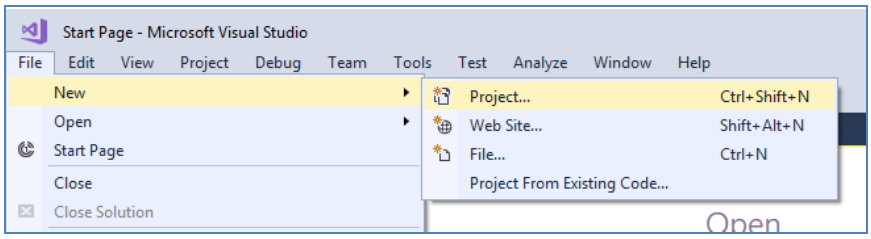
A Visual Studio első indításakor be kell jelentkeznünk Microsoft fiókunkkal. Ezt a lépést kihagyhatjuk ugyan, ám ebben az esetben kb. 30 nap múlva nem használhatjuk teljes funkcionalitással az eszközt.



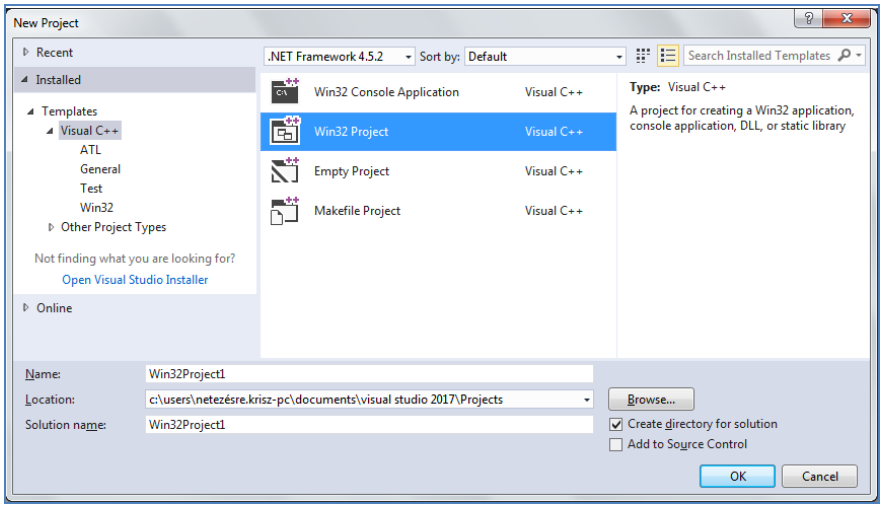
Ha végeztünk, megjelenik a kezdőképernyő.



Kattintsunk a File - New - Project... menüpontra!

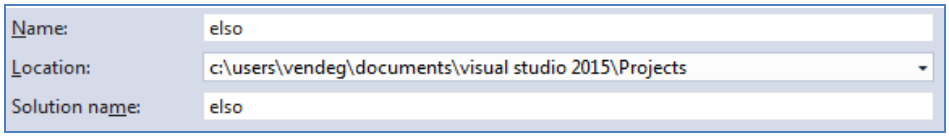


Ekkor választhatunk egy projekt sablont. Válasszuk ki a Templates - Visual C++ - Win32 - Win32 Project elnevezésű sablont!

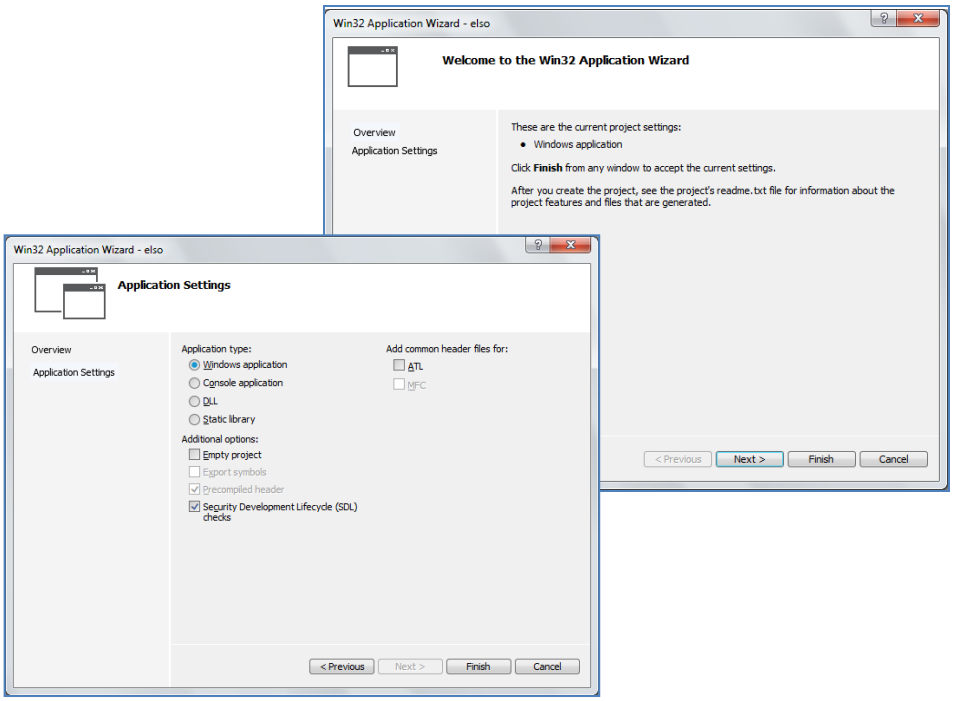


A Visual Studio projekteket és ún. megoldásokat (solution) kezel. Egy projekt alá több solution is tartozhat, ezért fontos, hogy pontosan meg tudjuk különböztetni őket egymástól. Ha ezt nem preferáljuk, nyugodtan hozzunk létre minden programunk számára egy külön projektet.

Adjunk tehát projektünknek és megoldásunknak nevet (esetünkben ez most 'elso') és adjuk meg a projekt tárolásának helyét!

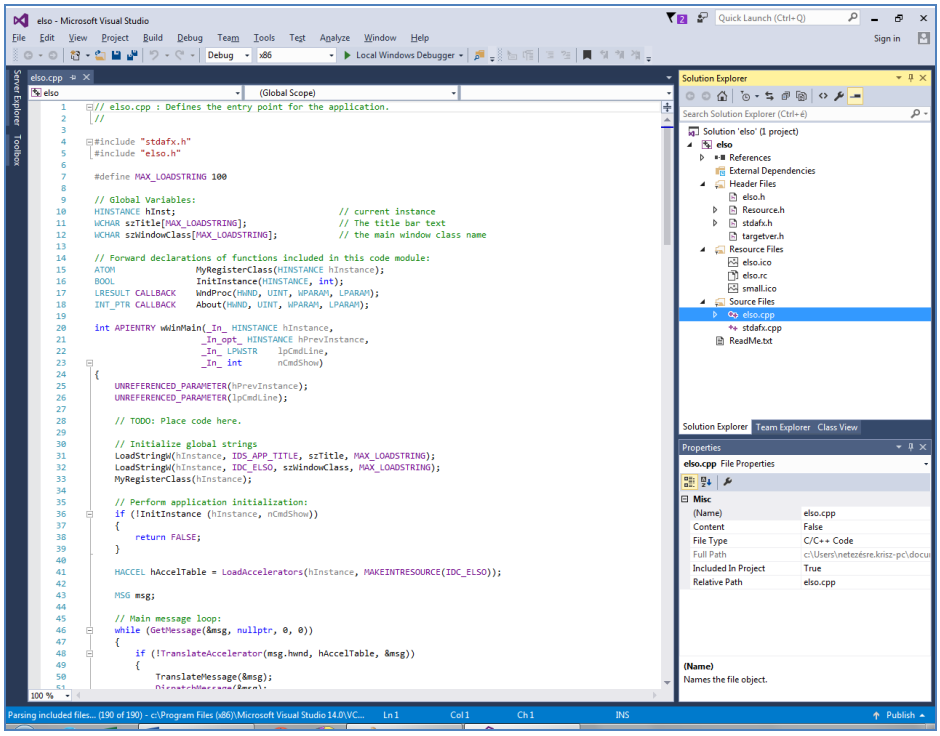


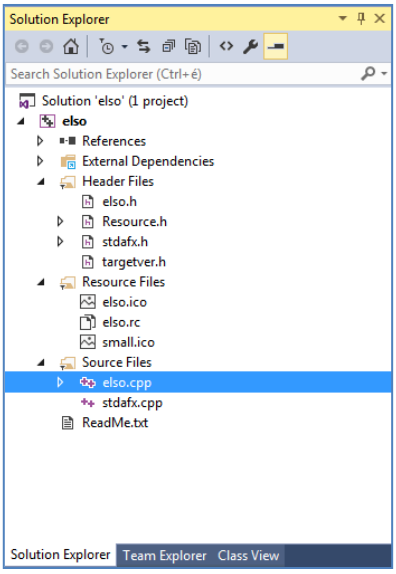
A többi beállítás maradhat az alapértékeken. Ezután az 'OK' gombra kell kattintanunk. Megjelenik egy leegyszerűsített varázsló, amely a projekt néhány alapvető konfigurációs lépésében nyújt segítséget. Nem kötelező használni (egyből kattinthatunk a 'Finish' gombra), de ajánlott, vagy legalábbis kényelmes.



