Debreceni Egyetem
Informatikai Kar

Oktatóprogram a matematika tanításához

Témavezető:
Nyakóné dr. Juhász Katalin
tudományos főmunkatárs

Készítette:
Kontra Zsolt
programtervező informatikus (tanári szakirány) – matematika minor

Debrecen
2010

Tartalomjegyzék

Bevezetés

A szakdolgozatom egy, a matematika középiskolai oktatását támogató számítógépes alkalmazás megvalósítását mutatja be. Ez egy meglehetősen összetett és sokrétű feladat, s a megoldása közben számos igényre és szükségletre oda kellett figyelni. A célom azonban nem az volt, hogy egy minden részletre kiterjedő programot alkossak, hanem hogy egy olyan alkalmazás szülessen, ami egyszerűen és hatékonyan oldja meg a kívánt feladatot, ugyanakkor magában rejti a később felmerülő vagy időközben megváltozó igényekhez való igazítás lehetőségét. Egy ilyen program létjogosultsága mellett úgy gondolom, nem szükséges különösebbképpen érvelni, hiszen az elmúlt két évtizedben a személyi számítógépek jelentősen elterjedtek az oktatás területén is, így a jövőben mindenképpen ajánlatos az oktatómunka során kihasználni az ezen a területen kínálkozó lehetőségeket.

Az oktatóprogramnak három fő feladatot kell megoldania: egyrészt nyilván kell tartania, és kezelnie kell a diákok adatait, akik részt vesznek az adott oktatásban. Másodsorban el kell látnia a tanulókat a megfelelő tananyaggal, ezáltal biztosítva a tananyagok megfelelő elsajátításának lehetőségét. Harmadsorban pedig lehetővé kell tennie az anyag elsajátításának ellenőrzését.

A tanulók, mint felhasználók kezelését, egy felhasználónév-jelszó páros segítségével képes a program megoldani. Felhasználók felvétele a tanár által lesz megvalósítható, hiszen csak bizonyos zárt közösségeknek kell tudni elérni a rendszert, így nem indokolt a bárki számára hozzáférhető, nyílt regisztráció lehetősége.

A tananyagok szemléletesek, jellemzően ábrák és a hozzájuk tartozó leírás, illetve magyarázat alkotja őket. Mivel a középiskolai matematika anyag meglehetősen bő témakör, így nem célom azt teljes mértékben lefedő ismeretanyagokkal feltölteni a kifejlesztett alkalmazást, de lehetőséget hagyok arra, hogy a már bekerült tananyagokat bővíteni lehessen, új témaköröket, fejezeteket lehessen bevinni a már meglévők mellé.

A számonkérés megvalósítása számítógépes környezetben jellemzően teszt jellegű kérdésekkel történik, hiszen a válaszok gépi feldolgozása egyéb esetekben igen nehézkessé válna.

Mivel egyidejűleg sok felhasználó jelenlétének a lekezelésére van szükség, akiknek nagyrészt ugyanazon anyagokhoz kell hozzáférniük, így kézenfekvő megoldásként kínálkozik, hogy az alkalmazás képes legyen ezt hálózatos megoldások keretén belül megvalósítani, vagyis szerver-kliens kapcsolatként értelmezzük a valós életben tanár-diák kapcsolatként megjelenő összerendelési viszonyt.

Az előbbiekben nagyvonalakban felvázolt oktatóprogram megvalósítása Java programozási nyelven történt, grafikus felülettel, ezáltal a korunk követelményeinek megfelelően felhasználóbarát programot alkothattam. A grafikus elemek használata ugyanakkor a programozói munkát is nagymértékben segíti, hiszen néhány komponens „összekattintgatásával" gyakorlatilag megvalósítható egy-egy képernyő elkészítése.

Mivel az alkalmazás nem kifejezetten olyan személyeknek készült, akik az informatika területén kimondottan járatosak, ezért lényeges szempont volt a tervezés és megvalósítás folyamán, hogy a lehetőségekhez képest egyszerű legyen mind a program működtetése, mind pedig a tananyagok bővítése. Ez utóbbi különösen fontos szempont volt, hiszen könnyű látni azt, hogy muszáj biztosítani a majdani felhasználók számára, hogy a tananyagot saját maguk által összeállított formában vihessék át ebbe a környezetbe.

Amennyiben ez túlzottan bonyolult vagy időigényes lenne, az bizonyára sokakat eltántorítana attól, hogy kipróbálják a hagyományos képzést felváltani próbáló újszerű lehetőségeket, Éppen ezért az alkalmazás megtervezése során külön szempont volt az, hogy az alapjául szolgáló tananyag könnyen összeállítható és a programba importálható legyen anélkül, hogy az azt végzőknek ez bonyolult feladatokat, vagy hosszas munkát jelentene. Természetesen ugyanez vonatkozik a diákok tudását felmérő feladatsorokra is.

