Debreceni Egyetem

Informatikai Kar

Biológiai oktatóprogram középiskolásoknak – Az emberi test

Témavezető:

Dr. Rutkovszky Edéné

egyetemi tanársegéd

Készítette:

Bécsi Zoltán

informatika tanári szak

Debrecen

2009

Tartalomjegyzék

1. Bevezetés

Az elmúlt száz évre visszatekintve megfigyelhetjük, hogy az emberi civilizáció hatalmas, és rohamos lépésekkel fejlődött. Ez a fejlődés nem állt meg, hanem még gyorsabb ütemben halad tovább, néha már teljesen követhetetlenül.

Olyan találmányok jelentek meg, melyek megváltoztatták a mindennapjainkat, némelyikük már teljesen a beépültek az életünkbe, természetessé váltak.

A számítástechnika fejlődését végigkísérve láthatjuk milyen hatalmas lépcsőfokokat ugrott. Kb. hatvan évvel ezelőtt a gépek még nagy termetűek, lassúak és megbízhatatlanok voltak, azaz hamar és könnyen meghibásodtak. A technika fejlődésével ezek a problémák megszűntek. A félvezetők feltalálásával és alkalmazásával, valamint a mikro gyártási technológiák tökéletesedésével a számítógépek mérete jelentősen csökkent, és a művelet végző sebességük jelentősen növekedett.

A miniatürizálódás egyben globalizálódást is jelentett az emberiség számára. Ez pedig nem más, mint az internet. Olyan lehetőségeket nyit meg, melyek nagyban megkönnyítik a feladatok elvégzését, ugyanakkor egy információs dzsungel, amelyben nagyon könnyen el lehet tévedni.

A mai oktatásban egyre több és komplex tudásanyagot kell átadni a következő kor generációjának, melyek megértetéséhez már segítségül kell hívni az új kor vívmányait és a tanítás szolgálatába kell állítani.

Dolgozatom az informatika adta lehetőségek közül mutat be párat, valamint egy DVD-ROM-ot, mely a biológia tanításában nyújt segítséget.

Egyre többet lehet hallani ma az oktatóprogramokról, melyek a multimédia és hipermédia eszközeit alkalmazva teszik szemléletesebbé tananyagot. Nemcsak formailag, hanem tartalmilag is többet nyújtanak az oktatásban felhasznált hagyományos eszközökkel szemben.

Azért döntöttem e téma mellett, mert én is szerettem volna egy eszközzel segíteni a biológiaoktatást. Igyekeztem igazodni a középiskolában tanítandó tananyaghoz, azon belül is az ember témakörhöz.

Az utóbbi években egyre feltűnőbb jelenség a középiskolások órai inaktivitása, érdektelensége. Véleményem szerint ehhez köze van a televíziónak, számítógépnek, és nem utolsó sorban a társadalomnak. Sajnos ezek a gyerekek sokat ülnek a fent említett eszközök

előtt, és esnek el a családi neveléstől. Az utóbbi években egyre több szülő vállal a főállása mellett további munkákat a család megélhetése érdekében, aminek eredménye az, hogy nincs ideje a gyerekkel foglalkozni. Két munkahely után örül, ha tud piheni, és ilyenkor a gyerek a tv-vel, számítógéppel, vagy egyéb módon köti le magát, nem is érzi a szülői társaság igényét. Mondhatni, hogy a tv és az internet neveli fel őket. Ők megszokják ezen eszközök színes, villogó világát olyannyira, hogy egyszerű tanítási órán „elalszik”, mert kevés inger éri („monoton tanári beszéd”, „négy-öt színnel rajzolt táblai rajz”).

A tanároknak fel kell venniük a versenyt az előbb említett a tényezőkkel, és át kell alakítaniuk az oktatói munkát úgy, hogy ezek a gyerekek újra érdeklődve, aktívan vegyenek részt az órákon.

Szerintem az lehet az egyik megoldás, ha nem eltiltjuk ezektől az eszközöktől őket, hanem beépítjük azokat a tanulási tevékenységbe. A számítógép előtt eltöltött idő egy része tanulásra is fordítható, ha van olyan termék, amellyel ezt megteheti.

Több különböző szakos pedagógussal is beszéltem erről a problémáról, akik szintén hasonló véleményen vannak: érzik, hogy valamire szükségük lenne, amivel érdekesebbé, figyelemfelkeltővé lehet tenni a tárgyaikat.

Már 2004 októbere óta dolgozok a tanári pályán, biológiát egy éve tanítok szakiskolás diákok számára. A fent leírtak rájuk a tapasztalatom szerint kiemelten igaz. Ennek az egy éves időszaknak a végére fogalmazódott meg bennem, hogy mennyire eredményesebb lenne a biológia tanítása, ha lenne a kezemben egy olyan multimédiás oktatóprogram, amellyel felkelteném a tanulók érdeklődését, ráadásul úgy, hogy a szabadidős tevékenységben használt „kedvenc” eszközüket használom fel.

Azért az embert választottam az oktatóprogram témájának, mert fontosnak érzem, hogy egy alap szinten mindenki legyen tisztában a saját testével, hogy mivel rombolhatja, óvhatja vagy építheti azt. A célom egy olyan oktatóprogram létrehozása, amely közérthető nyelven, szemléletesen mutatja be az embert. Természetesen olyan plusz információkat is tartalmaz, amelyek továbbtanulási felkészüléshez is alapul szolgálhat.

A program elkészítéséhez a C# programozási nyelvet választottam, mert nagyon jól használható fejlesztésre, és minden platformon megoldható a program futtatása.

2. Irodalmi áttekintés

2.1 A tanulást befolyásoló tényezők

Mielőtt rátérnék a program bemutatására összefoglalnám, hogy mik azok a tényezők, amelyek hatással vannak az alkalmazással megcélzott korosztály tanulási szokásaira és viselkedési mintáira. Az ismeretszerzést befolyásolják az adott életkorra jellemző tulajdonságok, azaz, hogy az egyén problémamegoldó, logikai gondolkodása, milyen fejlettségi szintet ér már el, és az én-tudat mennyire alakult már ki.
Életkori sajátosságok pszichológiai értelemben:

12 – 15 éves korig:

- Pszichoszociális fejlődés szakasza: identitás a szerepbizonytalansággal szemben.
- Kognitív fejlődés szakasza: megkezdődik az áttérés a formális gondolkodásra.
- Morális fejlődés szakasza: konvencionális szint, megkezdődik az áttérés a kényszererkölcsről az együttműködés erkölcsére.

Testi jellemzők:

- A lányok növekedése megáll, a fiúké tovább folytatódik.
- Nemi érés.
- Bizonyos méretű esetlenség.
- Nem esznek rendesen, nem alszanak eleget.

Társas jellemzők:

- A kortárs csoport az irányadó.
- Tetőpontjára ér a többiekhez való igazodás.
- Érdekli őket, hogy a társaiknak milyen véleményük van róluk.
- Érzelmi jellemzők.
- Viharzás korszaka.
- Megugrik a bűncselekmények száma.

Kognitív jellemzők:

- Áttérés a konkrét műveletekről a formális műveletekre.

- Kényszererkölcsről az együttműködés erkölcsére.
- Megfontoltakká válnak bizonyos, például politikai kérdésekben

16 – 18 éves korig:

- Pszichoszociális fejlődés szakasza: identitás a szerepbizonytalansággal szemben.
- Kognitív fejlődés szakasza: formális gondolkodás.
- Erkölcsi fejlődés szakasza: konvencionális szint, az együttműködés erkölcse.

Testi jellemzők:

- A diákok éretté válnak.
- Sok bizonytalanságot él át a szexuális kapcsolatok terén.
- Növekvő szexuális kapcsolatokkal nő a fiatalkori szülések és a nemi betegségek száma.

Társas jellemzők:

- Az aktuális cselekedetekben a kortársak, a hosszú távú tervekben a szülők véleménye fontos.
- A lányokat sokkal jobban foglalkoztatja a barátság, mint a fiúkat.

Érzelmi jellemzők:

- A lányok a gimnáziumi évek vége felé érzelmileg lehangoltabbak, mint a fiúk.
- Depresszió.

Súlyosbodó depresszió esetén az öngyilkosság gondolata:

- Gyerekkori beilleszkedési problémák.
- Előtte úgy érzi, összecsapnak felette a hullámok.
- Arra a következtetésre jut, hogy képtelen megbirkózni az őt körülvevő problémákkal.
- A szülőkkel való érzelmi kapcsolata kiüresedik, úgy érzi, hogy senki sem érti meg őt.

Kognitív jellemzők:

- Már képesek formálisan gondolkodni, de nem mindig használják ezt a képességüket.
- Akik a formális gondolkodásra képesek, féktelen elméleteket alkothatnak.

A gondolkodást nemcsak az életkori sajátosságok szabják meg, hanem a személyiség is, ami meghatározza a tudatot, az egyén viselkedését. Erre Freud dolgozott ki egy átlátható elméletet, mely szerint három összetevőből épül fel egy egyén személyisége.

Sigmund Freud személyiség struktúrája:

1. **Id (ösztön-én):**

Az örömelv alapján működik, célja az örömszerzés és a fájdalom elkerülése. Belső illetve külső eredetű feszültségből keletkezi.

- Reflexcselekvés: hunyorgás
- Elsődleges folyamatok: vágyteljesítő lelki képmások

2. **Ego (én):**

A realitás elven működik, igyekszik rátalálni arra a tárgyra, ami a belső feszültség leküzdéséhez szükséges. Másodlagos folyamatok irányítását szabja meg.

3. **Szuperego (felettes én):**

A társadalom normáinak, eszmevilágának belső képviselője.

- Lelkiismeret: mit szabad tenni és mit nem
- Én-ideál: amivé az egyén válni szeretne

Védekező mechanizmusok:

- Elfojtás: nem tudatossá tesz tudott dolgokat
- Projekció: szorongását másra vetíti ki
- Reakcióképződés: az indulati feszültség szabályozásának az a formája, amelyben a személy a feszültséget okozó cselekvéssel, szokással szemben merőben ellentétes irányú viselkedésmódot alakít ki.
- Fixáció: megrekedés a fejlődés egy adott szakaszán
- Regresszió: visszalépés egy fejlettebb fejlődési szintről

A tanulást más tényezők is befolyásolják. Egyik ilyen fontos tényező a motiváció. Lehet belső és külső egyaránt. Külső motiváció esetén valami, vagy valaki arra ösztönzi az egyént, hogy egy adott téma irányába fordítsa figyelmét. A tanulás eredményességét nagymértékben meghatározza, hogy az adott ismeret egyáltalán érdekli-e. Külső ösztönzés esetén, egy az érintett személyen kívül más kelti fel az érdeklődést.

Belső motiváció esetén az érintett személy saját magától kezd elérdeklődni, a téma iránt nem szükséges azt felkelteni. Ez esetben a tanulás sokkal eredményesebb, és rövidebb idő alatt valósul meg. Ez esetben ún. önképzés is megfigyelhető.

A sikerélmény is motivációs tényezővé válhat, főleg abban az esetben, ha a tanulónak az érintett kérdésben nem sikerült pozitív eredményt elérnie.

A bármilyen formában is történik az ismeretszerzés, fontos hogy szerepeljen benne a munka szeretetére, a humanizmusra, a hazafiasságra való nevelés.

A biológiára vonatkoztatva fel kell ismertetni a természet, az emberi test csodálatosságát. Lássák be az emberi test védelmének a szükségességét, fontosságát. Ha ellentétesen cselekednek, akkor a saját életvitelüket, életminőségüket rossz irányba sodorhatják.

2.2 E-leraning

Az e-learning a hagyományos és távoktatási, valamint az internet nyújtotta új lehetőségek együttes alkalmazását jelentő új és hatékony tanulási eljárás. Még gyermekcipőben jár és gyermekbetegségekkel küzd, de már ma is több mint kísérleti oktatási módszer. Előnye a rugalmasság, elérhetőség, kényelem, saját időbeosztás szerinti előrehaladás a tananyagban. Az e-learning fejlődését az egyre kifinomultabb internetes technológiák megjelenése tette lehetővé.

A távoktatást történelmileg a XVIII. századig vezethetjük vissza, amikor is az Egyesült Államokban megindult a levelező oktatás. A XIX. szászad közepén Európában is terjedni kezd a levelezős oktatás.

Az 1960-as évek végén, az 1970-es évek elején az új médiatechnológiák fejlődésének köszönhetően jelentős változások érték a távoktatást. A Nyitott Egyetem volt az első Nagy-Brittanában, amely távoktatás keretében kínált főiskolai diplomákat.

A World Wide Web és gyorsan fejlődő alkalmazásai terjedésének köszönhetően az infokommunikációs technológiák (ICT) által támogatott oktatás elég gyorsan felkapott téma lett az 1990-es években. Az új technológiák új lehetőségeket hoztak mind a nem hagyományos tanulóknak, mind a hagyományos oktatási intézményeknek.

Az e-learning gyűjtőfogalom, magába foglalja a tanulás bármely típusát, amely a legújabb információs és kommunikációs technikákat online alkalmazza.