Amennyiben a 'Next' gombra kattintunk, az alkalmazás típusát állíthatjuk be.

Az alapértelmezett beállítás a 'Windows application', nekünk pontosan ez kell. Kattintsunk a 'Finish' gombra, hogy létrejöjjön a projekt!





Vessünk egy pillantást az ablak jobb oldalán található 'Solution explorer' dokkolt ablakra! Ez az ablak tartalmazza forrásfájljainkat, szép faszerkezetben elrendezve.

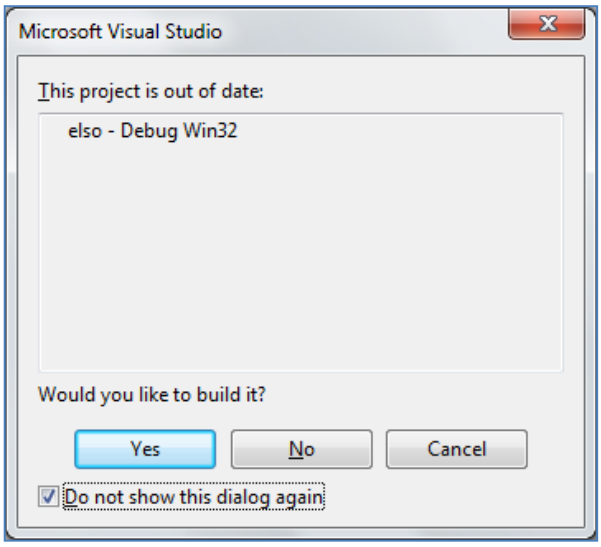
Figyeljük meg, hogy alkalmazásunk forráskódja, az ELSO.CPP, a Solution 'elso' nevű projektjének 'Source Files' "mappájában" található.

A Visual Studio standard C nyelvű projekteket nem támogat, csak C++ projekteket. Mivel azonban a C++ teljesen kompatibilis a C nyelvvel, ez számunkra nem jelent gondot.

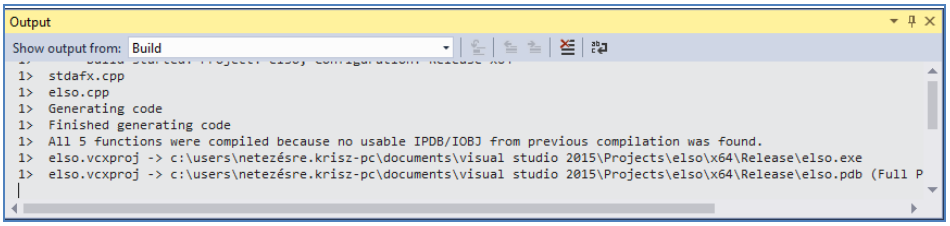
A képernyő nagy részét alkalmazásunk forráskódja foglalja el. Ez esetünkben egy kódsablon, ami egy egyszerű ablakos alkalmazás. Ne ijedjünk meg a közel 200 sor láttán, a későbbiekben minden részletet elmagyarázunk.

Egyelőre fordítsuk le és indítsuk el az alkalmazást! Ehhez az ablak felső részén, az ikonsorban található ikonra kell kattintanunk.

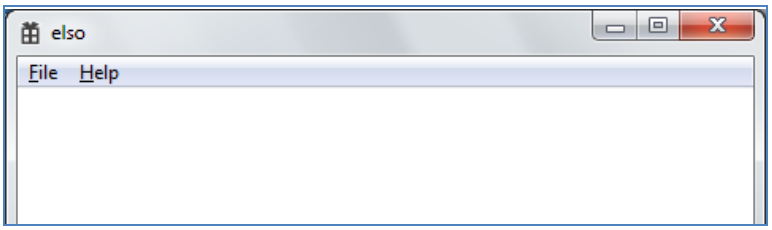
Előfordulhat, hogy az alábbi ablak jelenik meg, mely a forrásfájl mentésére hívja fel a figyelmet. Nyugodtan válasszuk a 'Yes' lehetőséget, de előtte pipáljuk ki 'Do not show this dialog again' opciót, hogy a jövőben ez a lépés automatikusan végrehajtódjon.



A fordítás folyamatát figyelemmel kísérhetjük az 'Output' ablakban, a képernyő alsó részén.



Ha minden jól megy, akkor megjelenik alkalmazásunk ablaka.

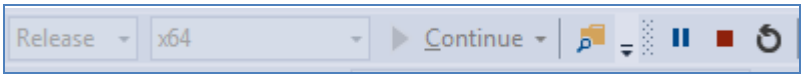


Ha emlékszünk még, a fordítási opciók ezeket a beállításokat tartalmazták:

Tehát egyelőre egy Debug verzió jött létre alkalmazásunkból. Ez futtatható ugyan, de rengeteg járulékos adat is beépül az alkalmazásba, ami a hibakeresést elősegíti, de megnöveli az alkalmazás méretét és lassabbá is teszi a végrehajtását. Ahhoz, hogy programunk publikálásra kész változatát elkészítsük, állítsuk be a Release - x86 / x64 lehetőségeket. Az alábbi beállításokkal például programunk 64 bites változata fog lefordítódni.

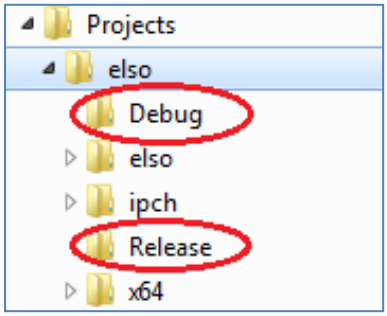
Ha elindítjuk ezekkel a beállításokkal az alkalmazást, látszólag nem történik semmi újdonság, ám ez már egy "éles" verziójú alkalmazás.

Vegyük észre, hogy miközben a fejlesztőrendszerből elindítjuk alkalmazásunkat, az indító ikon helyett más ikonok jelennek meg:



Ezek funkciója a következő:

- Break All: megállítja az alkalmazás futását a legközelebbi töréspontnál. Ilyenkor a futtatást a Continue ikonnal folytathatjuk.
- Stop Debugging: leállítja az alkalmazást.
- Restart: újraindítja az alkalmazást.



Felmerül a kérdés: hol találjuk a lefordított alkalmazásunkat? A futtatható EXE állományok a projektünk 'Debug' és 'Release' mappáiban lesznek megtalálhatóak. Esetünkben például a képen látható helyeken.

Amennyiben publikálni akarjuk alkalmazásainkat, vagy csak egyszerűen a Visual Studio nélkül futtatni, nyugodtan elindíthatjuk az itt található lefordított programokat. Arra kell csak ügyelnünk, hogy mindig friss fordítással rendelkezzünk.

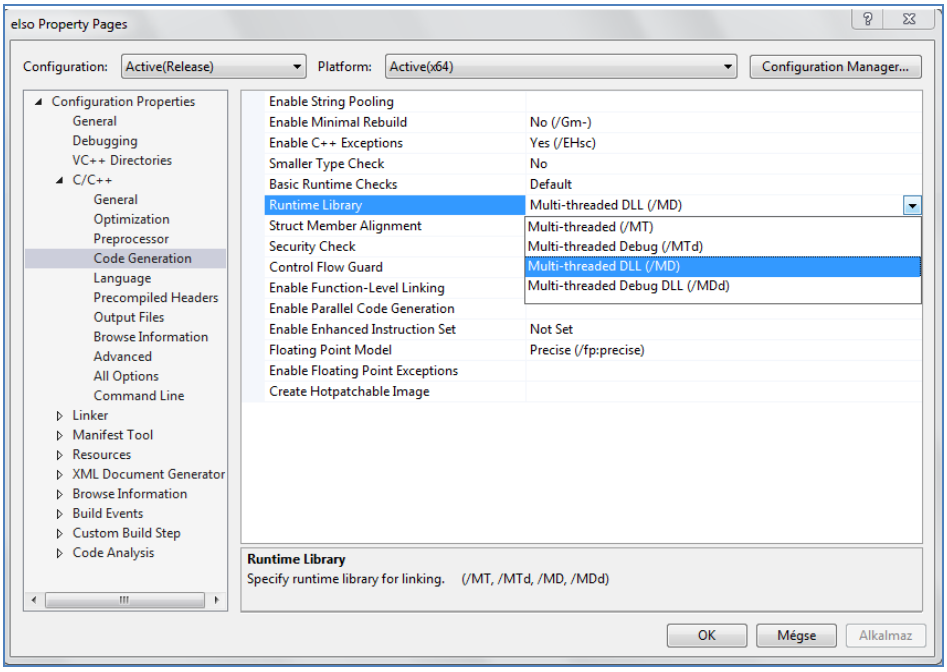
A végére hagytuk a legjobb hírt: a WIN32API alkalmazások nem igényelnek telepítést!

Az .EXE fájlokat egyszerű másolással áthelyezhetjük egyik gépről a másikra. Egy dologra azonban figyeljünk: amennyiben idegen gépre telepítjük az alkalmazásunkat, előfordulhat, hogy az MSVCP140.DLL fájlt a Windows hiányolni fogja. Ez Windows 10 rendszereken kevésbé

valószínű, de Windows 7 operációs rendszereken előfordulhat. A hiba oka, hogy a szükséges DLL nincsen hozzácsomagolva az alkalmazáshoz.