Arra, hogy a fentiekben megfogalmazott feladatoknak és célkitűzéseknek mennyire sikerült megfelelni, az alkalmazás elkészülte után tervezett éles kipróbáláson mindenképpen fény derül. Ekkor egy középiskolai délutáni fakultáció jellegű foglalkozás keretein belül kipróbáltam valós környezetben azt, hogy mennyire használható a létrehozott program arra a célra, amire tervezve lett. Ez a próba lehetőséget nyújtott arra is, hogy további fejlesztési lehetőségek, esetleges kijavítandó hiányosságok kerüljenek a felszínre.

Összefoglalva az eddigieket elmondhatjuk tehát, hogy célom egy viszonylag egyszerű, ámde jól használható és rugalmasan bővíthető oktatási segédeszköz megvalósítása, amelynek segítségével a matematika középiskolai oktatását számítógépes környezetben lehet majd végezni.

Az alkalmazás fejlesztésének lépései

Célok és feladatok meghatározása

Az alkalmazás fejlesztésének kezdeti fázisában meg kell történnie azon célok és feladatok meghatározásának, amiket el szeretnénk érni, illetve meg szeretnénk valósítani. Egy rendszer esetében a vele szemben támasztott követelmények alapjában véve kétfélék lehetnek: új szolgáltatáson, funkción alapuló követelmények, illetve a jelenlegi rendszer hibáin alapuló követelmények. Bár kezdetben a követelményeket elegendő nagyvonalakban meghatározni, minden lehetőséget meg kell ragadni annak érdekében, hogy ennek ellenére a követelmények oly módon kerüljenek megfogalmazásra, ami mégis számszerűsíthetővé, mérhetővé teszi azt bizonyos mértékben.

Mivel jelen esetünkben nem beszélhetünk előző rendszerről, korábbi szoftverről, így nem tudunk mérvadó összehasonlításokat tenni két rendszer között, így az előbbiekben említett két követelménycsoport közül a második nem jön számításba. Összehasonlítási alapként a normális tanrend alapjául szolgáló osztálytermi oktatás jöhet szóba, aminél az új, számítógépes lehetőségeket kiaknázó szoftvernek mindenképpen gyorsabb, hatékonyabb és kényelmesebb megoldásokat kell kínálnia. Ugyanakkor annak érdekében, hogy az alkalmazás elérje legfontosabb célját, vagyis hogy effektíven tudjon hatni az oktatómunkára, mindenképpen szükséges, hogy egyúttal egyszerű, könnyen áttekinthető is legyen, hogy ne a program megismerésére, használatára, hanem a tényleges tananyagok elsajátítására tudják az azt használók fordítani az energiájukat.

Az alkalmazás igyekeztem úgy megtervezni, hogy kihasználja a számítógépes hálózatok nyújtotta előnyöket, ennek megfelelően külön szerver- és kliensprogramból épül fel, ahol előbbi jellemzően a tanárt, utóbbi a diákokat testesíti meg. A szerver esetében feltételezzük, hogy ahhoz csak a tanár férhet hozzá, illetve ő működtetheti, ezért eltekintek az azon tárolt adatok védelmétől, titkosításától. Mivel azonban az alkalmazás nem hivatott a papír alapú értékelési formákat (osztálynapló, bizonyítvány, stb.) kiváltani, ezért ez nem okoz problémát az alkalmazás használhatóságának tekintetében.

A kliens programot a diákok fogják majd használni, így ennek a résznek egyszerűnek, könnyen áttekinthetőnek kell lennie. Ugyanakkor kerültem a személyes adatok,

információk kezelését ezen az oldalon, elkerülve ezzel az esetleges kellemetlenségeket. Azonosításhoz felhasználónév és jelszó párosa az, ami mindenképpen szükséges, ahogyan az mára már megszokott is hasonló esetekben. A diákok nem férnek hozzá személyes adatokhoz (a sajátjukhoz sem), azok kezelését a tanár végzi, ezzel is elkerülve a szándékos vagy véletlen adatmódosításokat, adatvesztést, illetve az azokkal való visszaélést.