Az e-learning előnyei:

- egyéni ütemezésű tanulási folyamat, rugalmasság, hozzáférhetőség, kényelem;
- utazási költség-, idő- és egyéb megtakarítások;
- nagymértékben testre szabható, egyéni tanulási stílus;
- interaktív és multimédiában gazdag tanulási tartalmak;
- tanulóközpontú tanulás, aktívabb részvétel a tanulási folyamatban;
- könnyebb tartalomkezelés, egyszerűbb adatkezelés, a frissítés egyszerűsége;
- a tartalom összekapcsolásának lehetősége más tanulási forrásokkal;
- osztott könyvtárak használata, olcsó világméretű szolgáltatás;
- integrált értékelés és tesztelési lehetőségek;
- a tanulás eredményességét mérő módszerek változatossága;
- a tanulás előrehaladásának ellenőrzése és értékelése a „befektetések várható megtérülése” (Return on Investment = ROI) mérésével;

Az e-learning legegyszerűbben megközelítve olyan távoktatás, amely elektronikus úton valósul meg. A távoktatás térhódításának számos elemzés szerint az oktatási szektorban nem a közoktatás, hanem a felsőfokú, illetőleg a posztgraduális képzés lehet a legfőbb színtere. A posztgraduális képzésben érintett hallgatók szűkebb szakterületek irányába igyekeznek orientálódni, ugyanakkor az ország különféle szögleteiben élnek. A távoktatás nagyszerű média egy potenciálisan növekvő jövedelmű hallgatóság elérésére, ráadásul oly módon, hogy az számukra is kényelmet nyújt.

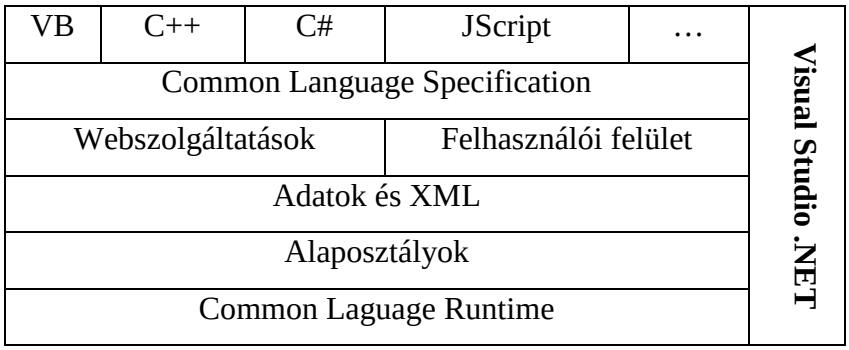
Hogyan kapcsolódik mindez az oktatóprogramokhoz? Igaz hogy nem online tananyag illetve segédanyag, de elektronikus formában létezik, és ugyan úgy megjelennek az e-learning előnyök nagy része. Leginkább az ismeretszerzése vonatkozók.

Az oktatóprogramok nagy előnye, hogy általában CD-ROM-on, vagy ma már annyira nem is ritkán DVD-ROM-on vannak tárolva, tehát offline – internetkapcsolat nélkül – alkalmazhatók, tanulhatók.

2.3 A .NET Framework

Az oktatóprogram és minden más C#-ban írt alkalmazás működéséhez elengedhetetlen, hogy az adott gépen telepítve legyen a fejezet címében megnevezett keretrendszer. Ez hasonló szerepet tölt be, mint a Java esetében a Java Virtual Machine (Java Virtuális Gép). Hogy mi is ez? A következő pár oldalon röviden mutatom be.

A .NET különböző méretű és jellegű szoftverek fejlesztéséhez alkalmazható előre gyártott infrastruktúra. Új szemléletet vezet be a fejlesztéshez. Több puszta specifikációnál, része egy termékhalmaz is, amely arra szolgál, hogy a mobil eszközöktől az asztali számítógépeket át a legnagyobb rendszerekig egységesen, konzisztens módon és gyorsan lehessen szoftvert fejleszteni, beleértve a webes és a nem webes megoldásokat is.



A .NET Framework összetétele

A .NET Framework legfontosabb eleme a közös futtató környezet, a CLR (Common Language Runtime). Ez az alrendszer gondoskodik a program végrehajtásáról, továbbá biztosítja az alapvető futtatási szolgáltatásokat. Ez CLR a Javaban meglévő JVM (Java Virtual Machine) szintjén helyezkedik el, és hasonló szerepet tölt be. A megírt programok esetében mind a kód mind az adat a CLR felügyelete alatt fut. Ezzel két célt lehet elérni: az egyik a biztonságos futás, a másik pedig az, hogy a futást egy virtuális végrehajtó rész végzi, ami biztosítja a több platformon való működés lehetőségét. Ennek a felügyelt futásnak hátránya is van: a futási sebesség csökken. Ha a sebesség az egyik a legfontosabb szempont, akkor a .NET felkínálja a vegyes futtatást, azaz felügyelt kódból belehívhatunk nem felügyelt kódba, elveszítve a közös futtatókörnyezet által biztosított előnyöket.

A CLR a CLI (Common Language Infrastructure) szabványos Windows operációs rendszer alatti implementációja. Minden CLR-kompatibils fejlesztőeszköz a forráskódot a

szabványos Microsoft IL (Intermediate Language) nyelvre fordítja le. Mivel minden fejlesztőeszköz az IL-re fordít, a megvalósítási különbségek a forráskód nyelvétől függetlenül eltűnnek. A fejlesztőeszközök által létrehozott IL-kód közvetlenül nem futtatható egyetlen számítógépen sem, szükség van az úgynevezett just-in-time (JIT-futásidejű) fordításra. A JIT-fordító beolvassa az IL-t, és az adott rendszeren futtatható gépi kódú programot készít belőle. Ezzel a .NET bizonyos mértékig rendszerfüggetlenné válik, mert minden rendszerre lehet írni külön JIT-fordítót.

<table>
<tr><td colspan="3">Osztálykönyvtár-kiszolgálás</td></tr>
<tr><td colspan="2">Szálkezelő</td><td>COM Marshaler</td></tr>
<tr><td colspan="2">Típusellenőrző</td><td>Kivételkezelő</td></tr>
<tr><td colspan="2">Biztonsági motor</td><td>Hibakereső motor</td></tr>
<tr><td>IL→ Natív JIT-fordítók</td><td>Kódkezelő</td><td>Szemétgyűjtő</td></tr>
<tr><td colspan="3">Osztálybetöltő</td></tr>
</table>

A Common Language Runtime felépítése

A COM (Component Object Model) 1992-ben született, mint 16-bites Windows-alapú rendszerek komponenstechnológiája. A mai napig jól használható, megoldotta a DLL okozta olyan problémákat, mint a fordító- és nyelvfüggetlenség, verziókezelés stb. Könnyen belátható, hogy a COM előtt az alkalmazások el voltak szigetelve egymástól. Egy dolgot azért fontos megjegyezni: a COM csak a kommunikáció lehetőségét teremti meg, a közös nevezőt nem. Ezt az adott objektumoknak kell megoldaniuk. A .NET Framework és a CLR segítségével a komponensek közös alapra kerülnek, így a közvetlen kommunikáció is lehetséges.

A COM Marshaler biztosítja, hogy a már fejlesztett COM-komponensek továbbra is használhatók legyenek a .NET alkalmazásokban. Lényegében a .NET támogatja az együttműködést a COM-mal, akár olyan .NET ügyfélként, amely COM kiszolgálót használ, akár fordítva.

A kivételkezelő (Exception Manager) kezeli a futási időben bekövetkező hibákat. Az egyes nyelvek eltérő módon kezelik ezeket a hibákat. A kivételkezelő egységes strukturált kivételkezelési mechanizmust biztosít a .NET-alkalmazásokhoz. Ha egy program megírása közben arra számítunk, hogy egy adott művelet esetleg nem sikerül, akkor adott műveletet egy kivételkezelő blokkba helyezzük el, ami az futáskor felmerülő kivételt elkapja és lekezeli.

A szaknyelvzsargon a kivételkezelőt a kódban használatos szakszavakkal illeti: Try-Catch blokk.

A biztonsági motor (Security Engine) biztonsági ellenőrzéseket hajt végre a kóderedethez kötött jogosultságok tekintetében. A kódot is azonosítja, nemcsak a felhasználót, megadja, hogy mit művelhet egy adott kód egy adott számítógépen. Véd a megbízhatatlan kódok és a rosszindulatú felhasználók ellen egyaránt.

A típusellenőrző (Type Checker) futásidejű típusellenőrzést biztosít. A komponensek nem bináris kompatibilitáson (COM), hanem típus-kompatibilitáson alapulnak, így lehetőség nyílik szigorú típusellenőrzésre, ami biztosítja az alkalmazások biztonságos futását. Platform- és nyelvfüggetlen megoldása lehetőséget ad tetszőleges nyelvek integrációjára, illetve alkalmazások, komponensek más platformokra történő futtatására. A típusellenőrző futási időben, a kötelező metaadat alapján figyeli az egyes műveletek típushelyességét. Ez az a komponens, amely a .NET Frameworkben mindenhol megjelenő erős típusságot futási időben is garantálja. Az erős típusosság azt jelenti, hogy minden művelet csak meghatározott hozzárendelt adattípusokkal végezhető. Ha egy adott műveletet olyan adaton próbálunk meg végrehajtani, amelyiknek a típusa nem felel meg az előírtaknak, akkor a CLR figyeli, és egy kivételt kapunk, ami arra utal, hogy itt valami meg nem engedett művelet akart megtörténni.

A szálkezelő (Thread Support) támogatja a többszálú futtatást, felügyeli annak helyes végrehajtását és biztosítja a szinkronizációt.

A szemétgyűjtő (Carbage Collector) végzi a nem használt objektumok takarítását, nélküle a felügyelt kód elvesztené erősségét.

2.4 A C# programozási nyelv

A C# (kiejtése: *szí-sárp*, esetenként *cisz*) a Microsoft által a .NET keretrendszer részeként kifejlesztett objektumorientált programozási nyelv. A nyelv alapjául a C++ és a Java szolgált. A C#-ot úgy tervezték, hogy meglegyen az egyensúly a fejlesztő nyelvi szabadsága és a gyors alkalmazásfejlesztés lehetősége között.

A C# az a programozási nyelv, ami a legközvetlenebb módon tükrözi az alatta működő, minden .NET programot futtató .NET keretrendszert, valamint erősen függ is attól: nincsen nem menedzselt, natív módban futó C# program. A primitív adattípusai objektumok, a .NET típusok megfelelői. Szemétgyűjtést használ, valamint az absztrakcióinak többsége

(osztályok, interfészek, delegáltak, kivételek…) a .NET futtatórendszert használja közvetlen módon.

A C vagy C++ nyelvhez hasonlítva a C# több korlátozást és továbbfejlesztést is tartalmaz. A mutatók és a nem ellenőrzött aritmetika csak egy speciális, nem biztonságos módban (unsafe mode) használható. A legtöbb objektum-hozzáférés csak biztonságos hivatkozásokon keresztül tehető meg, és az aritmetikai műveletek debug módban túlcsordulás szempontjából ellenőrzöttek. Az objektumok nem szabadíthatók fel közvetlen módon, ehelyett a szemétgyűjtő szabadítja fel őket, mikor már nincs rájuk hivatkozás. Ez a módszer kizárja a nem létező objektumokra való hivatkozás lehetőségét. A destruktorok nem elérhetőek. A legközelebbi megfelelőjük az IDisposable interfész, ami a using blokkal együtt kikényszerítheti az azonnali felszabadítást. A finalizerek szintén rendelkezésre állnak, de nem váltanak ki azonnali felszabadítást. A nyelv csak egyszeres öröklődést támogat, de egy osztály több interfészt is megvalósíthat. A C# sokkal típusbiztosabb, mint a C++. Az egyetlen implicit konverzió a biztonságos konverzió, úgy, mint az egészek tágabb intervallumba konvertálása vagy a leszármazott osztályok alaposztályba konvertálása. Nincs implicit konverzió az egészek és a logikai típus (boolean) között, a felsorolás tagok és az egészek között. Nincsenek *void* mutatók (bár az *Object* osztályra mutató mutatók hasonlóak), valamint bármely, a felhasználó által definiált implicit konverziót explicit módon meg kell jelölni. A tömbdeklaráció szintaxisa eltérő (int[] a = new int[5] az int a[5] helyett). A felsorolás adattagjai a saját névterükben helyezkednek el. A C# 1.x nem rendelkezik template-ekkel, de a C# 2.0 már rendelkezik generikusakkal.

A legtöbb programozási nyelvtől eltérően a C# megvalósítások nem rendelkeznek önálló, eltérő osztály- vagy függvénykönyvtárakkal. Ehelyett a C# szorosan kötődik a .NET keretrendszerhez, amitől a C# kapja a futtató osztályait és függvényeit. A .NET keretrendszer osztálykönyvtárat tartalmaz, ami a .NET nyelvekből felhasználható egyszerű feladatok (adat reprezentáció és szöveg manipuláció) végrehajtásától kezdve a bonyolult (dinamikus ASP.NET weblapok generálása, XML feldolgozás és reflekció) feladatokig. A kód névterekbe van rendezve, mely a hasonló funkciót ellátó osztályokat fogja össze. Például System.Drawing a grafikai, System.Collections az adatstruktúra és System.Windows.Forms a Windows Forms funkciókat fogja össze.

A Microsoft benyújtotta a C# nyelvi specifikáció az ECMA-hoz formális szabványosításra. 2001 decemberében az ECMA kiadta az ECMA-334 *C# Language Specification* szabványt. 2003-ban a C# ISO szabvány lett (ISO/IEC 23270).

2.5 A Microsoft Visual Studio 2008 fejlesztői környezet

A Visual Stúdió első változata 1997-jelent meg, Microsoft Visual Studio 97 néven. Rá egy évvel 1998-ban megjelent az újabb változata, amely az elnevezésben nem évszámot tartalmazott, hanem verziószámot: Microsoft Visual Studio 6. Ezen verziók az alábbi programozási nyelvet támogatták: Visual Basic, Visual C++, Visual FoxPro, Visual J++.