Erre többféle megoldás létezik. Az egyik megoldás, ha mi magunk gondoskodunk a szükséges kiegészítők hozzácsatolásáról a programunkba. Ez némileg megnövelheti az alkalmazás méretét.

Ehhez válasszuk ki a projektünket a Solution Explorer-ben, majd a jobb egérgombbal megjelenített helyi menüből válasszuk ki a 'Properties' pontot! Az ezután megjelenő projektablakban navigáljunk el a Configuration Properties - C/C++ - Code Generation - Runtime Library opcióhoz, ahol kiválaszthatjuk, hogyan készüljön el a végrehajtható állományunk.



A másik módszer a felhasználói oldalon nyújt megoldást. Ehhez le kell töltenünk és telepítenünk kell a Microsoft oldaláról a szükséges kiegészítőt:

https://www.microsoft.com/en-us/download/details.aspx?id=48145

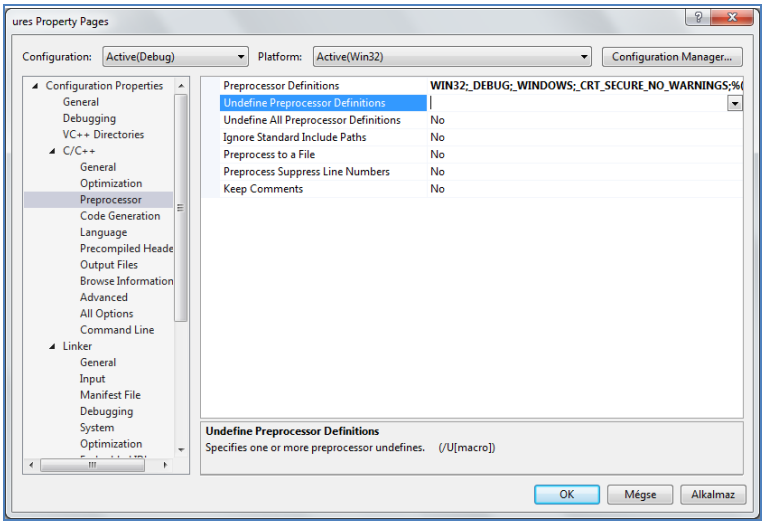
Ez egyáltalán nem különleges dolog, rengeteg játékprogram is telepít ilyen kiegészítőket, csak ezt legtöbbször automatikusan végzik el.

3.2.2. Régi típusú függvények - figyelmeztetés kikapcsolása

Akár könyvünk példáiban is előfordulhat egy-egy olyan függvény, ami a standard C nyelv univerzumában még teljes jogú állampolgár, ám a szép új modern világban már bizony elavultnak számít. Természetesen használhatjuk az új, biztonságosabbnak mondott függvény variánsokat is, ezek neve általában _s utótaggal végződik.

Ám ha mégis ragaszkodunk a régi függvények használatához és szeretnénk, ha az alkalmazásunk lefordulna, akkor be kell állítanunk az ún. _CRT_SECURE_NO_WARNINGS előfeldolgozó definíciót, ami kikapcsolja a figyelmeztetéseket.

A beállítást a projekt nevén a jobb egérgombbal kattintva tudjuk kezdeményezni. Menjünk a Properties - C/C++ - Preprocessor - Preprocessor definitions részhez, ahol írjuk be a sor végére a _CRT_SECURE_NO_WARNINGS bejegyzést, gondosan lezárva egy pontosvessző karakterrel, majd nyomjunk ENTER-t a véglegesítéshez!

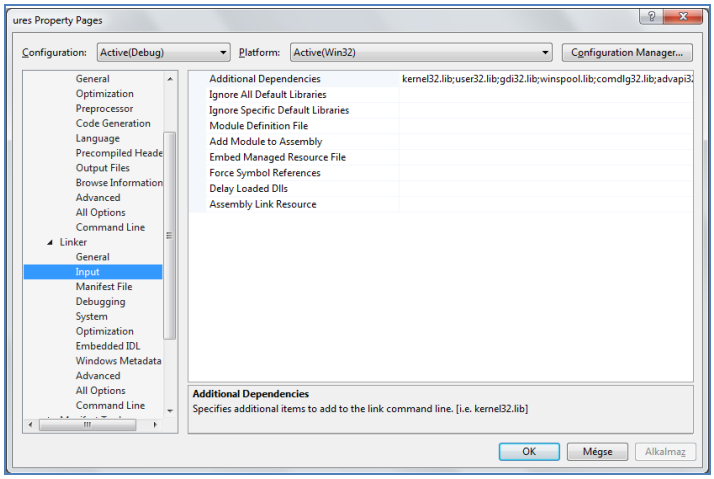


3.2.3. Külső könyvtárak hozzáadása projektekhez

Amennyiben külső könyvtárak használatára fanyalodnánk (nem kell nagy dolgokra gondolni, ide tartozik például az OpenGL is), akkor azt kétféleképpen tehetjük meg. A kényelmesebb, kissé több kattintgatást igénylő módszer a projekt tulajdonságai közt megadni a könyvtárat.

A beállítást a projekt nevén a jobb egérgombbal kattintva tudjuk kezdeményezni. Menjünk a Properties - Linker - Input - Additional Depen-

dencies részhez és adjuk meg a könyvtár nevét, majd utána írjunk még egy pontosvesszőt is.



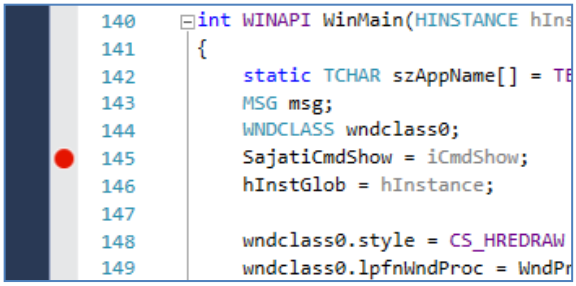
A másik, kissé technikásabb módszer, a programunk legelejére beírni a könyvtár nevét, a #pragma direktívával, például így:

```
#pragma comment(lib, "winmm.lib")
```

3.2.4. Hibakeresés Visual Studioban

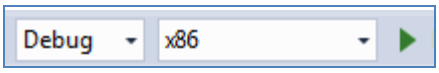
Alkalmazásaink tesztelése során óriási segítséget jelent a Visual Studio beépített **debugging (hibakereső)** szolgáltatása. Ennek segítségével ún. **töréspontokat (break points)** rendelhetünk a forráskód egyes soraihoz. A speciális, ún. debug futtatás során a végrehajtás minden esetben megszakad, valahányszor egy törésponthoz ér a végrehajtás.

A hibakeresési futtatás technikailag a futtatható EXE állomány szerkezetében is kiegészítéseket tesz szükségessé, ezért az ilyen módon lefordított programok mérete rendre nagyobb a nem debugging fordítású alkalmazásoknál. Ezek az alkalmazások rendre picit lassabban is futnak (legalább kb. 10%-kal).

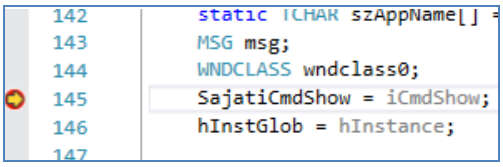


Töréspontot úgy tudunk létrehozni, hogy egy programsor mellett kattintunk a bal egérgombbal. (Ugyanígy távolíthatjuk is el a töréspontokat.) Ekkor egy piros pötty kerül a sor mellé.

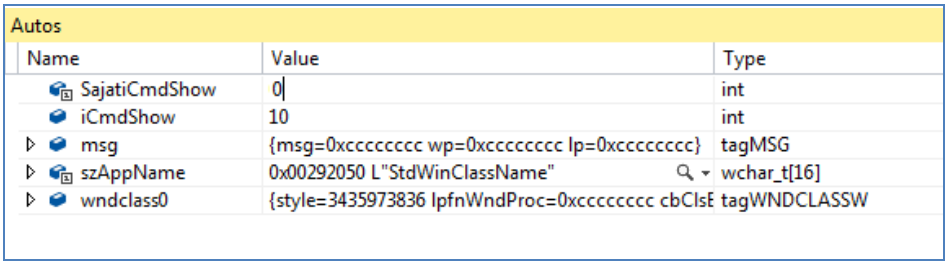
Mielőtt kipróbálnánk a dolgot, győződjünk meg róla, hogy Debug futtatási módot használunk az alkalmazás elindításához.



Amint a végrehajtás az adott sorhoz ér, a program futása megszakad és a fókusz visszakerül a Visual Studio-ra. A piros pöttybe is bekerül egy sárga nyíl (ez azért hasznos, mert elképzelhető, hogy több töréspontot is használunk egy helyen és így pontosan láthatjuk, melyiknél állt meg a végrehajtás.



A Visual Studio ablakának alsó sorában megjelenik a töréspont közelében található változók listája és a változók tartalma is.



A Visual Studio felső sorában a Continue gombra kattintva folytathatjuk a végrehajtást, de teljesen le is állíthatjuk, vagy újra is indíthatjuk az alkalmazást.