Mint már említettem, az alkalmazás használatához minden diáknak szükséges az, hogy regisztrálva legyen a szerver oldalon. Ezt a tanár végezheti el, s mivel az alkalmazás amúgy is kisebb csoportok, esetleg egy osztály számára készült, így ez nem okoz túlságosan hosszadalmas adminisztratív munkát. Ennek megfelelően a szerver oldalon lehetőség van tehát az egyes diákok adatainak felvételére, elírás, illetve változások lehetőségére felkészülve pedig azok módosítására is. Mivel az alkalmazásnak nem célja az oktatással kapcsolatos adminisztratív jellegű feladatok megoldása is, ezért nem célszerű egy-egy diák esetében túlságosan sok adatot tárolni, elegendő csupán az egyes személyek beazonosításához feltétlenül szükséges információk elraktározása.

Ahogyan azt már a bevezetőben is említettem, rendkívül lényeges a tananyagok és feladatsorok könnyen összeállíthatósága, bővíthetősége. Mivel ezeket jellemzően a tanár feladata elkészíteni, így szintén a szerver oldalán kell megjelenni az oktatás e két fontos elemének. Ugyanakkor ezekhez már a diákoknak is hozzá kell tudni férni, vagyis számukra is biztosítani kell azt, hogy elérhessék azokat. Ezt vagy a szerveren található példány konkurens elérésével, vagy pedig az ott található példányok kliensekre történő lementésével oldhatjuk meg.

A tananyagok esetben nincs szükség semmiféle visszajelzésre a tanár irányába, ellenben a kérdéssorokat illetően szükséges lehet az egyes tanulók eredményeinek tárolása, azok megtekinthetőségének biztosítása a tanár számára. Ezért a tesztfeladatok esetében gondolkodni kell arról, hogy az azt kitöltők eredményei valamilyen formában visszajussanak a tanárhoz.

Összegezve az eddigieket elmondhatjuk, hogy a célom egy olyan szerver-kliens program előállítása, amely képes egy csoport adatainak kezelésére, tetszőleges mennyiségű és tartalmú tananyag, illetve az ahhoz kapcsolódó kérdések szolgáltatására, illetve az eredmények megállapítására. Mindezen célok elérése érdekében szükség volt

arra, hogy számos fejlesztési lépésen menjen keresztül a program, mire abból kész alkalmazás válhatott.

Adatgyűjtés

Ahhoz, hogy ki tudjuk alakítani a megvalósítandó szoftver lapjait és hozzávetőleges felépítési vázát, mindenképpen szükség van arra, hogy adatokat gyűjtsünk. Mivel semmiféle régi rendszert nem kell integrálnunk az elkészítésre váró alkalmazásba, ezért nem szükséges foglalkoznunk az iskolában végzett adminisztratív feladatokkal.

Viszont ahogyan azt már a feladatok megfogalmazásakor is előrevetítettem, arra ellenben szükség van, hogy a diákokat valamilyen módon kezelni tudjuk. Ehhez csupán néhány alapadatra van szükség, éppen csak annyira, ami elengedhetetlenül szükséges az osztály tanulóinak azonosításához. A kliensen való belépéshez szükség van egy felhasználói név, illetve jelszó tárolásához. Ezeket tehát mindenképpen kezelnünk kell. Annak érdekében, hogy ne kelljen a felhasználónevek kuszaságában eligazodni, vagyis ne kelljen a tanárnak egy esetlegesen semmitmondó felhasználónév alapján beazonosítani, hogy melyik diákjáról van szó, mindenképpen hasznos lenne a diákok nevét is eltárolni. Mivel azonban könnyen előfordulhat, hogy egy osztályban két, esetleg három azonos nevű diák is van, ezért szükség van még egy, az azonos nevűek megkülönböztetését segítő információra. Ez jellemzően lehet például az anyja neve, valamilyen azonosításra alkalmas szám (például a diákigazolvány száma) vagy éppen a születési idő. Mivel nekünk tökéletesen megfelel, ezért ez utóbbit választottam, vagyis a születési időt, hiszen elenyésző a valószínűsége annak, hogy egy osztályban két azonos nevű, ugyanazon a napon született tanuló legyen.

Egy másik lényeges terület a tananyagok kérdése. Mivel a matematikának már a középiskolai viszonylatban nézve is számtalan részterülete megjelenik, ezért szükség van egy olyan struktúra meghatározására, ami lehetővé teszi az algebrai ismeretanyagok közlését, a geometriai feladatok bemutatását, és a bonyolult képletek ismertetését is, nem beszélve sok más területről. Erre legalkalmasabbnak az a megoldás tűnik, amely szerint egy képernyőn egyszerre egy ábrát, illetve hozzá tartozóan egy magyarázatot jelenítünk meg. Az ábrán elhelyezhető egy geometriai szerkesztés ismertetése, bonyolult képletek, vagyis gyakorlatilag bármilyen matematikai ismeretanyag, míg a hozzá tartozó szöveg alkalmas a tanár anyaghoz fűzendő magyarázatának a továbbítására.