Egy hosszabb szünet után, 2002-ben megjelent a Microsoft Visual Studio .NET 2002, amely egy nagyon fontos technológiai újítást hordozott magában a Windowsos alkalmazások terén, konkrétan a .NET Framework 1.0 –t. A következő programozási nyelvet tartalmazta: Visual Basic .NET, Visual C++ .NET, Visual C# .NET, Visual J# .NET, ASP.NET.

A következő évben kiadásra került a 2002-esnek egy felújított változata, amely a .NET Framework 1.1 verzióját tartalmazta, a szoftver Microsoft Visual Studio .NET 2003 néven került a boltok polcaira.

2005-ben megjelent a Microsoft Visual Studio 2005, amely már a .NET Framework 2.0 tartalmazta. Ezen generációnak már van egy ingyenesen használható Express néven futó változata, amely nem tartalmaz minden funkciót, de otthoni és oktatási célokra kifejezett használható. Az alábbi nyelveket támogatja: Visual Basic .NET, Visual C++ .NET, Visual C# .NET, Visual J# .NET, ASP.NET.

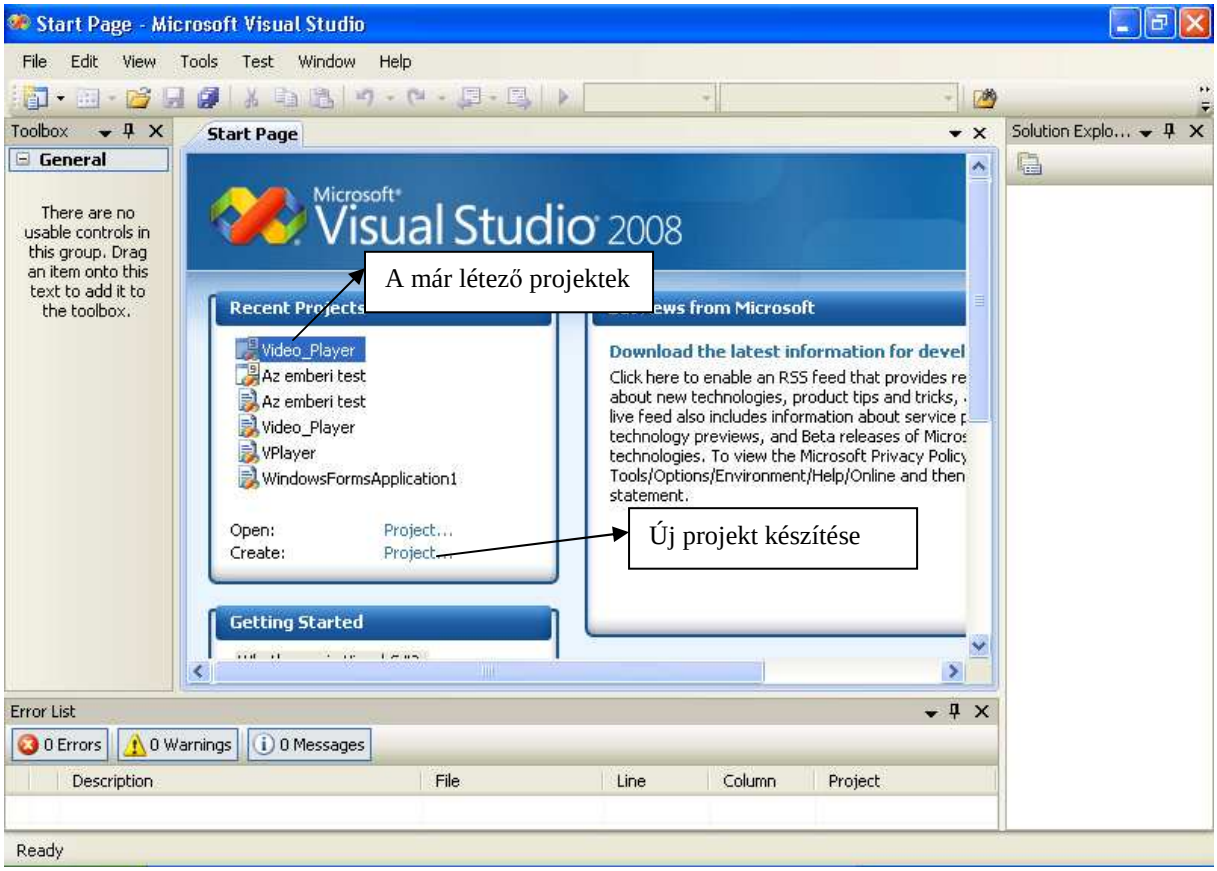
A legutolsó verzió 2007 végén látott napvilágot, Microsoft Visual Studio 2008 néven, amely a továbbfejlesztett .NET Framework 3.0-ás, majd 3.5-ös változatát is támogatja. A támogatott nyelvek listája megegyezik a 2005-ös verzióéval. Azt azért érdemes megemlíteni, hogy letölthető hozzá egy ún. Crystall kiegészítő, amely segítségével kisebb módosítással, a Delphiben megírt kódok is fordíthatók.

A már korábban ismertetett programozási nyelv egyik fejlesztői környezete. A más nyelvekhez készített fejlesztői környezetekkel (pl.: Javanál a Netbeans vagy az Eclipse, Deplhinél a Delphi 2007) összehasonlítva arra megállapításra jutottam, hogy ez a program nagyon is a fejlesztő keze alá dolgozik, megkönnyítve annak munkáját.

Egy grafikus felülettel rendelkező program készítésekor a felület elkészítését majd életre keltését ezzel a fejlesztői programmal és magával nyelvvel, sokkal egyszerűbben tudtam elvégezni, mint mondjuk a javas Netbeansszel. Az egyes kontrollokat működtető metódusokat egyszerűbben lehet azokhoz rendelni, elérni, mint mondjuk a NetBeansben.

A program elkészítését először Javaban gondoltam elkészíteni. Majd egy jó barátom, aki .NET fejlesztő mutatta meg a C# programozási nyelvet és annak a fejlesztői környezetét. Amikor láttam, hogy milyen praktikus is, akkor határoztam el, hogy ezen a nyelven készítem el a programot. Számomra sokkal egyszerűbben lehet benne egy GUI-val (Graphical User Interface - Grafikus felhasználói felület) rendelkező programot elkészítése, mint a Javaban a SWING (grafikus felületet készítését szolgáló eszközkészlet) használatával.

Az egyes programokat a Visual Studio projektekként kezeli. Az indulási képernyőn több lehetőség közül választhatunk: megnyithatunk már létező projekteket, vagy készíthetünk újat. Természetesen ezek a File menüből is elérhetjük. Új projekt készítése esetén a több projekttípus közül is választhatunk (pl.: parancssoros program, grafikus program, de akár csak egy osztályt is létrehozhatunk stb.)

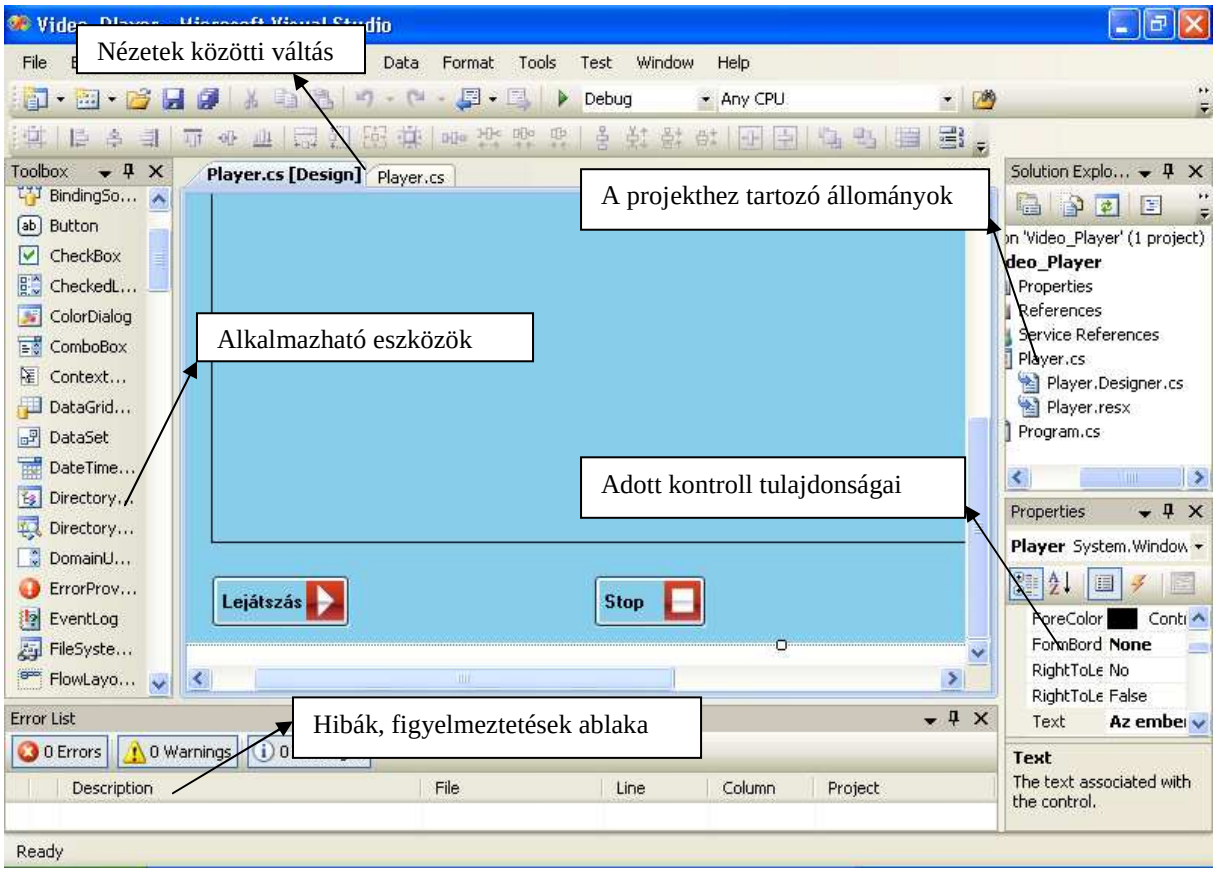


A Microsoft Visual Studio 2008 indulási képernyője

Konkrét program írása közben lehetőségünk van váltogatni a tervező (*Designer*) és a program kódját tartalmazó nézetek között, annak megfelelően, hogy mikor melyikre van szükségünk. A program bal oldalán választhatunk azokból az eszközökből (*Toolbox*), amiket fel lehet használni a program felületének elkészítéséhez. Jobb oldalon találjuk az adott projekthez kapcsolódó állományok, erőforrások stb. listáját (*Solution Explorer*). Ez alatt találjuk meg, azt a panelt (*Properties*), amely a felületen elhelyezett kontrollerek tulajdonságait tartalmazza, amit igényeinknek megfelelően állítgathatunk.

Az ablak alsó részén, egy kis panelen (Error List) a program fordítója közli velünk ez esetleges hibákat, figyelmezetéseket, amelyek a program működésével kapcsolatosak. A szintaktikai hibákból eredő problémákra már gépeléskor figyelmeztet (pl.: pontosvessző elhagyása; kapcsos zárójelek nem megfelelő helyen történő nyitása, zárása stb.).

A képen nagyon jól láthatók azok a programelemek, amiket az előbb említettem, amelyek nagy segítséget jelentenek egy program fejlesztése során.



Fejlesztés közben így néz ki a Microsoft Visual Studio 2008 felülete

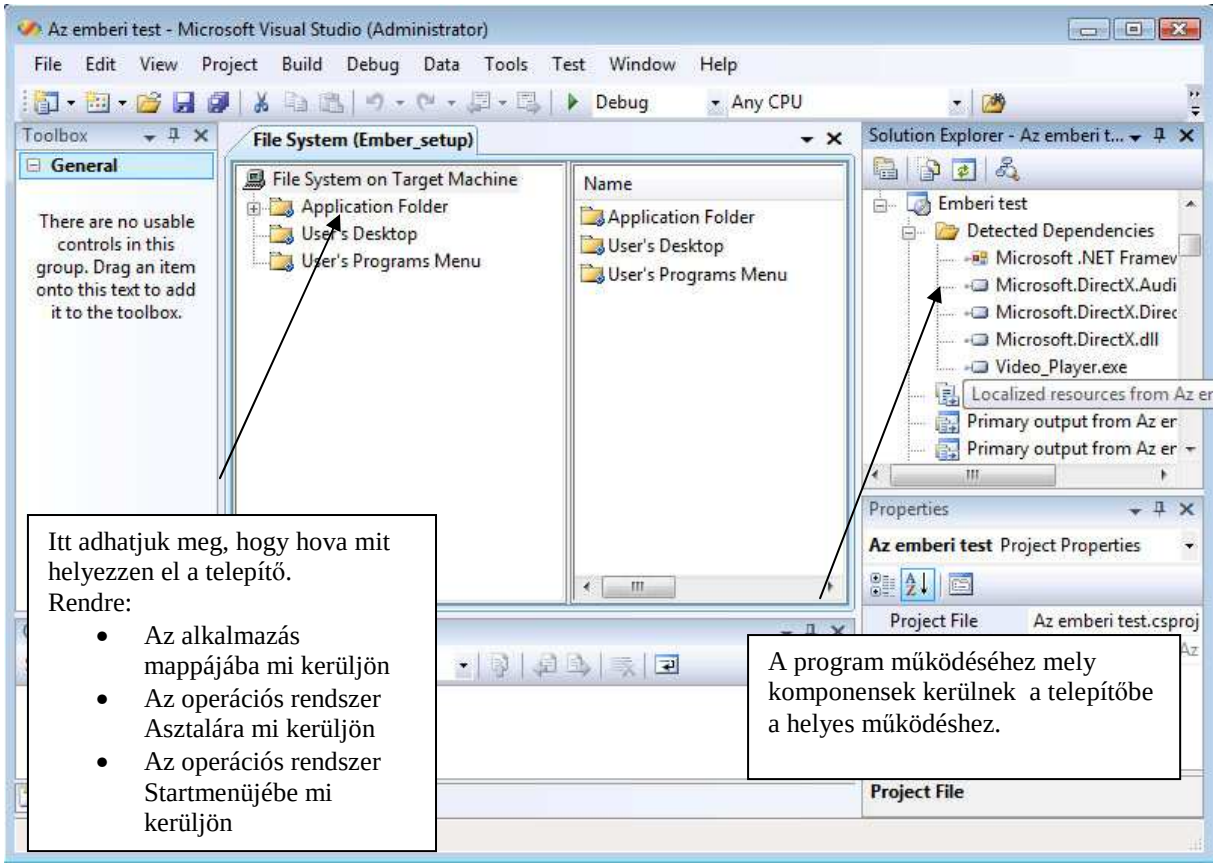
A programozói gyakorlatom során először most használtam ún. breakpointokat hibadetektálásra, az egyes programegységek helyes működésének az ellenőrzésére. A jó felületnek köszönhető, hogy a futó program kódját menet közben ki lehet elemezni, az aktuális változóértékeket meg lehet tekinteni. Elég csak a futó kódban a változó fölé vinni az egérmutatót, és kiírja annak az aktuális értékét. Természetesen egyéb adatszerkezetekre is igaz ez, például egy tömb elemit is meg tudjuk tekinti.