A hibakeresés sokszor az egyetlen módja annak, hogy megállapítsuk, alkalmazásaink miért nem úgy működnek, ahogyan szeretnénk. Éppen ezért programozói szemszögből óriási segítséget jelent.

3.3. DEV C++ - Alternatív C fejlesztőeszköz

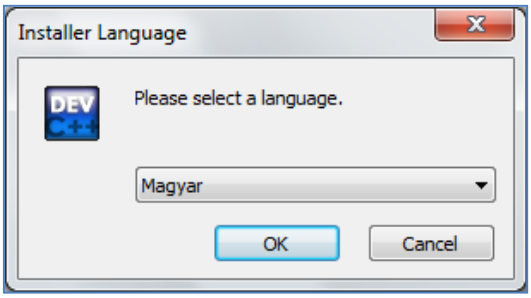
Amennyiben egy kisebb hely- és erőforrásigényű, de modern fordítóval ellátott fejlesztőeszközt keresünk, érdemes megfontolnunk a DEV C++ használatát.

Ez az eredetileg Bloodshed DEV C++ nevű, nyílt forrású eszköz egy továbbfejlesztett változata, mely GCC fordítót tartalmaz és alkalmas 32 és 64 bites Windows-os alkalmazások létrehozására is, akárcsak a Visual Studio.

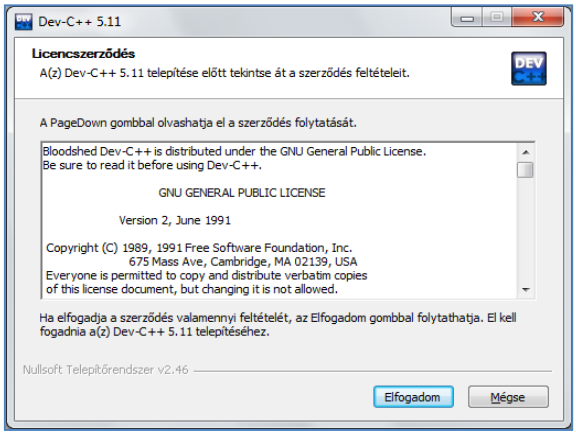
Az alábbi oldalról tölthetjük le a telepítőcsomagját:

https://sourceforge.net/projects/orwelldevcpp/

A telepítőcsomag mérete kb. 50 MB és a fejlesztőeszköz telepítve is csak kb. 350 MB tárhelyet igényel. A telepítőt elindítva először is kiválaszthatunk egy nyelvet.

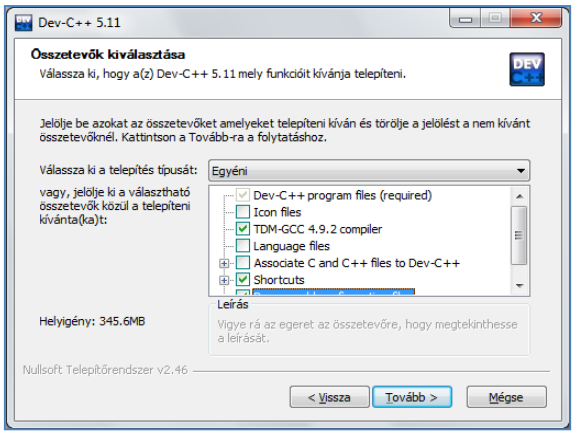


Ezt követően el kell fogadnunk a felhasználási feltételeket. Kattintsunk tehát az 'Elfogadom' gombra!

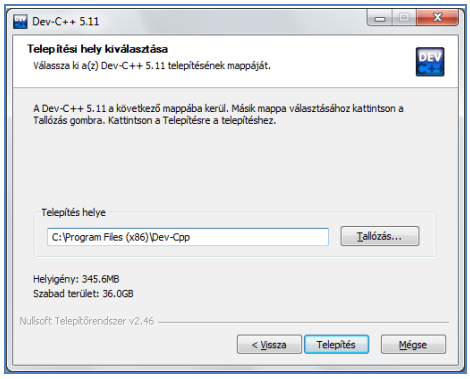


Ezután kiválaszthatjuk a telepítendő összetevőket. Fontos tudni, hogy a Dev C++ többféle fordítóval is képes együttműködni, ezért nem meglepő, hogy a TDM-GCC compiler telepítése is opcionális. Kezdőknek ezt az opciót mindenképpen érdemes kipipálniuk.

A folytatáshoz kattintsunk a 'Tovább' gombra!



A tényleges telepítés megkezdése előtt mégegyszer lehetőségünk nyílik ellenőrizni a telepítési beállításainkat, majd a 'Telepítés' gombra kattintva indíthatjuk el a folyamatot.

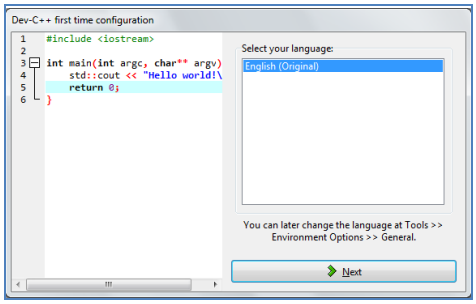


A telepítés folyamatát kísérhetjük nyomon a következő megjelenő képernyőn.

A telepítés igen hamar befejeződik és egyből el is indíthatjuk az alkalmazást.

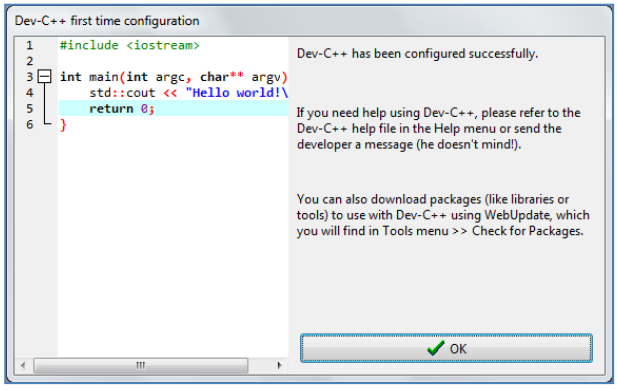
3.3.1. A fejlesztőeszköz kipróbálása

Az első indítás alkalmával el kell végeznünk néhány alapvető beállítást a külcsínt illetően. Elsőnek be kell állítanunk a kezelőfelület nyelvét.

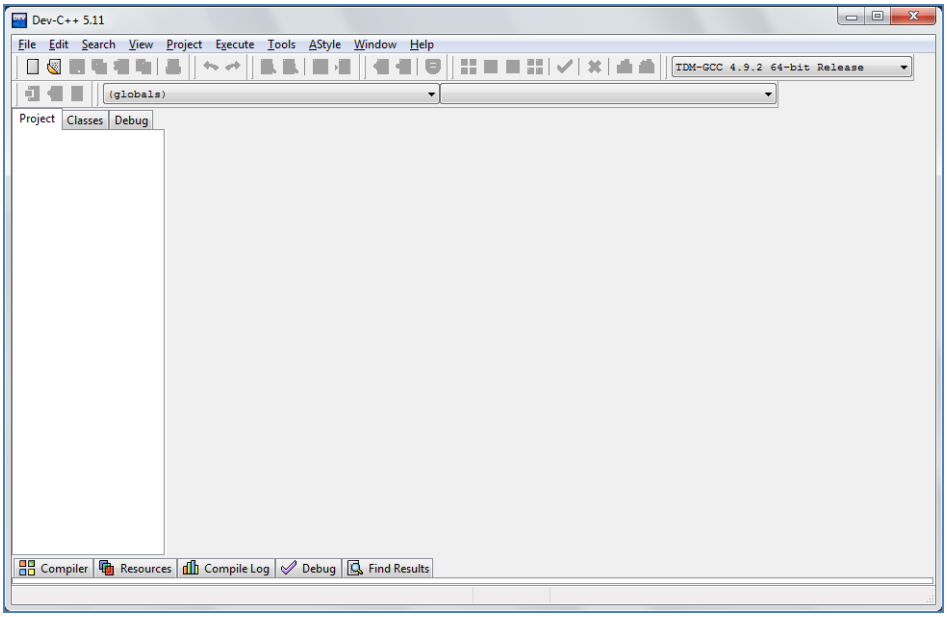


Ezt követően a kezelőfelület megjelenítési stílusát állíthatjuk be.

A konfigurációt végezetül az 'OK' gombra kattintva fejezhetjük be.



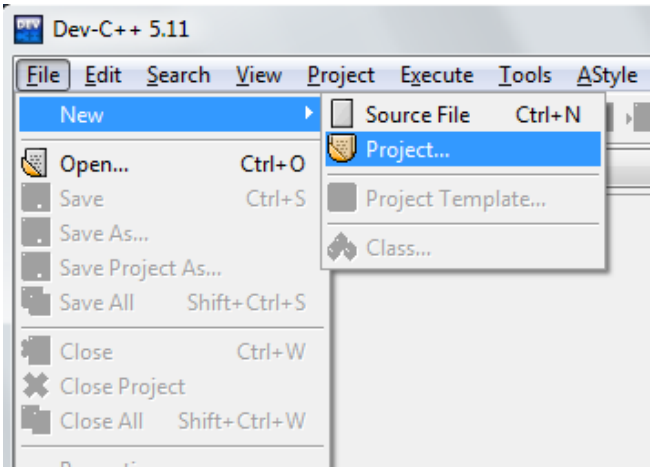
Ezután megjelenik a fejlesztőeszköz tényleges felülete.