A Microsoft Visual Studio számtalan olyan funkciót hordoz magában, amiket egy programozó jól ki tud aknázni munkájának megkönnyítésére, hatékony program írására.

3. Az oktatóprogram bemutatása

3.1 Az oktatóprogram telepítése

Az oktatóprogram egy DVD lemezen található. A DVD-ROM meghajtóba történő behelyezést követően a telepítő magától elindul. Az automatikus indítást a Windows operációs rendszerek elvégzik, ha az adott adathordozón található egy úgynevezett autorun.inf állomány. Ez mutatja meg, hogy mit indítson el a beolvasást követően.



A Microsoft Visual Studio telepítő készítője

A program telepítőjét a Microsoft Visual Studioval készítettem. Az már korábban említett *Solution Explorer*ben az aktuális projekthez hozzá lehet adni egy *Setup projek*tet. Amikor ez hozzáadásra kerül, különböző beállítási lehetőségek közül lehet választani. A fejlesztői környezet az alapértelmezett telepítési útként a szokásos Program Files mappát kínálja fel. Nekem ez nem megfelelő, mert a tananyag szöveges része file-okban van tárolva, az oktatóprogram onnan olvassa ki. Az elérési utat úgymond abszolút módon adtam meg: „*C:\Az emberi test\Az emberi test\Az emberi test\Tananyag*”, ahol a tanegységnek megfelelő

állományra hivatkozok. Ezért fontos volt hogy a telepítési mappa ennek megfelelően kerüljön kialakításra.

További beállításként megadtam, hogy az operációs rendszer Asztalára, és Startmenüjébe is helyezzen el egy-egy indító ikont.

A telepítő mellett megtalálható egy .NET Framework 3.5 verziójú telepítő, ami nélkül az oktató program nem futtatható, és ha nincs a gépen, akkor telepíteni lehessen. Elhelyeztem továbbá egy DirectX 9-es verziójú telepítőt is, mert ha ez nincs a felhasználó gépén, akkor a programban lévő videó lejátszó nem használható.

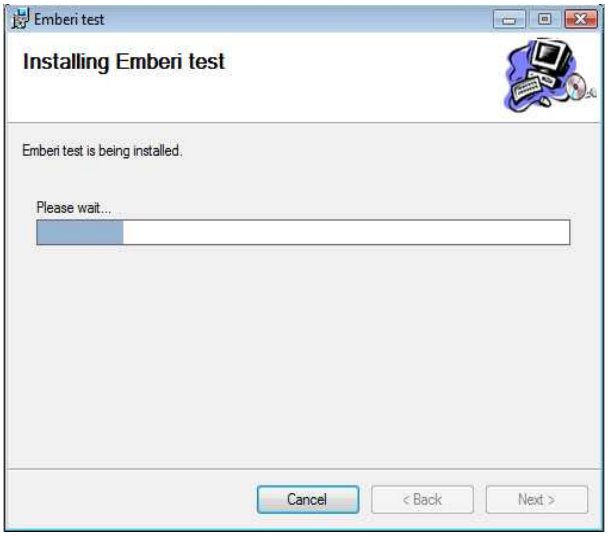
A videók DIVX formátumban vannak kódolva, AC3-as hangsávval. Ha ezek sem lennének feltelepítve, akkor ezen telepítők is megtalálhatók a teljes működés érdekében.

Olyan funkciót is beállítottam, hogy amikor a telepítő indításakor ellenőrizze, hogy az oktatóprogram volt-e már telepítve az adott számítógépen. Ha nem találja meg, akkor felkínálja a telepítést az adott helyre. Ha megtalálja, akkor felkínálja a javítást vagy az eltávolítást. Ez egy praktikus beállítás volt, mert ha valami miatt a már telepített program nem működne, akkor egy megfelelő kattintással pillanatok alatt a hiba elhárítható.

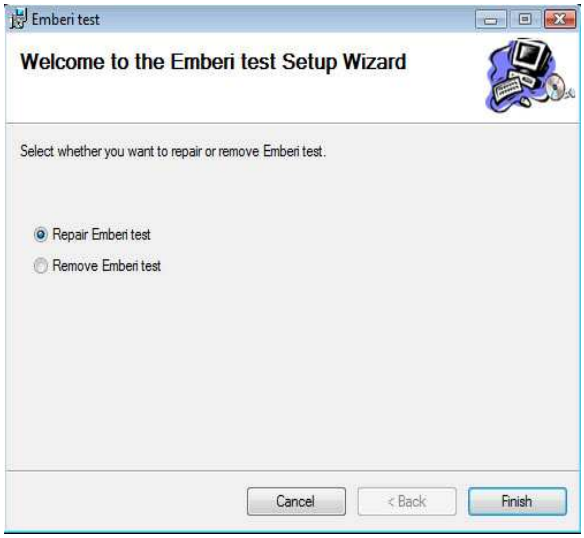
Amikor minden beállítást megadtam, egy fordítást (*Build*elést) kellett végrehajtanom. Ez a művelet nem az oktatóprogram kódjára vonatkozik, hanem a telepítő projektre. Az eljárás végére előállított egy „setup.exe” és egy „Emberi test setup.msi” állományt. Bármelyikre is kattintunk a telepítési folyamat elindul.

A telepítési művelet pár percet vesz igénybe, ha minden egyéb szükséges program a gépen van (.NET Framework, DIVX codec, AC3 codec, DirextX). Ha ezek is telepíteni szükséges, akkor további percekre van szükség, de ezen programok telepítése hamar végbemegy.

Az oktatóprogram telepítési képernyője lépésről lépésre



Az oktatóprogram telepítője munka közben



Ha telepítő megtalálja a gépen az oktatóprogramot

Az automatikus indítás elkészítéshez az Autorun Wizard programot használtam. Ezzel a kis programmal egyszerűen megadhatom, hogy melyik állományra szeretnék „*autorun.inf*” állományt készíteni és azt el is készíti. Ezt követően csak annyi a volt feladatom, hogy kiírni a lemezre ezeket az állományokat. Arra viszont figyelni kell, hogy a lemez gyökér könyvtárába kerüljön az imént említett állomány, mert csak ekkor fog az oktatóprogram telepítője elindulni.

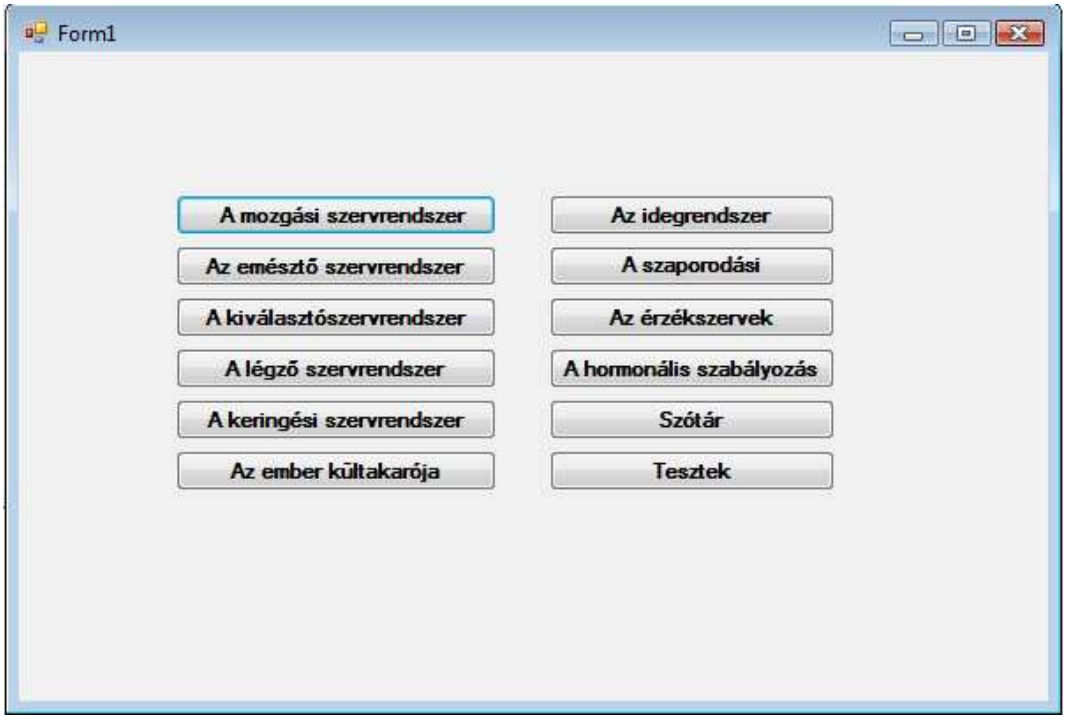
3.2 Az oktatóprogram indítása és felülete

A telepítést követően a program két helyről is elindítható parancsikon segítségével, melyek megtalálunk az operációs rendszer Asztalán, és Startmenüjében. Többszöri próbálkozás után egy egyszerű ikonnál maradtam, amire kétszer kell kattintanunk az oktatóprogram indulásához.

Az oktatóprogram ikonja

A program ablakát úgy igyekeztem megtervezni, hogy egy átlagos monitoron is könnyen használható legyen, ennek megfelelően a mérete 1000*724 képpont. Az ablak szegélyéről levettem a Tálcára leküldő és Teljes méretre nagyító gombokat. Mindezt azért, mert nincs funkciójuk. A program ablakát szándékosan nem méretezhetőnek állítottam be, induláskor a képernyő közepén jelenik meg, melyet szintén tudatosan állítottam be. Azért döntöttem így, hogy a program a működése közben egy egységes képet adjon a könnyebb kezelhetőség érdekében, valamint a szövegek pozicionálása is egyszerűbb volt így.

A legelső változat egy nagyon egyszerű felületű volt, ahol csak gombok voltak elhelyezve a *Form*on (a program ablakán). Ezekre a gombokra klikkelve újabb *Form*ok voltak elhelyezve témáknak megfelelően. Nem tudom megmagyarázni miért, de éreztem, hogy ez így nem jó, és előről kezdtem az egész program írását, tervezését. Ennek az eredménye az aktuális felület.



Az oktatóprogram első felülete

Az új felület kialakítása során elsődleges szempont volt, hogy az egyes témakörökhöz bármikor hozzá lehessen férni, de legyenek kihagyott, üres területek a felületen. Ahogy elkezdtem megismerni a Microsoft Visual Studiot, a beépített eszközök között megtaláltam a „*panel*” kontrollt. Kisebb próbálgatások után úgy gondoltam, hogy ezt fogom használni a felület elkészítéséhez.

A menügombokat is egy panelre helyeztem fel, mely minden körülmény között látszik, ezáltal a rajta lévő gombok is. Így azt a szempontot, hogy a témák bármikor elérhetők legyenek, de legyen nagy üres felület, megoldottam.

A program viszonylag sok panelt tartalmaz. Ezek háttérszíne világoskék – vagy ahogy a Visual Studio nevezi: SkyBlue. Mindegyiket külön névvel láttam el a tartalmának megfelelően, hogy könnyű legyen őket a programozás során megkülönböztetni (pl.: *MozgasPanel1*, *LegzesPanel1* stb.).

A menügombok ennek megfelelően az adott téma első paneljére, vagy a szótár esetében annak egyetlen paneljére, vagy a teszt esetében annak paneljére hivatkoznak.

Az oktatóprogram induló képernyője

A kezdeti felülethez képest nagyban átalakult, az újratervezett felület felhasználóbarát, egyszerű kezelést biztosít a felhasználó számára.

3.3 Az oktatóprogram menüje

A menü elkészítésénél elsődleges szempont volt az áttekinthetőség. Ennek megvalósítása véleményem szerint jól sikerült. Színes, figyelemfelkeltő és az egyértelmű működésről árulkodik. A témák kiválasztását egy-egy gomb lenyomásával teheti meg a felhasználó. A gombok hátterének kisméretű képeket használtam, melyek ábrája az adott témakörhöz kapcsolódik. A gombok alatt címkékre (*label*ekre) helyeztem fel a témakörök címeit. Fontosnak véltem az első tervezet után, hogy a szöveg ne a gombokon legyen, hanem alattuk. Az áttekinthetőséget ez szolgálta a legjobban.

3.4 A szöveges tartalmak megjelenítése

Az egyes témakörök gombjaira klikkelve lehet az adott témához kapcsolódó tananyagot megjeleníteni. Már korábban is írtam, hogy a tartalmak témakörönként paneleken helyezkednek el. Ezek a panelek kezdetben „láthatatlanok”. Amikor kiválasztunk egy témakört és annak gombjára kattintunk, akkor az adott téma első panelje jelenik meg. A paneleknek van egy úgynevezett „*Visible*” tulajdonsága. Ennek az értékét igazra (true) vagy hamisra (false) állíthatjuk. Alapértelmezettként igaz ez az érték, azaz a panel megjelenik a program indulásakor. Én minden panelnél ezt az értéket hamisra állítottam, kivéve az indulóképernyőét, így csak akkor jelenik meg, ha az adott gombra klikkelünk. (pl.: *MozgasPanel1.Visible = false;* de a kezdőképernyő panelre vonatkozó érték: *KezdoPanel.Visible = true;*)

Minden panelre igaz az, hogy annyi szöveget tartalmaznak, amennyi az adott területre kvázi oldalra kifér. A panelek mérete egységesen 995*615 képpont, kivéve a menügombokat tartalmazó panelé, mert az 1004*89 képpont. A panelek mérete kicsit túlnyúlik az ablak méreténél, amely nem zavarja a program működését. A felhasználó semmit nem érzékel ebből, de tervező nézetben látszik. Erre a kis apróságra a program befejezésekor figyeltem fel, természetesen a forráskódban egyesével ez korrigálható, amit a dokumentáció elkészítésekor még nem tettem meg.

A panelek alján gombokat helyeztem el, amelyek segítségével oda-vissza tudunk lapozni egy témakörön belül. Ezek a gombok a menügombokhoz hasonlóan működnek. A rajtuk lévő szövegnek megfelelő tartalommal rendelkező panelt láthatóvá teszik, de az összes többi panel láthatósági értékét hamisra állítják. Hogy ne kelljen minden egyes gombnál az összes panel láthatóságát beállítani, írtam egy eljárást, amely minden panel láthatóságának az értéket hamisra állítja kivétel nélkül (neve: *panelsHide()*). Ezt az eljárást meghívom ott, ahol az a metódus szerepel, amely a gombot vezérli. Majd a következő sorban láthatóvá teszem azt a panelt, amelyiket az adott gombnak kell láthatóvá tennie. A gyakorlatban mindig csak egy panel látható, tehát valójában fölösleges mindegyik panelt „elrejteni”, de én ezt így láttam egyszerűnek. Egy példa arra, hogyan valósítottam meg egy gombra kattintáskor, hogy az adott panel látható legyen, míg a többi „elrejtésre kerül”.

```
private void MozgasMasodikButton2_Click(object sender, EventArgs e)
{
     this.panelsHide();
     this.MozgasPanel2.Visible = true;
}
```

A témakörök címét egyszerűen úgynevezett „*label*ekre” helyeztem el. A *label* szintén a beépített eszközök között megtalálható. A címeket igyekeztem középre pozícionálni a paneleken. Ezt úgy tettem meg, hogy az aktuális cím méretét (szélességét) kivontam a panel szélességéből, majd elosztottam kettővel. Az így kapott számot adtam meg a *label* egy tulajdonságának az értékéül. Ez a tulajdonság „*Location*” (elhelyezkedés) néven jelenik meg, amely rendelkezik egy X és Y komponenssel. Természetesen ezek X és Y koordináták, amely számítógépes grafikában megszokott módon a bal felső sarokban a 0,0 ponttal kezdődnek. Ez a 0,0 pont történetesen az adott panel bal felső sarka. A fent említett értéket az X-nek adtam értékül, így sikerült a címeket középre helyezni. A *label* kontrollnak be lehet állítani betűkre vonatkozó tulajdonságokat. Ezek *Font* néven érhetők el a tulajdonságok között. Be lehet állítani azok méretét, színét típusát és stílusát. A címekhez 12-es betűméretet, félkövér betűstílust, és Microsoft Sans Serif betűtípust használtam.

A szöveges tartalmak elhelyezésére is a beépített eszközök közül válogattam. Konkrétan a *TextBox*okat és a *RichTextBox*okat (szövegdobozokat) vegyesen alkalmaztam. A képekhez fűzött magyarázó szövegek tervező nézetben kerültek begépelésre. A tananyaghoz kapcsolódó szövegek egyszerű szöveges állományokban vannak, melyek futási időben kerülnek kiolvasásra. Azért döntöttem a külső állományok használata mellett, az esetleges gépelési hibákat így könnyebb javítani. Amikor ezek a szövegdobozok felkerülnek a Formra, kerettel rendelkeznek, és fehér háttérrel. Ezek a tulajdonságok nem illettek az oktatóprogram felületére ezeket meg kellett változtatnom. A Visual Studio ebben is segítségemre volt. A hátterüket a már fent leírt „SkyBlue”-ra állítottam, és az azokat határoló keretet beállítottam, hogy ne legyenek (*BorderSytle*). A méretüket tetszés szerint változtattam, ahogy a formai elhelyezés kívánta. Igyekeztem olvashatóan, tagoltan elhelyezni a szövegeket. A bekezdésekkel nem volt különösebb probléma, mert a szöveges állományokban lévő tabulátorokat ugyan olyan jól értelmezte beolvasáskor, ahogy azt vártam.

A file-ban lévő szöveget egy String típusú változóba olvastam be, majd ezt a változót adtam értékül az adott szövegdoboz *Text* tulajdonságának. Az alábbi kódrészlet ezt szemlélteti.

```
      String text = File.ReadAllText(@"C:\Az emberi test\Az emberi test\Az
emberi test\Tananyag\emesztes1.txt");
      this.richTextBox9.Text = text;
```

A szövegdobozokat akkorára méreteztem, hogy pont elférjen benne a szöveg, mindezt azért mert így könnyebb volt a szövegeket elhelyezni, valamint ha a szöveget görgetni lehet a dobozban, számomra rontotta a program szépségét. Ezért ezt a tulajdonságot is letiltottam. Ezt úgy tudtam megtenni, hogy a szövegdobozok *ScrollBars* tulajdonságának az értékét *None*-ra állítottam.

A szövegdobozok nemcsak arra szolgálnak, hogy szövegeket jelenítsünk meg bennük, hanem hogy a futás közben beleírt szöveget felhasználjuk. Ez viszont azt eredményezi, hogy a beolvasott szöveg módosítható. Meg is jelent előttem az a kép, hogy a gyerekek mindenféle jópofa szövegeket írnak be, majd jelzik a tanáruknak, hogy miket tartalmaz ez a program. Erre a problémára is nagyon egyszerű a megoldás. A szövegdobozok tulajdonságai között találunk egy olyan lehetőséget, amivel megszabhatjuk a szövegdoboz írhatóságát. Ha a *ReadOnly* értékét *True*-ra állítjuk, akkor a szövegdoboz tartalma nem módosítható. Ezzel a kis művelettel a készülődő diákcsínyt már ki is küszöböltem.

3.5 A képek megjelenítése

A képek elhelyezését a *PictureBox*ok segítségével tudjuk megvalósítani. A képeket importálással helyeztem el az oktatóprogram képdobozaiba. Az importálás során a képek beépülnek a forrásba, így nem szükséges azokat az oktatóprogram mellé mellékelni, mint a tananyag szövegét tartalmazó állományokat.

A képek egy kis nézetben jelennek meg a tanegységekben. Természetesen ezeket a képeket meg lehet tekinteni eredeti méretükben is. A *PictureBox*ok egyik tulajdonságának segítségével meg tudjuk határozni, hogy a bene lévő kép hogyan jelenjen meg. Ezt a *BackgroundImageLayout*al tudjuk megtenni. Ha azt állítjuk be, hogy a kép elrendezése *Tiled* akkor a kép a dobozban mozaikszerű elrendezést kap. Ha *Center*t választjuk, akkor egyszerűen a doboz közepére igazítja a képet. A *Strech* opcióvel a képet a doboz aktuális méretéhez igazítja. Én a kis képek esetében az utóbbit használtam. A nagy képek esetében

pont akkorára állítottam a doboz méretét, amekkora a kép mérete, természetesen képpontokban számolva.

A képek mellett elhelyezett magyarázószövegek hívják fel a figyelmet a nagy kép elérésének a lehetőségére. A nagyobb kép megtekintéséhez a felhasználónak nem kell mást tennie, mint az adott képre egyszer kattintania. Ekkor megjelenik az érintett kép nagyobb méretben. Ezek a nagyméretű képeket tartalmazó dobozok tervezőnézetben lettek kialakítva úgy, hogy a „Visible” tulajdonságuk alapértelmezetten hamisra lett beállítva, de kép az importálva lett bennük. Ez a program írásakor úgy nézett ki, hogy amikor egy adott panelt elkészítettem a legvégén helyeztem rajta nagy képeket tartalmazó *PictureBox*okat.

Amikor a felhasználó rákattint a képre, a nagykép *Visible* tulajdonságának az értéke igaz értéket vesz. Illetve ezek a metódusok jelenítik meg a „bezár” szöveget tartalmazó címkéket is.

Egy objektumhoz több esemény is hozzárendelhetünk. A képekre az egy bal egérgombbal történő kattintást rendeltem hozzá. Ez egyszerűen „*Click*” néven fut a C#-ban. Ezt kiválaszthatjuk az eseménykezelőből, vagy ha az adott objektumra kattintunk kettőt tervező nézetben, akkor automatikusan erre az egy balklikkes eseményre iratkozunk fel, és a Visual Studio be is hozza a forráskód azon részét, ahol ez a metódus megjelenik. Innentől kezdve már csak azokat a sorokat kellett begépelnem, amiket akartam, hogy abban a pillanatban elvégezzen a program. Az alábbi kódrészlet egy olyan metódus, ami egy kis képre kattintva megnyit egy nagyobb képet.

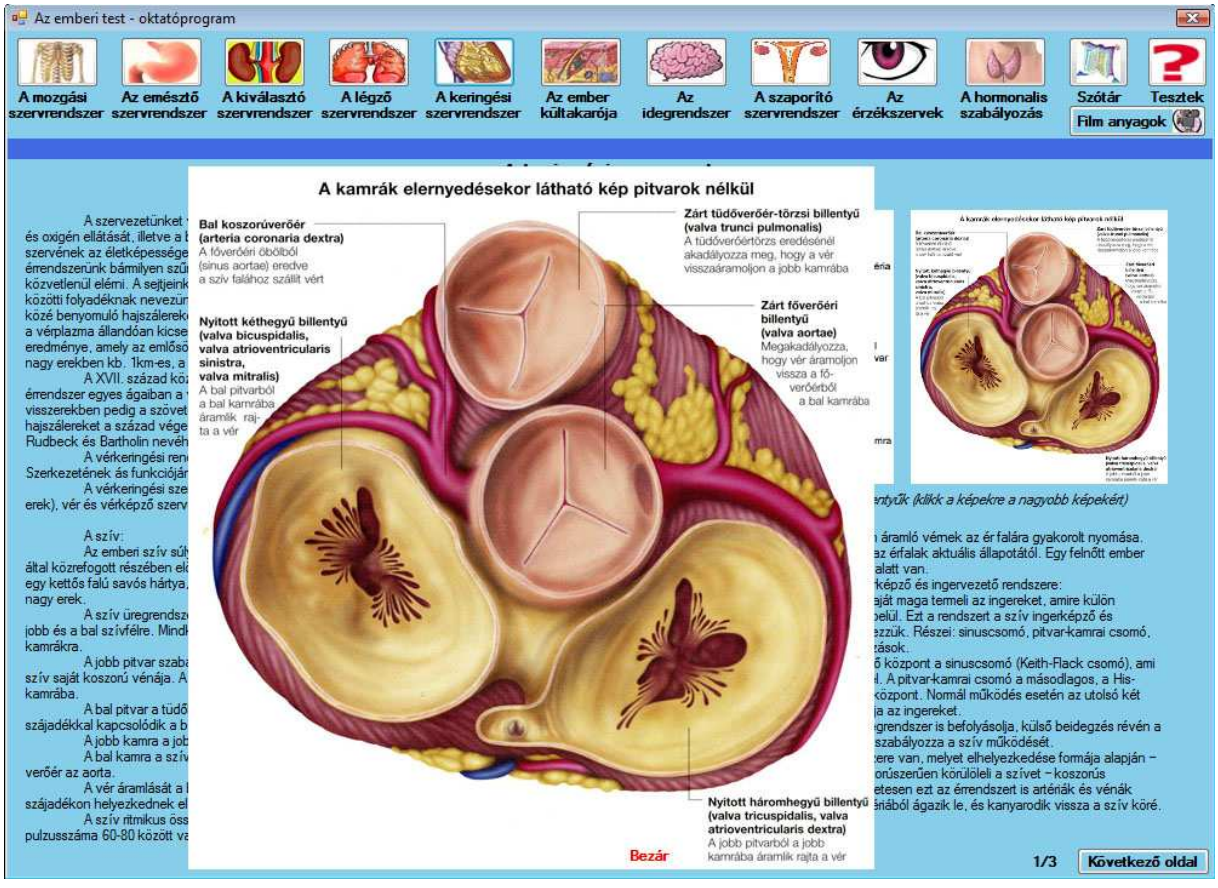
```
private void IzuletKicsi_Click(object sender, EventArgs e)
{
      this.IzuletNagyBezar.Visible = true;
      this.IzuletNagy.Visible = true;
}
```

A nagy kép bezárásához nem kell mást tenni, mint a megjelent kép valamelyik sarkában található „bezár” feliratára kattintani. A „bezár” szövegek *label*eken vannak elhelyezve. Ezeknek a *label*eknek a *Cursor* tulajdonságánál a „*Hand*” opciót állítottam be. Ez azt a változást eredményezi futási időben, hogy amikor az egeret föléjük mozgatjuk az egérmutató egy kézre vált át, mint a webes linkek esetében egy böngészőben. A könnyebb használat érdekében használtam ezt a beállítási lehetőséget. Ezek a címkék alapestben nem láthatók, csak a képekre történő kattintások teszik elérhetővé azokat. Amikor erre a szövegre

kattintunk, a kép eltűnik. A mögöttes kód nem tesz mást, mint az érintett kép, és a képhez tartozó „bezár” címke láthatósági beállítását hamisra állítja, így azok eltűnnek. Az alábbi kódrészlet ezt a műveletet végzi el.

```
private void IzuletNagyBezar_Click(object sender, EventArgs e)
{
      this.IzuletNagyBezar.Visible = false;
      this.IzuletNagy.Visible = false;
}
```

A fenti kódrészletből is látható, hogy mind a képeket tartalmazó dobozokat, mind a címkéket beszédes nevekkel láttam, így téve könnyebbé számomra az azonosításukat. Ez szerintem nagyon fontos dolog, mert a későbbiekben lehetetlenné válik a kód értelmezése, mert nem lehet majd tudni, hogy melyik képre is vonatkozik a metódus. Talán a képek nevei segíthetnének ebben, de így nem lehet áttekinthető kódot írni.