Mielőtt folytatnánk az ismerkedést a fejlesztőrendszerrel, zárjuk be az alkalmazást és hozzunk létre egy parancsikont az asztalon az indításához!

Ezután állítsuk be, hogy az alkalmazás rendszergazdai jogosultságokkal induljon! Ez fontos lépés, ellenkező esetben ugyanis előfordulhat, hogy az alkalmazás nem találja az új projektek létrehozásakor a kódsablonokat, ill. akár a fordító elérési útját sem!

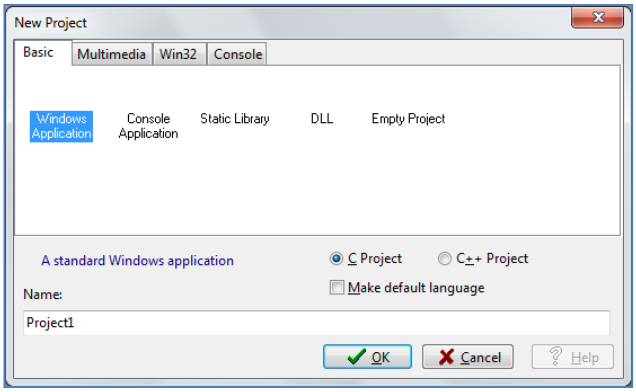
Indítsuk el tehát így ismét az alkalmazást! Kattintsunk a File - New - Project... menüpontra!



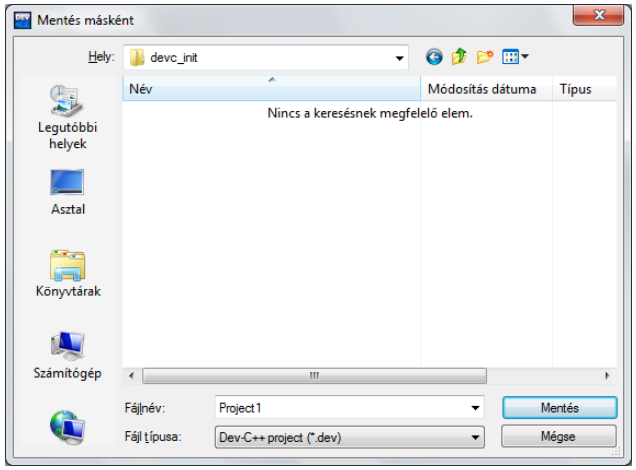
A megjelenő párbeszédablakban, a Basic fül alatt válasszuk ki a 'Windows Application' lehetőséget!

A projekt típusa legyen 'C Project'!

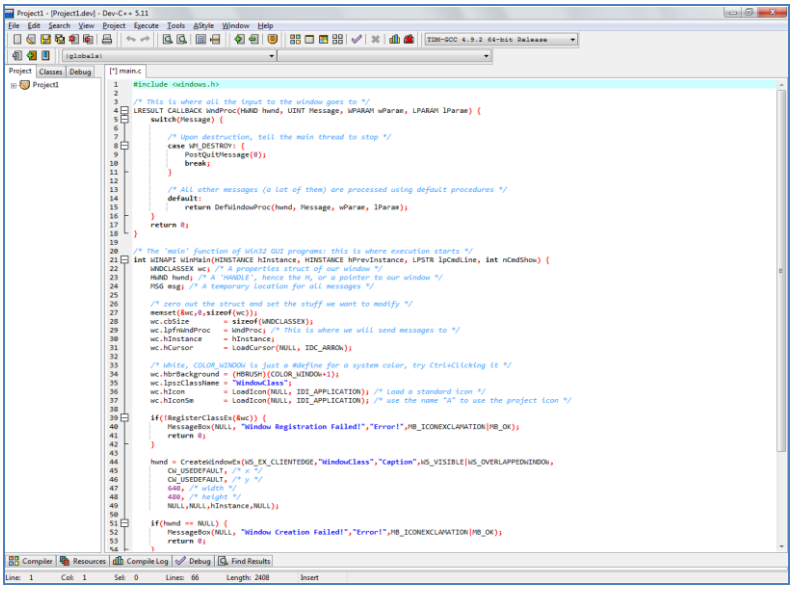
Kattintsunk az 'OK' gombra!



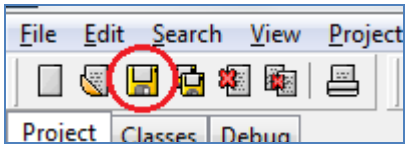
Ezután egyből el kell mentenünk a projektleíró fájlt. Ehhez hozzunk létre egy új, üres könyvtárat!

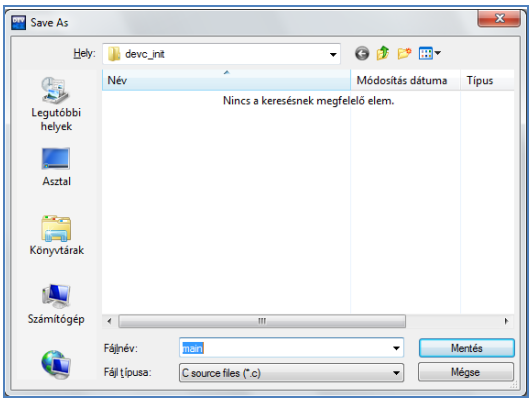


Ezt követően betöltődik egy C nyelvű kódsablon, ami egy standard ablakos alkalmazás forráskódja.



Mielőtt azonban kipróbálnánk, mentsük el a forrásfájlt is!

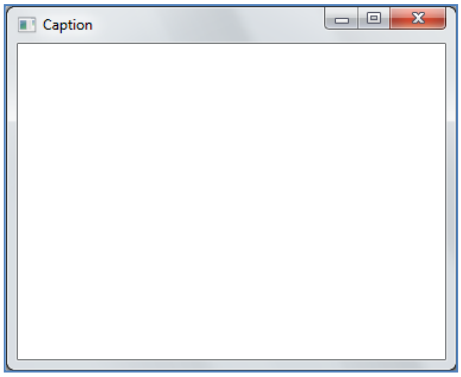




Ezután kattintsunk a 'Compile' ikonra, vagy nyomjuk meg az F9 billentyűt! A Dev C++ ablakának alsó részén erre valami hasonlónak kell megjelennie:

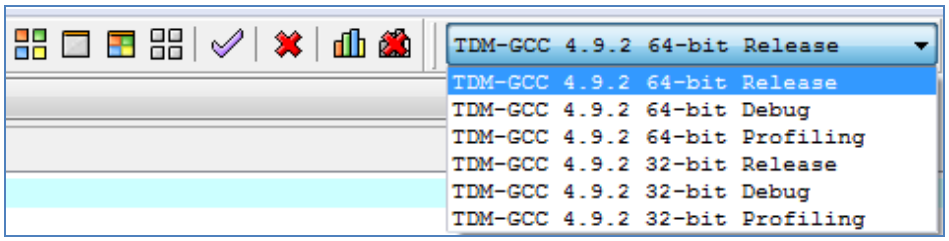
```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename:
C:\Users\Vendeg\Desktop\Könyvek\2D_3D_programozas\peldaprogramok\de
vc_init\Project1.exe
- Output Size: 134,4287109375 KiB
- Compilation Time: 2,36s
```

A lefordított alkalmazást a 'Run' ikonnal indíthatjuk el, vagy az F10 billentyű lenyomásával. (Megjegyezzük, hogy a 'Compile & Run' ikon a fordítást és a futtatást egyszerre végrehajtja.) Megjelenik alkalmazásunk ablaka.

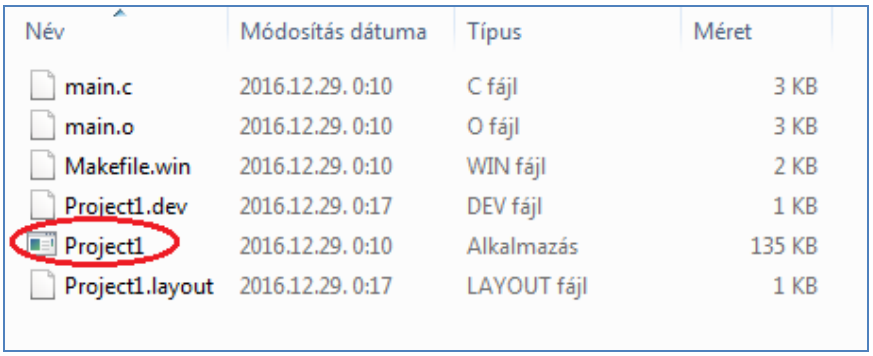


Alkalmazásunk alapértelmezetten egy 64 bites, azonnal publikálható EXE fájl lesz, ami bármilyen elterjedt mai Windows rendszeren futtatható.

Ezenkívül azonban többféle típusú végrehajtható fájlt is készíthetünk. Ezt egy listából állíthatjuk be, amely a fordítás és futtatás ikonjaitól jobbra helyezkedik el.



Hol találjuk meg a lefordított programunkat? A projektünk mappájában. A Visual Studio-val ellentétben a DEV C++ egyetlen mappát használ, ahová mindent "bepakol". Itt található programunk is. A Debug és Release verziókról nem készül külön változat, mindig csak egy EXE fájlt fogunk itt találni.



Jó tudni, hogy a Dev C++-szal készült alkalmazások általában nem igényelnek külső DLL-t, így még egyszerűbben használhatóak más gépeken is, mint a Visual Studio-val készült alkalmazások.

A tapasztalat azt mutatja, hogy a GCC fordítóval készült alkalmazások enyhén lassabban futnak, mint a Microsoft Visual C++ fordítójával készítettek.

Megjegyezzük, hogy a Dev C++ is támogatja a debuggingot. Erre külön nem térünk ki, mert a folyamat nagyban hasonlít a Visual Studionál bemutatott módszerre.

Ezzel a végére értünk a DEV C++ előkészítésének.

3.4. A Flash Builder

3.4.1. Az Adobe AIR platformról

Az AIR az **Adobe Integrated Runtime** elnevezés rövidítése és egy olyan technológiát takar, mellyel Flash-alapú alkalmazásokat böngészőn kívül, az operációs rendszer natív alkalmazásaként lehet futtatni. Ezzel a

Flash rugalmasságát a natív alkalmazások előnyeivel (pl. hozzáférés a lokális erőforrásokhoz) lehet ötvözni.

A programozás során használt programozási nyelv, az **ActionScript**, ECMA szabványra épülő nyelv, akárcsak a Java. Az Actionscript lehetővé teszi, hogy a Flash példátlanul látványos multimédiás képességeit interaktivitással ruházzuk fel.

Az AIR alkalmazások készítését külön SDK-val lehet elvégezni. Az AIR SDK ingyenesen letölthető az Adobe weboldaláról:

http://www.adobe.com/devnet/air/air-sdk-download.html

Ám leghatékonyabban valamilyen integrált fejlesztői környezettel lehet kihasználni a technológia összes képességét.

Az AIR fejlesztések páratlan előnye, hogy ugyanazzal a kódbázissal többféle platformra készíthetünk alkalmazásokat, a kód újraírása nélkül. Így, aki megtanul AIR alkalmazásokat írni, „egy csapásra" képes Windows, Mac OSX, Android, iOS, web böngészős, de még SmartTV rendszerekre is alkalmazásokat készíteni.

Az ActionScript-alapú fejlesztésekkel karöltve zajlott a Flex keretrendszer fejlesztése is. A Flex nem más, mint egy ActionScriptben megírt osztálygyűjtemény és architektúra, mely az alkalmazások kezelőfelületének az elkészítésére is használható. A Flex-ben írt alkalmazások forráskódja XML alapú és nagyon könnyen olvasható. Az Adobe a Flex keretrendszer gondozását 2012-ben az Apache alapítványnak adta át.

Az ActionScript aktuális verziója ezen könyv megírásakor a 3.0, a Flex jelenleg a 4.16-os verziónál tart.

Mobilalkalmazásokat Flex és tisztán ActionScript alapon, vagy keverten is fejleszthetünk. Ez utóbbi megoldást mutatja be a könyv is.

3.4.2. A Flash Builder és a mobilvilág

Az ActionScript-alapú fejlesztőeszközök két vezető eszköze az Adobe Flash Builder és az Adobe Flash Animate (régebben: Flash Professional). Előbbi egy Eclipse-alapú, komplett fejlesztőrendszer, utóbbi pedig egy erősen dizájn szemléletű, szintén nagyon hatékony fejlesztőeszköz. Könyvünk példáihoz a Flash Builder-t fogjuk preferálni. (Mindkét eszköz fizetős, liszenszköteles szoftver, de lehetőség van néhány napos próbára is.)

A Flash Buildernek jelenleg a *Flash Builder 4.7 premium* verziója kapható.

Könyvünkben a Flash Builder 64 bites, 4.7 Standard verziójával fogunk dolgozni, ami ugyan egy korábbi verzió, de teljesen kompatibilis a Premiummal.

A Flash Builder-rel robusztus, nagy teljesítményű üzleti alkalmazásokat, professzionális weblapokat és játékokat is fejleszthetünk, több más programozási nyelvet és szerveroldali technológiák széles palettáját felhasználhatjuk a fejlesztéseinkhez, köztük például az elterjedt és közkedvelt PHP technológiát is.

3.4.3. A Flash Builder telepítése

A Flash Builder rendszerkövetelményei:

Windows

2GHz-es vagy gyorsabb processzor
Microsoft® Windows® XP Service Pack 3, Windows 7 (32 bit vagy 64 bit), Windows 8, Windows 10
2GB RAM
5GB HDD
Java™ Virtual Machine (32 bit): Oracle® JRE 1.6 vagy 1.7
Java Virtual Machine (64 bit): Oracle JRE 1.6 vagy 1.7
1024x768-as felbontású kijelző (1280x800 ajánlott)
16-bites videókártya
Eclipse™ 3.7 vagy Eclipse 4.2 (a plug-inként történő telepétshez)

Mac

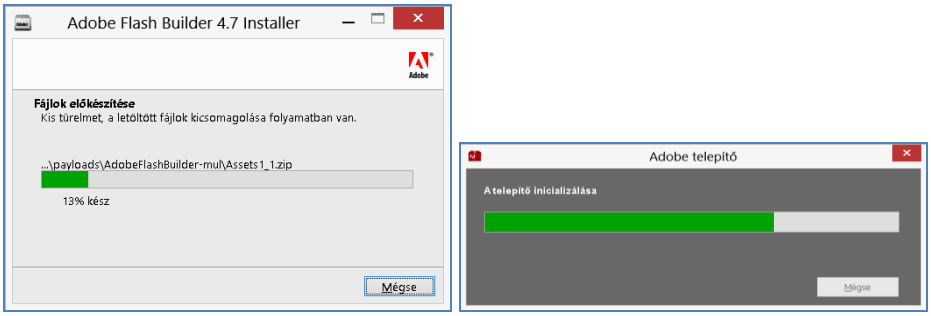
Intel® processor
Mac OS X v10.6, v10.7, vagy v10.8
2GB RAM
4GB HDD
Java Virtual Machine (64 bit): JRE 1.6
1024x768-as felbontású kijelző (1280x800 ajánlott)
16-bites videókártya
Eclipse 3.7 vagy Eclipse 4.2 Cocoa version (a plug-inként történő telepítéshez)

A Flash Builderhez az Adobe honlapjáról juthatunk hozzá (www.adobe.com). A letöltés feltétele (a megvásárlás után), hogy létrehozzunk egy Adobe Creative Cloud fiókot (ingyenes).

A Flash Builder kétféleképpen telepíthető: önálló alkalmazásként, vagy Eclipse plug-inként. Utóbbi esetben a következő Eclipse disztribúciók támogatottak: *Eclipse IDE for Java EE developers, Eclipse IDE for Java developers, Eclipse Classic, Eclipse for PHP developers*.

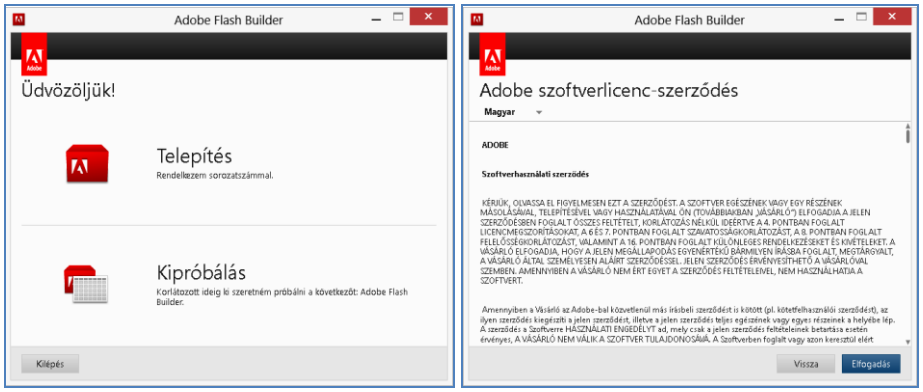
Az önálló alkalmazásként történő telepítés mindent tartalmaz és gyakorlatilag néhány kattintásból áll. A telepítő 32 és 64 bites változatokban is elérhető, Windows és MAC OS operációs rendszerekre is. A könyvben a 64 bites Windows változatot fogjuk használni, önálló alkalmazásként telepítve. A telepítő magyar nyelven is tud kommunikálni.

A telepítés a szükséges fájlok automatikus kicsomagolásával kezdődik.



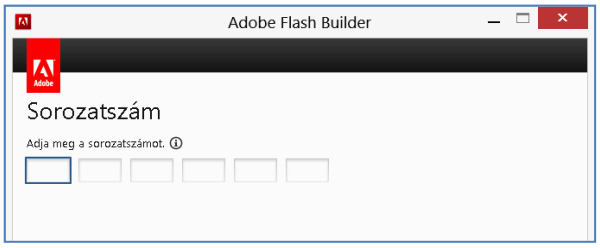
Következő lépésként a telepítő inicializálására kerül sor.