Az oktatóprogram működés közben nagy kép megnyitásakor

3.6 A szótár bemutatása

Minden témakör végén található egy kisebb lista, ami az ahhoz kapcsolódó latin kifejezéseket tartalmazza. Természetesen nagyon kevés latin kifejezésről van szó, az egész program kb. 130 latin-magyar kifejezéspárt tartalmaz.

Az oktatóprogram írása közben felmerült az igény, hogy legyen benne egy szótári rész, ami ezeket a kifejezéseket tartalmazza. Ezek a kifejezéspárok témakörönként külön szöveges állományban vannak tárolva a következő formában: máj - hepar. Egyszerűen annyit tettem, hogy egy szöveges állományba összemásoltam a fejezetekhez tartozó kifejezés párokat. Ezt követően kitöröltem a magyar megfelelőket, majd elmentettem „latin.txt” néven, majd ugyanezt elvégeztem a latin megfelelőkkel, és az eredményt elmentettem „magyar.txt” néven. A két állományban a szavak sorrendje nem változott, így ha a hatodik sorban a „latin.txt” file-ban a dens szerepel, akkor a „magyar.txt” állományban a hatodik sorban a fog szónak kell szerepelnie. Ez az utolsó sorban szereplő szavakra is igaz. Ezt használtam ki a keresés megoldásához is. Azért döntöttem a file-ban tárolás mellett, mert az esetleges gépelési hibákat sokkal könnyebb így javítani, mint ha a forráskódban kellene keresgélni.

A szótárban a párosítási lépést ezzel megoldottam. A következő szint az volt, hogyan oldom meg a keresést. Egyszerűen deklaráltam két 150 elemmel rendelkező tömböt, magyar és latin nevekkel illettem őket. Ezekbe beolvastam az elnevezésnek megfelelően a szavakat a „magyar.txt” valamint a „latin.txt” állományokból. Egy kicsit féltem a file-kezeléstől, mert a C-ben nem volt *StreamReader és FileStream* osztály, egy kicsivel rövidebb is a C-ben lévő kód, de végül nagyon egyszerűen megírtam, és elsőre helyesen működött a tömbök feltöltése.

Az oktatóprogram szótár panelén lehet választani azon opciók közül, hogy a latin vagy magyar szónak keressük a megfelelőjét. Ha a magyar gombra kattintunk, akkor a magyar szavakat tartalmazó tömbben keres, ha a latinra, akkor a latin szavakat tartalmazó tömbben keres. A választható opciók eléréséhez rádiógombokat (*RadioButton*) raktam fel a panelra. A keresés gombra kattintva a gombhoz rendelt függvény ellenőrzi, hogy melyik gomb lett kiválasztva.

Ezt hogyan is lehet megvizsgálni? A rádiógomboknak van egy úgynevezett *Checked* tulajdonsága. Ennek az értéke logikai értékként igaz (*true*) vagy hamis (*false*) lehet. Értelemszerűen, ha a rádiógomb be van jelölve, akkor igaz értéket képvisel, egyébként hamisat.

A keresés gombhoz írt metódust úgy írtam meg, hogy arra is figyel, ha egyik rádiógomb sincs kiválasztva. Ha ilyen eset van, akkor egy felugró ablak figyelmezteti a felhasználót a következő szöveggel: „Nem választottad ki, hogy melyen szót keresel!”.

A keresendő szót egy *TextBox*ba kell beírni. Már korábban írtam, hogy ezeknek a szövegdobozoknak a *Text* nevű komponense hordozza a szöveget. Az ide beírt szöveget egy *string* típusú változóba teszem be, amikor a keresés gombra kattint a felhasználó. A metódus arra is figyel, hogy legalább 3 karaktert adjon meg a felhasználó a beviteli mezőbe. Ha ennél kevesebbet ad meg, akkor szintén egy felugró üzenetablak figyelmezteti erre.

Ha a fenti feltételek teljesültek, akkor elindul a keresés a latin vagy magyar tömbben a választásnak megfelelően. A szavak vagy szövegrészek keresésére a *Contains()* beépített metódust használtam. Ez a metódus szöveges típusú adattagokon használható, és igaz értékkel tér vissza, ha a paraméterül megadott szöveg megtalálható a változó által képviselt szövegben. A kódban ez így fordul elő.

```
if (magyar[i].Contains(word) == true)
```

Ha keresés eredményesen ér véget, akkor a keresett szórészlethez kapcsolódó szót, attól függően, hogy magyar vagy latin kifejezések között kerestünk kiírja, majd mellé a másik megfelelőjét, mindezt egy másik szövegdobozba. Ha több ilyen is van, akkor mindegyiket kiírja, természetesen soronként.

A fenti kódrészletből látható, hogy a magyar szavakat tartalmazó tömbön szalad végig a keresés, és a *word* változóban tárolt karterláncot keresi annak elemeiben. Ha megtalálja, akkor az adott pillanatban az *i* változó által képviselt indexen lévő szavak összefűzve kiírja a szövegdobozba. A fenti kódrészlet ennek megfelelően kiegészítve az egyik tömbre nézve:

```
if (magyar[i].Contains(word) == true)
{
      this.ResultBox.AppendText(magyar[i] + "     -     " + latin[i] +
"\n");
      tmp++;
}
```

A fenti kódrészletből már csak a *tmp* változó szorul magyarázatra. Az elnevezés az angol *temp* szóból ered, annak ideiglenes, segéd mivoltára utalva. Annyi a feladata, hogy

számlálja a találatok számát. Utolsó lépésként megvizsgálja a metódus, hogy mennyi az értéke. A kódból egyértelmű, hogy ha van találat, akkor mindig nő egyel az értéke. Ha nulla marad, mert a metódus elején ki van nullázva, akkor nem volt találat, és kiírja hogy a „A keresés nem járt eredménnyel”. Ezzel az utolsó mozzanattal zárul a kereső metódus működése. A *ResultBox* azt a szövegdobozt jelöli, ahova a keresés eredménye kiíródik. A kódban azt a szövegdobozt, amibe a keresendő szöveget lehet beírni, a *searchBox*n nevet adtam.

Mindkét szövegdobozt igyekeztem azonos magasságba elhelyezni panelen, igyekeztem folyamatosan az esztétikát szem előtt tartani. A dobozok *Border* tulajdonságát *FixedSingle*re állítottam. Véleményem szerint jól mutat ebben a formában a szótár paneljén, a szépségérzetet nem rontja, de eltér egy picit az eddig megszokottól. Némely esetben szerintem ez a felhasználó véleményét javítja az adott programmal kapcsolatban. Ezt saját tapasztalat alapján gondolom így. Nekem általában tetszeni szoktak az olyan alkalmazások, amelyeken belül kisebb eltérő megoldásokat találok. Természetesen ez csak az én saját véleményem.

A szótári rész működés közben

3.7 A teszteket tartalmazó menüpont

Az oktatóprogram egyik legösszetettebb része, mind a külsőt, mind a működést tekintve. Alapvetően két típusú teszt található a programban. Minden témakörhöz van egy tíz kérdést tartalmazó teszt, kérdéseként három válaszlehetőséggel. Ezeket „statikus” teszteknek neveztem el. Ezen felül van egy úgynevezett „Komplett, minden témát átölelő teszt”. Ezt a tesztet „dinamikusnak” kereszteltem működése alapján.

3.7.1 A „statikus” tesztek

Először is statikus jelző megmagyarázásával kezdem. Azért kapta ezt a fajta elnevezést, mert a kérdések és a hozzájuk kapcsolódó válaszlehetőségek nem változnak. Azért döntöttem ezen megoldás mellett a témakörök tekintetében, mert egy idő után, ha mást nem, de legalább az adott kérdésre a választ meg fogja tanulni a felhasználó. Ne feledjük, az elsődleges cél továbbra is az új ismeret elsajátítása. Erre az oktatóprogram csak egy eszköz lehet, ami segítséget nyújt.

A teszteknek legfőképp az a célja, hogy az elsajátított tudást mérje, egy visszajelzést adjon az aktuális tudásszintről az érintett témakörben. Épp ezért, tartós gyakorlás mellett tíz kérdésre legalább tudni fogja a választ. Igaz ez nem a teljes tudást fogja tükrözni. Ha egy tanuló otthon használja a programot, a saját lelkiismeretére van bízva, hogy mennyi időt fordít a tananyagegységek tanulmányozására, és mennyit a tesztek megoldására. Ahogy mi tanárok mondani szoktuk: „saját magukat csapják be”. Ezek alapján nem gondolom, hogy hibás döntés volt a témakörökhöz kapcsolódóan állandó kérdéseket-válaszlehetőségeket tartalmazó teszteket készíteni.

Amikor a „Tesztek” gombra kattintunk, megjelenik előttünk a teszteket tartalmazó ablak, amely több részegységeket tartalmaz. Elsőként egy legördülő menüt (*ComboBox*) találunk. Ebben a legördülő menüben találhatók a témakörök címei. Természetesen a hozzájuk kapcsolódó teszteket jelenítik meg. A legördülő menü mellett találjuk az eredményt jelző dobozt. Ebben százalékos formában jelenik meg a tanuló teljesítménye a jó válaszok tekintetében.

A már említett tíz kérdés egyszerre jelenik meg az adott témakör kiválasztása után a hozzájuk kapcsolódó válaszlehetőségekkel együtt. A kérdéseket címkékkel jelenítettem meg.

A válaszlehetőségeket a már korábban említett rádiógombokkal oldottam meg. Ha összeszámoljuk, akkor számszerűen harminc rádiógomb jelenik meg egy időben egy panelen. Ezzel az a probléma, hogy ha futáskor ki akarjuk jelölni a helyes válaszokat a panelen, akkor a harminc lehetőség között „ugrálni” fog a kiadott válaszunk. Ilyenkor nem kell mást tenni, mint az összetartozó válaszlehetőségeket egy csoportba kell foglalni. Ezt megtehetjük panelekkel (én ezt alkalmaztam), vagy csoportképzéssel (*GroupBox*). Innen már nem nehéz kitalálni, lesz tíz darab kisméretű panelünk, egyenként három darab rádiógombbal.

A fent említett vázat még tervezőnézetben kialakítottam, de szövegi részt nem rendeltem hozzá. Ezek egy nagy panelen vannak, és nem láthatók. Akkor válnak láthatóvá, amikor kiválasztunk egyet a legördülő menüből, de ekkor már a hozzájuk rendelt szöveggel.

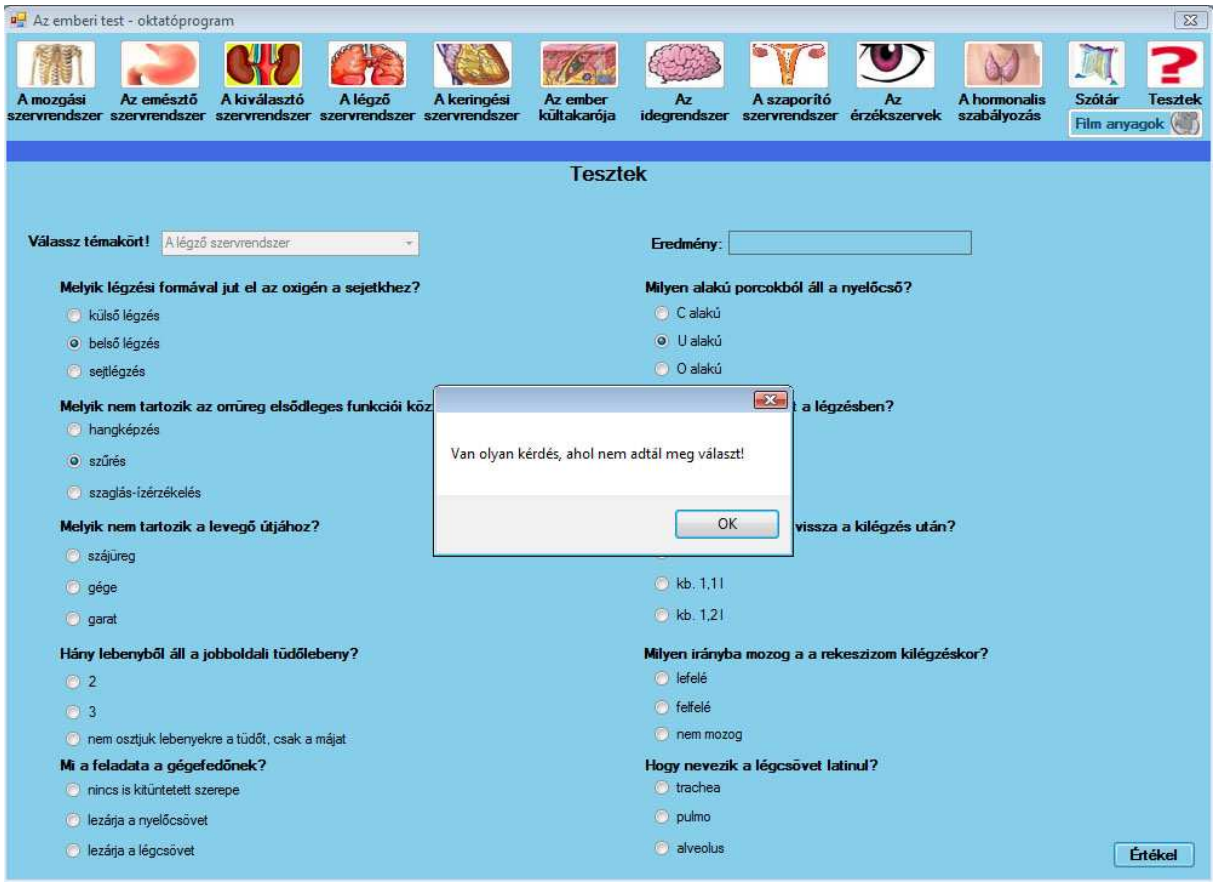
A kérdéseket és válaszokat nem tettem állományokba, hanem a forráskódban helyeztem el. Azért írtam meg így a programot, hogy ne lehessen a megnézni azokat. Amikor megtörténik egy téma kiválasztása és megjelenik a teszt, ezzel egy időben az oktatóprogram menüje elérhetetlenné válik. Ezzel is igyekeztem azt elkerülni, hogy meg lehessen keresni a kérdésekre a választ. Erre írtam egy olyan eljárást, amely a gombok *Enabled* tulajdonságát hamisra állítja, azaz inaktívvá válnak. A kódban *disableButtons()* néven jelenik meg. Természetesen létezik vele ellentétes működésű eljárás is, amely az *enableButtons()* nevet viseli, és a teszt kiértékelését követően lép működésbe. A legördülő menü is letiltására kerül egy teszt működése közben, majd újra elérhetővé válik a teszt kiértékelését követően, amit a *ComboBox Enabled* tulajdonság hamis-igaz állítgatásával értem el. Az eredményt tartalmazó szövegdoboz írásvédett. Ezekkel a beállításokkal igyekeztem „bolondbiztossá” tenni a program teszt részét.

A kérdések és a válaszok szöveget témakörönként manuálisan állítom be metódusokkal. Minden egyes témakörhöz tartozik egy ilyen eljárás, ami ezt végzi el. Az ellenőrzést is külön függvények végzik témakörönként. Minden egyes esetben megadtam hogy mely válaszok a jók, és a függvény azt figyeli, hogy azok lettek-e bejelölve. Ha egyezik a bejelölés, akkor megnöveli az ott lévő segédváltozó értékét. Egyértelmű, hogy ez a segédváltozó a helyes válaszok számát jelöli. Mivel a teszt tíz kérdést tartalmaz, a helyes válaszok számát elosztom tízzel, majd szorzom százzal, majd az így kapott értéket írom ki az eredményt jelző szövegdobozba.

A teszt működése közben figyel arra is, hogy minden válasz meg volt-e jelölve. Ha van olyan kérdés, amelyiknél nincs bejelölve válasz, akkor egy felugró ablakban figyelmeztet

erre a következő felirattal: „Van olyan kérdés, ahol nem adtál meg választ!”. Ez a függvény egy logikai igazzal tér vissza, ha fent említett eset következik be, ellenkező esetben hamissal. Ezt a függvényt is saját magam írtam, *hasMoreAnswer()* néven található meg a program forráskódjában.

Amikor egy teszt kiértékelése megtörténik, az összes szöveget tartalmazó elemet üressé teszem, hogy ne legyen benne maradó „szemét”, majd láthatatlanná állítom. Ezt azért írtam meg így, hogy ne lehessen a kérdéseket és a válaszokat leírni. Igaz, hogy a futás közben ezt megteheti, de a teszt kiértékelését követően tud következtetni a helyes eredményre, amit viszont szertettem volna meghagyni.



A „statikus” teszt működés közben

3.7.2 A „dinamikus” teszt

Az elnevezés arra utal, hogy a kérdések változó sorrendben érkeznek a program felületére. Ez a fajta teszt a „Komplett, minden témát átölelő teszt” menüponttal érhető el. Ez a teszt alkalmas arra, hogy mérje a szerzett tudást az összes témakörrel kapcsolatban.

A működése alapvetően tér el a „statikus” tesztekhez mérve. Egyszerre csak egy kérdés jelenik meg a három válaszlehetőséggel. A kérdések változó sorrendben jelennek meg. Egy tesztsorozatban tizenöt kérdésre kell válaszolni. A program minden egyes kérdésnél figyeli, hogy a felhasználó jelölt-e be választ. Ha ezt elmulasztotta, akkor egy felugró ablak figyelmezteti erre a következő szöveggel: „Nem jelöltél be választ!”. A teszt kiválasztásakor a másik teszttípushoz hasonlóan a menügombok és a legördülő menü elérhetetlenné válnak, majd teszt befejeztével „aktivizálódnak”.

A feladatmegoldást a „Teszt indítása” gomb lenyomásával lehet elkezdeni, a válaszadást követően a „Következő kérdés” gombbal lehet kérni az új kérdést. Az utolsó kérdést követően megjelenik az eredmény az eredményt kiíró dobozban, ugyanúgy értékelve a teljesítményt, ahogy azt a témaköröknél leírtam, csak tizenöt kérdésre levetítve.

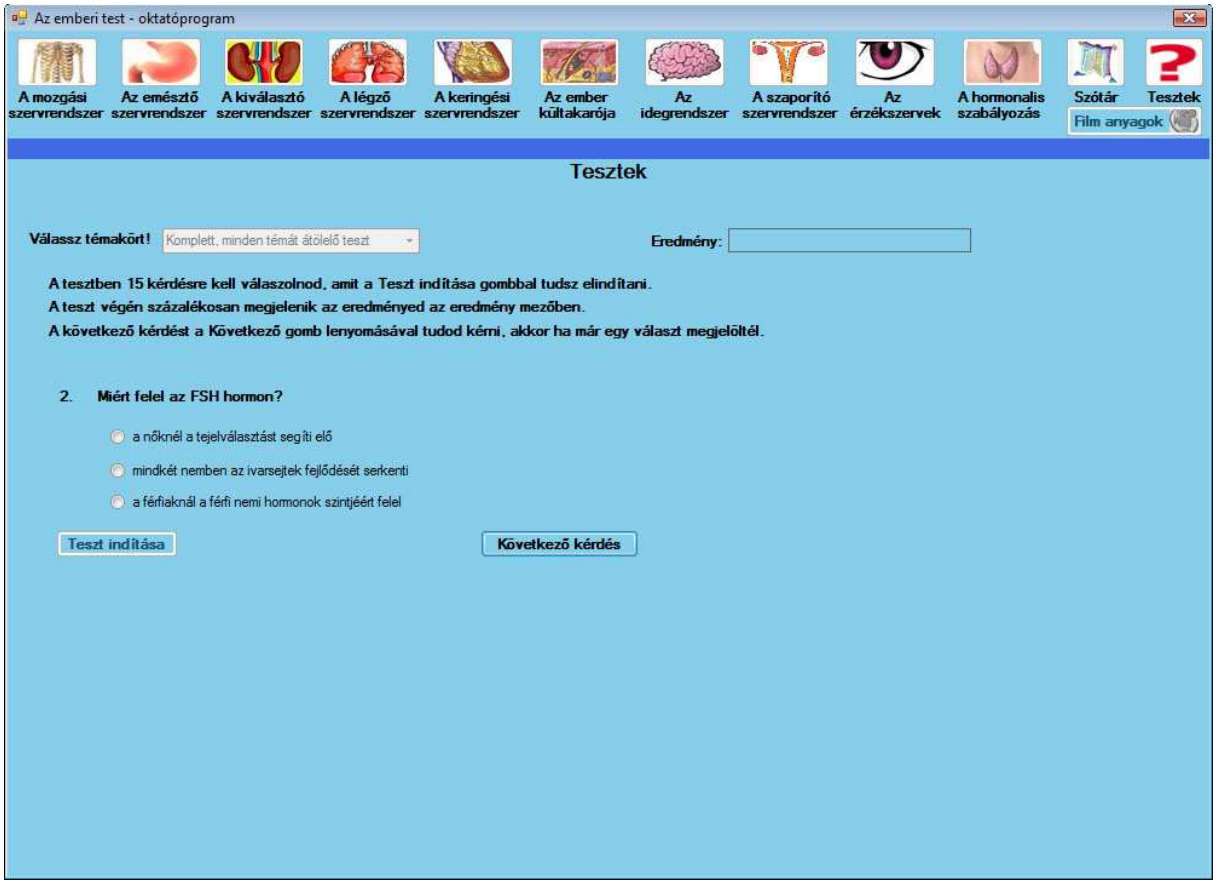
A tizenöt kérdés egy ötvenelemű kérdésbázisból kerül kisorsolásra véletlenszerűen. Erre a *Random*ot használtam. Felvettem öt darab ötvenelemű tömböt. Az egyikbe a kérdéseket vettem fel, másik háromba a kérdésekhez tartozó válaszokat, majd az ötödikbe a jó válaszokat. Itt is az indexek egyezőségét használtam ki. Például az 5. indexen lévő kérdéshez a másik három tömb 5. indexen lévő válaszok passzolnak. A helyes válaszok tömb 5. indexén a helyes válasz szerepel, amely megegyezik a három választ tartalmazó tömb valamelyikével. Ennek majd az ellenőrzéskor lesz szerepe.

Felvettem egy tizenöt elemű tömböt, amely egész számokat tartalmazhat. Ebbe a tizenöt elemű tömbbe sorsolok ki számokat 1 és 50 között. Ha a szigorú tömbindexelést nézzük, akkor 0 és 49 között a határokat is beleértve. A számokat úgy generálom, hogy a sorsoláskor már figyelem azt, hogy az addig létrehozottak között ne legyen ismétlődés, azaz amikor generálódik egy új szám, akkor végigszaladok a tömbön, hogy az már létezik-e. Ha nem, akkor letárolom, ha igen akkor generálok egy újat, majd azt megint ellenőrzöm. Így elkerülöm, hogy egy kérdés többször előfordulhasson. Az így kapott számok lesznek a kérdéseknek a tömbben lévő indexei. Azok a kérdések jelennek meg majd, amelyeket kisorsoltam.

Az ellenőrzéskor nem kell mást tenni, mint megnézni, hogy a bejelölt rádiógomb szövege (a *Text* komponens által hordozott szöveg) megegyezik-e az adott kérdés indexéhez tartozó, helyes válaszokat tartalmazó tömb azon indexén lévő szöveggel. Ha igen, akkor megnövelem a jó válaszokat számoló segédváltozó értékét. A teszt végén pedig kiíratom százalékos formában.

Mindkét teszttípusnál ügyeltem arra, ha egyáltalán nincs jó válasz, akkor kiírom a 0 %-ot, és nem osztok nullával. Mint tudjuk ez kivételt okozna futás közben.

Ez a fajta teszt már jobban alkalmas a tényleges tudás mérésére, viszont a teljes anyagra vonatkozó tudást feltételezi.