Ezt követően választhatunk, hogy csak kipróbáljuk-e a terméket néhány nap erejéig, vagy végleges (ez utóbbi telepítési módot mutatja be a könyv is) használatra telepítjük. Ez utóbbi esetben elő kell készítenünk a megvásárolt termékhez kapott sorozatszámot is.

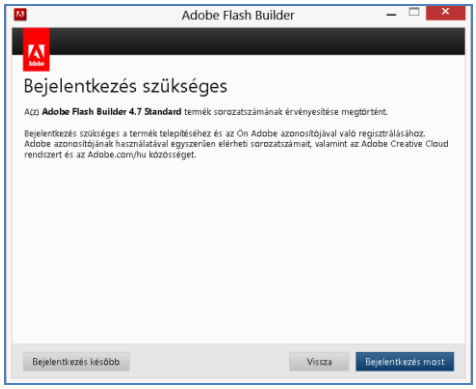


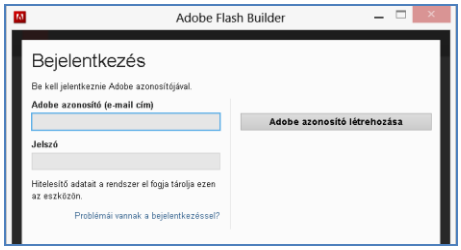
Következő lépésként el kell fogadnunk az Adobe szoftverliszensz-szerződését.

Ezt követően meg kell adnunk a termék egyedi sorozatszámát. Mielőtt a 'Tovább' gombra kattintanánk, ellenőrizzük, hogy aktív internet-kapcsolattal rendelkezünk-e, mert a telepítő interneten keresztül aktiválni fogja Flash Builder példányunkat! A tényleges telepítés addig nem kezdődik el, amíg ez nem történik meg.



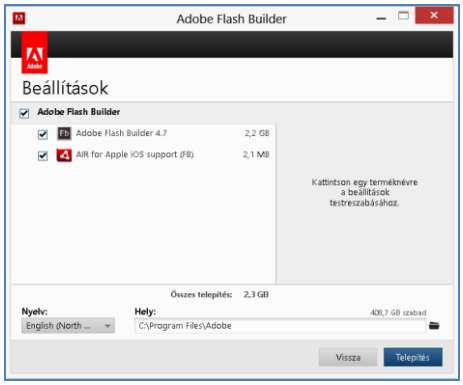
Ezt követően a telepítő felhívja a figyelmet arra, hogy utolsó lépésként meg kell adnunk az Adobe Creative Cloud fiókunk adatait.





A bejelentkezés gyors, egyszeri és nem igényel további felhasználói interakciót a későbbiekben.

A telepítés következő lépésének beállításait nyugodtan hagyhatjuk az alapértékeken.



A telepítés folyamata kb. 15-20 percnél nem tart tovább.

Immár rendelkezésünkre áll egy komplett keresztplatformos fejlesztőrendszer! A Flash Buildert az alábbi ikonnal indíthatjuk el (itt is érdemes rendszergazdai jogosultsággal indítani).

3.4.4. A Flash Builder kezelőfelülete

A Flash Builder az alábbi képernyővel indul:

Az integrált fejlesztői környezet (IDE) fő részei a következők:

Menüsor és ikonsor (1.) a leggyakoribb műveletek gyors eléréséhez.

Nézetek (2.). A Flash Builder két legfontosabb nézete a Flash és a Flash Debug, mi az előbbivel foglalkozunk majd. Egy nézet nem más, mint a szerkesztőablakok elrendezése. Ezeket az ablakokat a Flash Builder „**View”**-knak, a nézeteket pedig „P**erspective”**-nek nevezi. A nézetek ablakainak méretét és elrendezését tetszőlegesen módosíthatjuk, akár több ablakot is megnyithatunk, ilyenkor átlapolt elrendezésben jelennek meg az ablakok és fülekkel válthatunk közöttük. Ha egy View fülére duplán kattintunk, akkor a Flash Builder ikonsor alatti részét az adott View fogja teljesen kitölteni. Ha még egyszer duplán kattintunk a fülön, visszaáll az előző méret.

Szerkesztőterület (3.) („**editor”**-ok): Ebben az ablakban szerkeszthetjük alkalmazásunk forráskódjait és itt jelenik meg a kódellenőrzés, a fordítás, illetve a Debug futtatás kimenete is. Szintén itt található az alkalmazás indításakor automatikusan betöltődő 'Welcome Screen' is,

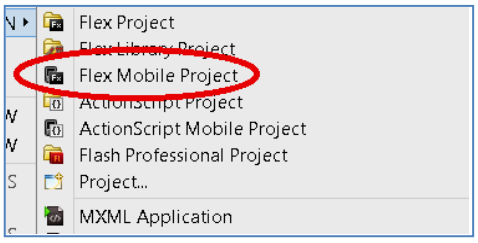
ahonnan közvetlenül is létrehozhatunk, vagy megnyithatunk projekteket és számos online információforráshoz is hozzáférhetünk.

Package Explore (4.): ebben az ablakban projektjeink jelennek meg. Egy projekttérhez („**Workspace”**) több projekt is tartozhat. A workspace tulajdonképpen egy komplex mappastruktúra, melyben a projektjeink almappákba rendezve vannak tárolva. Egy új projekt tehát a workspace egy almappájába lesz elmentve és ez jelenik majd meg a Package Explorer-ben is. A Package Explorer-ben lényegében ezeknek a mappáknak a tartalmát láthatjuk, egy hierarchikus struktúrába rendezve. Tetszőleges Workspace-t létrehozhatunk, de egyszerre mindig csak egy lehet megnyitva. A Flash Builder alapértelmezés szerint a Windows aktuális felhasználójának dokumentum mappái között hoz létre egy workspace mappát és ennek egy-egy almappájában a fejlesztett alkalmazás Debug verzióját és a telepítőkészletét. A workspace-ek között a 'File' menü 'Switch Workspace' menüpontjával tudunk váltani, ez a Flash Builder újraindítását igényelheti. Ha egy Workspace-en belül egy projekttel nem dolgozunk aktívan, erőforráskímélés végett bezárhatjuk azt. Ehhez kattintsunk a jobb egérgombbal a Package Explorerben a projekt nevére és a helyi menüből válasszuk ki a 'Close project' menüpontot. A projekt ikonja bezárt állapotban kiszürkített. Bezárt projektet úgy tudunk leggyorsabban újra megnyitni, hogy duplán kattintunk a nevére. A workspace könyvtárak gyökerében létrejön egy '.metadata' nevű könyvtárt is, mely a Workspace-ről tárol különféle leíró adatokat és a Flash Builder kezeli. Soha ne töröljük, vagy módosítsuk kézzel ezt a könyvtárat!

Az **Outline (5.)** nevű ablak az aktuálisan szerkesztett forrásfájl objektumait jeleníti meg. Itt most csak a rend kedvéért említjük meg, a könyvben a későbbiekben nem fogjuk használni.

3.4.5. Mobilprojekt létrehozása

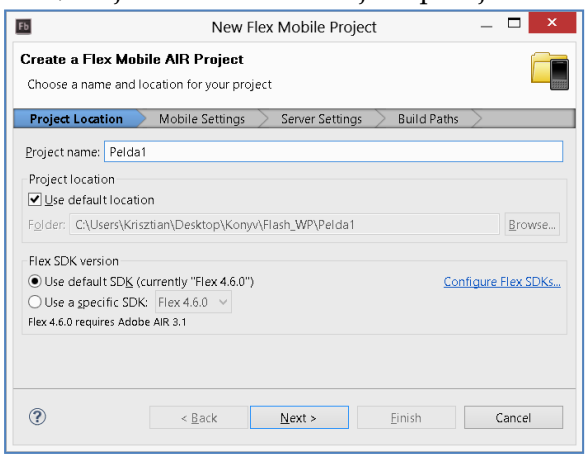
Könyvünkben a Flash Builder-t androidos alkalmazások készítésére is használni fogjuk. Ez egy külön projekt típus ugyan, de megjegyezzük, hogy az asztali alkalmazások és a böngészős alkalmazások előállítása is nagyon hasonló az alább bemutatott lépésekhez.



A Flash Builder alapvető funkcióinak mélyebb megismerése érdekében hozzunk létre egy mobilprojektet!

Kattintsunk a File menü New, majd 'Flex Mobile Project' pontjára!

A megjelenő varázsló párbeszédablakában több lépésben be kell állítanunk új projektünk alapvető tulajdonságait. A 'Project Location' lapon először is nevet kell adnunk projektünknek, illetve kiválaszthatunk egy tetszőleges Flex SDK-t. A példaprojektnek a 'Pelda1' nevet adtuk.