A komplett teszt működés közben

3.8 A Film anyagok menüpont

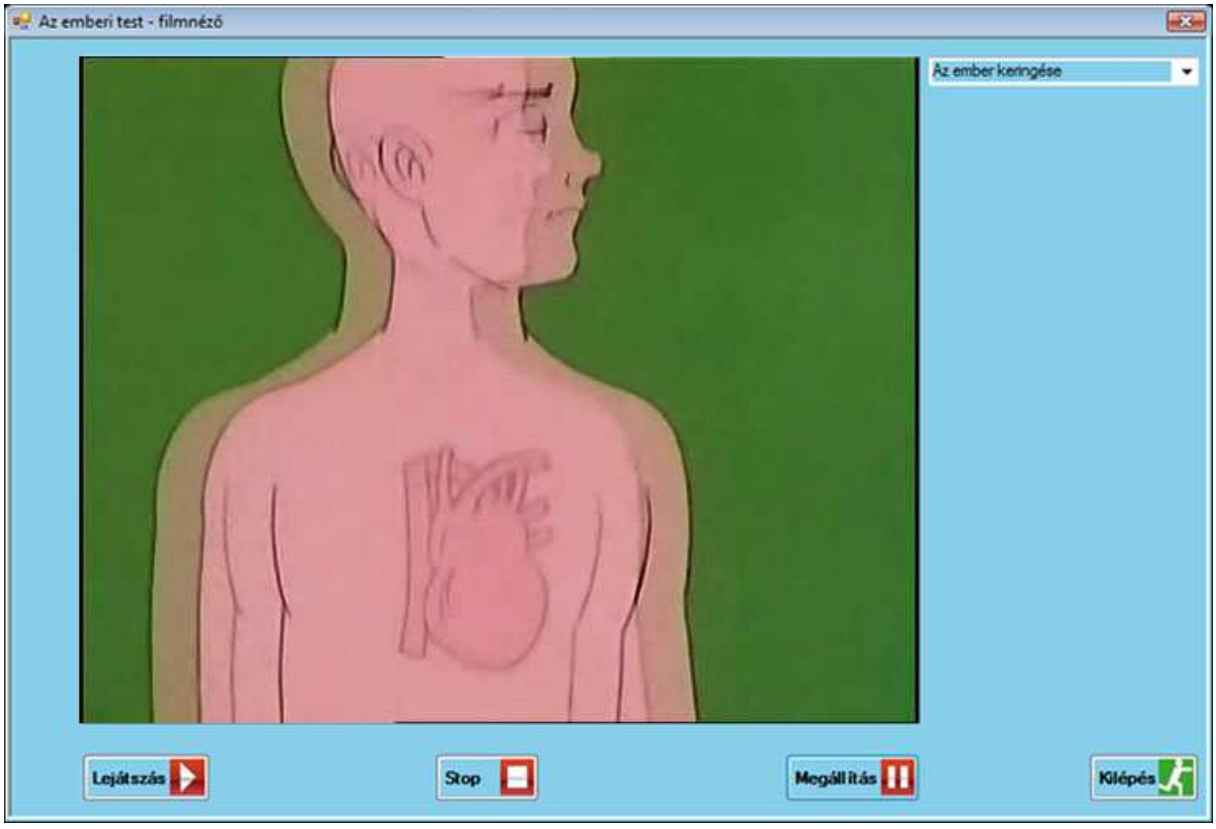
A program készítése közben gondoltam arra, hogy jó lenne valamilyen multimédiás ponttal bővíteni azt. Ekkor jutott eszembe, hogy volt régen a tv-ben az Egyszer volt hol nem volt, az élet c. francia rajzfilmsorozat, amely tökéletesen mutatta részenként az emberi szervezet egy-egy szervrendszerét, szervét. Nagyon jó oktatófilmeknek találom nemcsak kicsiknek, hanem akár a felnőtteknek is. Úgy gondoltam, hogy felhasználom oktatásra. Annyira jónak találtam, hogy nem is igazán tudtam kiragadni részeket az egyes epizódokból, így inkább három komplett részt tettem be a programba.

Nem akartam a Windows Media Player programját használni, így gondoltam készítek egy egyszerű saját filmlejátszót.

Ezt egy új alkalmazásként írtam meg, különálló futtatható állományként. A program felületén találunk alapvető vezérlő gombokat, mint a „Lejátszás”, a „Stop”, a „Megállítás” és a „Kilépés”. A videót vezérlő gombokat alapesetben letiltottam, mivel induláskor még semmilyen videó nincs betöltve. A felületen találunk egy legördülő menüt, amiből a videókat tudjuk kiválasztani. Amikor megtörténik a kiválasztás, a film betöltésre kerül, majd a vezérlőgombok aktívvá válnak, és el lehet indítani a lejátszást.

A film képanyaga középen egy panelban jelenik meg. Ezt kód szinten kellett megadni, hogy a képanyag tulajdonosa ez a panel (owner), ahol annak meg kell jelenni. A videólejátszó DirectX alapú, amire mindenképp szükség van annak működéséhez. Konkrétan a *Microsoft.DirectX.AudioVideoPlayback* által definiált specifikációkat használja. Az ehhez szükséges dll állományt is hozzá kellett rendelni a projekthez.

Csúszkát szándékosan nem rendeltem a lejátszóhoz. Egyik szempontból hátrány, másik szempontból előny. Engem az vezérelt, ha órai keretben akarjuk használni, akkor a gyerekek ne tekergessenek bele a filmbe. Viszont már hátrány, ha a tanár akarna kiragadni részeket. Ez a hiányosság orvosolható.



A lejátszó program felülete (a képkockák mozgása okozza a homályos képet)

4. A program írása közben adódott problémák és megoldásuk

A program megírása során több akadályba is ütköztem. Ezeket kisebb nagyobb sikerrel oldottam meg, hárítottam el.

A C# programozási nyelvvel a szakdolgozat elkészítésekor kezdtem el ismerkedni. A grafikus felhasználó felület készítésében nem nagyon voltam járatos, nem tudtam, hogy kell azt életre kelteni, mivel csak konzolos programokat készítettem eddig. Ahogy elkezdtem egyre mélyebbre jutni fokozatosan értettem meg azt, ahhoz hogy a felületen történések legyenek, valaminek ki kell azt váltani, legyen az akár csak egy kattintás. Kezdetben ez nagyon furcsa volt, de megszoktam, mondhatni hamar.

Kezdetben, amikor írtam a programot az volt a feltűnő, hogy a mindig a legfrissebben elkészült panel látszik induláskor. Ekkor esett le, hogy ezek *Visible* tulajdonságát alapértelmezettként hamisra kell állítani, így nem látszanak induláskor, csak majd amikor valamilyen esemény ezt váltja ki.

A következő problémával programírás közben szembesültem. Semmilyen program nem futott a háttérben, nem futtattam az oktatóprogramot sem, egyszerűen a Visual Studio „Out of memory” hibaüzenettel leállt. Mondanom sem kell, már pár panel készen volt, a képek is be voltak importálva, pozícionálva. Nem tudtam menteni az utolsó lépéseket. Ezek a tünetek, csak laptopon jelentkeztek. Csodálkoztam, mert 2GB memóriával ez olyan hihetetlennek tűnt, természetesen az itthoni asztali gépen semmi gond nem volt. Az igaz, hogy abban 4GB RAM van, de ekkor még mindig nem hittem a kevés memóriában és arra gyanakodtam, hogy a notebook memóriája hibás. Kis idő elteltével egy asztali gépen, ami szintén 2GB memóriával rendelkezett szintén előfordult a hiba. Ekkor zártam ki a hardware hibát. Ténylegesen a programírás felé fordultam, hogy mit ronthattam el. Kisebb gondolkodás után döbbentem rá, hogy a nagyméretű képek okozzák a problémát. Mint már írtam, minden kép tervező nézetben lett elhelyezve és importálva. Ezt a sok adatmennyiséget a Visual Studio nem tolerálta. Annyit kellett volna változtatnom a kódon, hogy a képeket nem importálom tervező nézetben, hanem futáskor töltöm be megadva a kép elérési útját. Ezzel rengeteg memóriát spórolhattam volna meg. Mivel már a tananyagegységek feldolgozása 80%-os volt, már nem kezdtem meg a program ezen részének az átalakítását. Viszont, ha nincs 4GB memória az itthoni gépben, akkor minden bizonnyal erre kényszerülök.

A legérdekesebb buktatót a lejátszó programmal éltem át. A fejlesztést párhuzamosan 64-bites Microsoft Windows Vista, és 32-bites Microsoft Windows XP rendszereken végeztem, mikor melyik gép volt a közelemben, az aktuális forráskódot pendrive-on hordoztam. Az itthoni asztali gépen, Vista rendszer alatt készítettem a lejátszót. Lefordítottam, de nem működött. A program indításkor a program azonnal leállt. Kódszinten kerestem a hibát, de nem akadtam rá. Végső elkeseredésemben egy barátomnak elküldtem a forrást, hogy segítsen mi a hiba. Az volt a legérthetetlenebb, hogy hiba és figyelmeztetés nélkül fordult le a kód. Átnézte, majd jelezte, hogy nincs benne hiba, neki tökéletesen megy. Még értetlenebbül áltam a hiba előtt.

Átjött hozzám, majd mutatta, hogy az ő laptopján tökéletesen megy. Megint átnézte, de még mindig nem látott benne hibát. Kérte, hogy nézzük meg, az én noteszgépemen megy-e. Természetesen ment, ráadásul az a kód, amelyet az asztali gépemről vittünk át a notebookra. Arra a következtetésre jutottunk, hogy az én Windowsom bolondult meg. Az ő gépén szintén Vista 64bit volt fent, az én laptopomon 32-es XP. Két este Windows Vistát telepítettem. Újraraktam mindent, update-t futtattam a Windowson is. De a hiba csak nem múlt el. Ekkor eszembe jutott, hogy ha a Visual Studioban futtatom a kódot, akkor lehet debugot vagy nyomkövetést végezni. Elkezdtem futtatni a kódot, ami persze azonnal leállt. A Visual Studio kidobott egy úgynevezett *BadImageFormatException*t. Ennek nagyon megörültem, nem is értettem, hogy miért nem ezzel kezdtem. Felmentem az internetre, a google-be beírtam a fenti kivétel nevét és vártam a megoldást, de nem volt. Kb. másfél órán át bújtam az internetet, de nem akadtam semmire.

Fordítgattam, beleirogattam, de semmire nem jutottam, folyamatosan a fenti kivétellel leállt a lejátszó program. Egyszer arra lettem figyelmes, hogy a Microsoft Visual Studio felkínálja, hogy keressen-e ő megoldást a problémára. Nem szoktam hinni az ilyen megoldásokban, más ötletem nem volt, engedélyeztem. Olyan másfél percen belül behozott egy angol szakmai fórumot. De nem a böngészőben, hanem saját ablakban, vélhetően valamilyen Microsoft „belső”, csak a Visual Studiot használók körébe elérhető fórumot. Ott egy felhasználó szintén ugyan ezt írta le, és hogy nem talált ő sem megoldást. Az ő kérdésére kapott válasz volt az én válaszom is. Egyszerűen annyit kellett átállítani a fejlesztői környezetben, hogy fordításkor x86-os platformra fordítson. Ezzel megoldódott a probléma, és fut hiba nélkül a lejátszó program.

A program megírása közben más emlékezetes hibába nem futottam. A fentiek pedig jó példák arra, hogy mikre kell nagyon figyelni egy program írása közben.

5. Összefoglalás

Az oktatóprogram elkészítésével egy jól használható segédeszköz előállítása volt a cél. Tartalmi szempontból egy kicsit hiányosnak érzem, mert az egyes szervrendszereknek inkább az anatómiai bemutatása valósult meg, azok élettani bemutatása nem. Mivel elég széles kört igyekeztem lefedni, úgy érzem, ez nem csökkenti a használhatóságát. Egy újabb célt vet fel bennem, mégpedig azt, hogy készítsem el ennek az oktatóprogramnak a folytatását, ha úgy tetszik a második részét, amelyben inkább a most kevésbé kifejtett egységek kapjanak nagyobb hangsúlyt.

A program megírása során szakmai előrelépésre tettem szert. Megismerkedtem egy eddig számomra idegen programozási nyelvvel. Ez a nyelv egyáltalán nem áll messze tőlem, úgy érzem könnyen el tudnék benne merülni. Számomra sokkal érthetőbb és használhatóbb, mint a Java vagy a Delphi. Fejlődött az algoritmikus gondolkodásom, a problémamegoldó képességem.

Az egyetemen tanult programozói ismeretek egy része most nyert igazi világosságot. A Programozás 1 gyakorlatain rengeteg függvényt, eljárást kellett megírni, amiknek akkor nem sok jelentőséget tulajdonítottam, de most a szakdolgozat megírása során nagyon sokat tisztultak. Valójában előre vetítette azt, hogy az objektumorientált programozás során rengeteg eljárást és függvényt kell használni, amelyek csak segítik a munkát. Nagyon jól tudom, tudom hogy az OO világ nemcsak ebből áll.

Eddig nem szerettem programokat írni, de most nagyot változott a szemléletem. A félelmeim elmúltak, úgy is mondhatnám, hogy tudásszomjjá alakult át. Azzal is tisztában vagyok ez az oktatóprogram a .NET programozás felszínét kapargatja a technológia kihasználásának szempontjából. De mint mondottam készen állok ezen irányban új ismeretek befogadására.

Az oktatóprogram teszt részében fejlesztési lehetőségként jelölném meg a több válasz megadásának lehetőségét, azaz a jelölőnégyzetek (*CheckBox*) használatát. A második rész esetében animációk készítése, amelyek az egyes szervek, szervrendszerek működését mutatják be. Ezzel ténylegesen az élettanra fektetve a hangsúlyt. Videóanyag használatakor a lejátszó programba a pozíciós csúszka beépítése.

Összességében úgy gondolom, hogy a program felhasználóbarát, szemléletes, jól használható. Bár egy programról sosem lehet elmondani, hogy kész van teljesen, mert mindig ott vannak háttérben húzódó további igények, apróbb finomítások, fejlesztési lehetőségek.

6. Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani konzulensemnek Dr. Rutkovszky Edénének a dolgozat elkészítéséhez nyújtott segítségéért, tanácsaiért és türelméért. Folyamatos biztatása állandó támasz volt a helyes út megtalálásához.

7. Irodalomjegyzék

Dr. Tóth László – Pszichológia a tanításban

Pedellus Tankönyvkiadó Kft. 2005

Albert István szerkesztésében – A .NET Framework és programozása

Szak Kiadó Kft. 2004

Demeter M. Ibolya – Visual Studio 2005 & 2008

Panem 2008

Nagy István Dr. Nyilas Károly – Az ember biológiája és egészségtana

Nemzeti Tankönyvkiadó 2003

Peter Abrahams – Az emberi test atlasza

Gabo Kiadó 2003

Online irodalom:

Reiter István – C# 2009

Elérhetőség: http://people.inf.elte.hu/reiter_i/sharp.html
http://devportal.hu/groups/fejlesztk/media/p/1122.aspx

Dr. Kovács Emőd Hernyák Zoltán Radványi Tibor Király Roland –

A C# programozási nyelv a felsőoktatásban Programozás tankönyv 2009

Elérhetőség: http://csharptk.ektf.hu/

Felhasznált internetes linkek:

http://www.enc.hu/1enciklopedia/fogalmi/ped/elern.htm

http://tmt.omikk.bme.hu/show_news.html?id=4255&issue_id=467

http://hu.wikipedia.org/wiki/C_Sharp