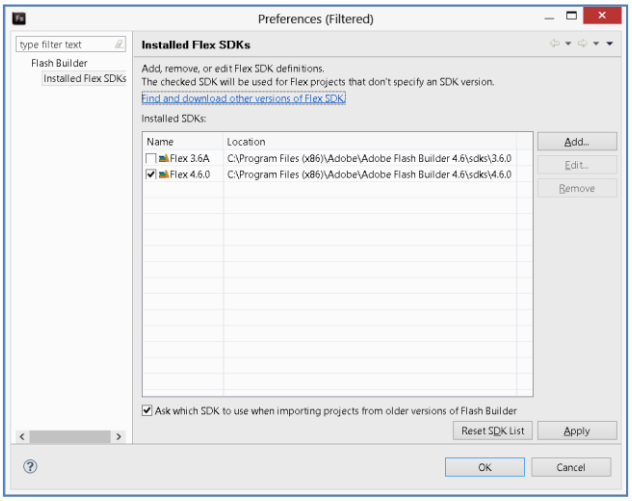
3.4.5.1. Pár szó a Flex SDK-ról

A Flex SDK (Software Development Kit, azaz: szoftverfejlesztő eszközkészlet) tulajdonképpen nem más, mint egy könyvtár a számítógépen, mely számos, a fejlesztéshez használható komponenst és segédprogramot tartalmaz.
Az aktuális SDK-t az Apache Foundation weboldaláról tölthetjük le:
http://flex.apache.org

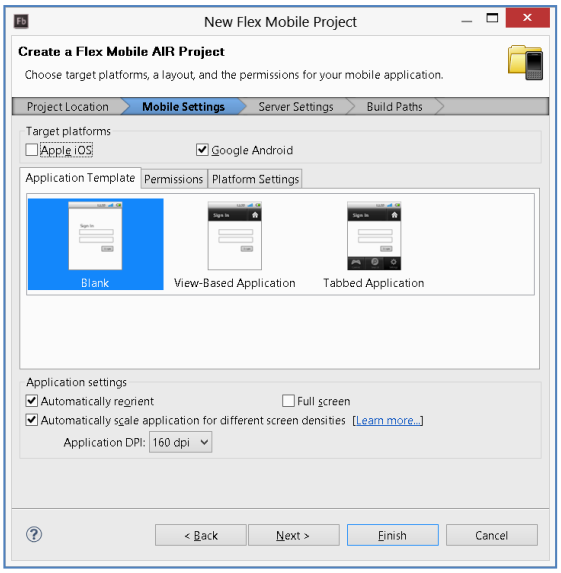
Szintén innen letölthetjük az SDK Manager nevű segédprogramot, aminek a segítségével igényeinkhez szabottan, egy lépésben letölthetjük az elérhető SDK-kat. Az SDK beszerzéséről és használatba vételéről később még részletesen szólunk.

Jó hír, hogy a Flash Builder alapértelmezetten telepíti a 3.6-os és 4.6-os Flex SDK-kat is, amik nem túl frissek, de indulásnak, otthoni tanulásra megfelelőek és éles projekthez pedig majd később letölthetjük a legfrissebb SDK-t.

Az imént ismertetett párbeszédablakban a 'Configure Flex



SDKs' hivatkozásra kattintva adhatunk meg további SDK elérési útvonalakat a rendszerünkben. A könyv írásakor legfrissebb SDK a 4.16.

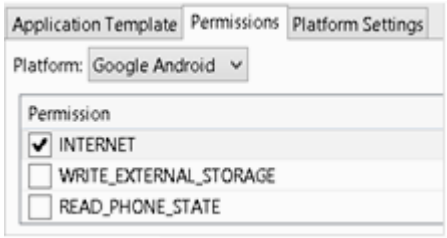


Térjünk vissza a 'New Flex Mobile Project' párbeszédablakhoz és kattintsunk a 'Next' gombra!

A „Mobile settings” lapon a célplatform alapvető tulajdonságait adhatjuk meg projektünk számára. Példánkban egyedül a 'Google Android'-ot válasszuk ki!

Az 'Application Template' fülön válasszuk a 'Blank' sablont, amivel egy teljesen üres felületen, „nulláról indulva” építhetünk fel alkalmazásfelületeket. (A másik két lehetőséggel, a View-alapú és a Tab-alapú alkalmazásfelületekkel könyvünk nem foglalkozik.)

A 'Permissions' fülre kattintva megadhatjuk, hogy alkalmazásunk milyen jogosultságokat igényel, azaz az adott mobileszköz mely szolgáltatásaira tart igényt.



A jogosultságokat az Android nagyon komolyan veszi. Amennyiben a megfelelő jogosultság nincs beállítva, úgy a kész alkalmazás akkor sem fog tudni hozzáférni az adott szolgáltatáshoz, ha maga a forráskód hibátlan!

Az egyes jogosultságok jelentése:

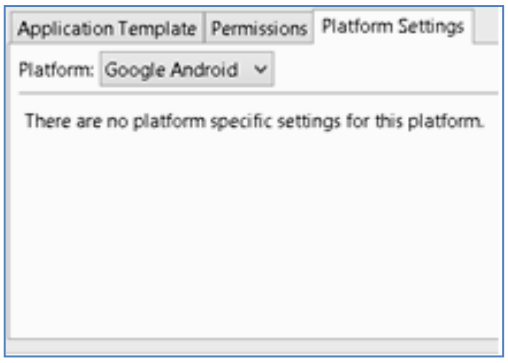
INTERNET: internet-hozzáférés engedélyezése. (AIR alkalmazások esetében kötelező beállítani.)

WRITE_EXTERNAL_STORAGE: hozzáférés a készülékek írható memóriájához, mint fájlrendszerhez.

READ_PHONE_STATE: a telefon állapotának lekérdezése.

ACCESS_FINE_LOCATION: hozzáférés a GPS helymeghatározás által szolgáltatott adatokhoz.

DISABLE_KEYGUARD, WAKE_LOCK: a készülék zárolt állapotának kezelése.
CAMERA: hozzáférés a beépített kamerfunkcióhoz.
RECORD_AUDIO: hangfelvételi szolgáltatás igénybevételéhez.
ACCESS_NETWORK_STATE,
ACCESS_WIFI_STATE: a hálózati kapcsolat állapotának a lekérdezéséhez.



Megjegyezzük, hogy az alkalmazásleíró fájl (lásd későbbiekben) számos más hozzáférési beállítást is lehetővé tesz, sőt, kézzel is beírhatjuk az adott jogosultság nevét. Ez azért fontos, mert a Google időnként plusz jogosultságok hozzáadását követeli meg az alkalmazásoktól. Erre tehát gondolnunk kell.

A 'Platform Settings' fülön nincsenek további beállítási lehetőségek Android esetében.

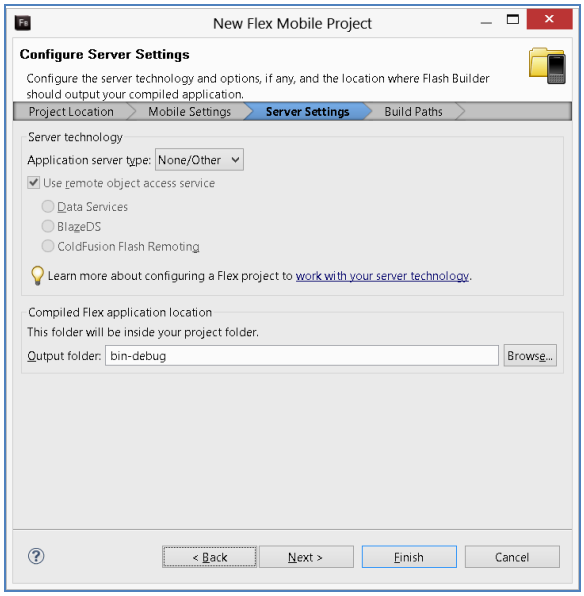
A párbeszédablak alsó részén, az „Application Settings” alatt további beállításokat eszközölhetünk.

'Automatically reorient': bekapcsolva az alkalmazás külön kód írása nélkül is automatikusan megpróbálja áthelyezni a GUI elemeit, ha elforgatjuk a mobilkészüléket, amelyen fut.

'Automatically scale application for different screen densities': bekapcsolva a különböző DPI-vel rendelkező kijelzőknél a megjelenítés automatikusan idomul a készülék aktuális DPI-jéhez. Kiválaszthatjuk a viszonyítási DPI értéket is. Az alapérték 160.

'Full screen': kiválasztva a mobileszköz képernyőjének alsó, vagy felső részén található szoftveres 'Home', 'Back' stb. gombok is takarásba kerülnek alkalmazásunk felületének javára. Csak nyomós okból érdemes beállítani (pl. teljes képernyős játékok esetében).

A 'Next' gombra kattintva a 'Server settings' lapra jutunk és szervertechnológiák közül választhatunk. Könyvünk péáldáihoz ezekre a beállításokra nem lesz szükségünk. Első példaalkalmazásunk esetében tehát nyugodtan hagyhatjuk a szervertípust 'None/Other' értéken.



A 'Next' gombra kattintva az utolsó, 'Build Path' lap jelenik meg, ahol fordítással kapcsolatos beállításokat végezhetünk el.

Egyetlen érték változtatását érdemes alapesetben végrehajtanunk, ez pedig az Application ID megadása. Ezzel egy egyedi elnevezést biztosíthatunk alkalmazásunk számára. Az alapérték megegyezik projektünk nevével, mi ezt módosítottuk a 'FeherKrisztian.Pelda1' értékre. Az ID egyes szavait pontokkal kell elválasztanunk egymástól, akárcsak egy domain nevet.