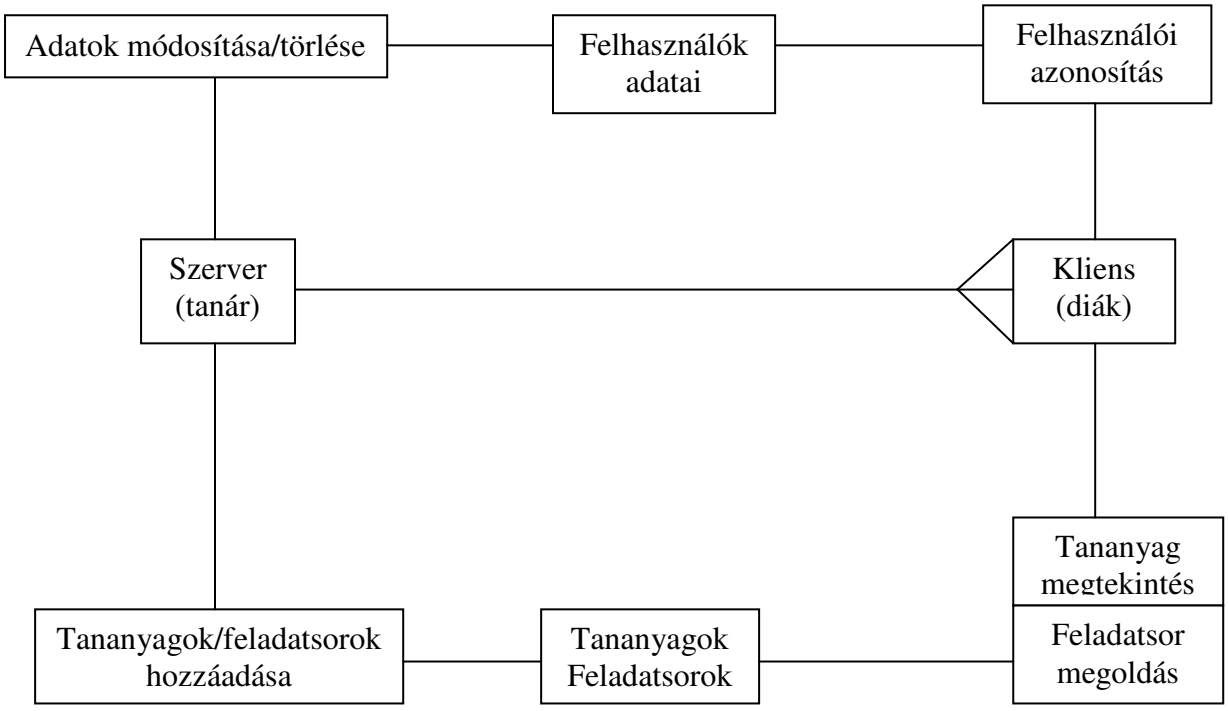
A tananyagok elsajátítását tesztjellegű kérdésekkel célszerű megoldani, hiszen ezt a számítógép által azonnal fel tudjuk dolgozni, és ki tudjuk értékelni. A kérdéssorok esetében tehát a kérdés szövegén kívül a négy válaszlehetőség, illetve a helyes válasz megadása tűnik mindenképpen szükségesnek.

A kliensprogramok esetében szükség van továbbá a szerver hálózati címének az ismeretére is. Mivel azonban ez olyan adat, amely gyakorta változik, ezért célszerű olyan formában kezelni, hogy könnyen módosítható legyen. Erre legalkalmasabbnak egy konfigurációs állomány tűnik, amely a szerverprogram IP-címének tárolására szolgál.

A rendszer logikai adatmodellje

A rendszer logikai adatmodellje segítséget nyújt abban, hogy áttekinthető képet kapjunk az elkészítendő rendszer hozzávetőleges felépítéséről. A logikai adatmodell segít átlátni a rendszerben résztvevő egyedek és a köztük fennálló kapcsolatok hálóját. A logikai adatmodell célja, hogy világosan érthető módon írja le az előbb említett egységeket, ugyanakkor az áttekinthetőség is igen lényeges, éppen ezért az ábrák alkalmazása mindenképpen ajánlott, amikor logikai adatmodell elkészítésébe kezdünk.

Ennek megfelelően először egy ábra segítségével megpróbálom összefoglalni azokat az információkat, amelyeket az eddigiekben megállapítottam, ezért alább látható az alkalmazások egyes részfeladatai közötti kapcsolatokat bemutató ábra.

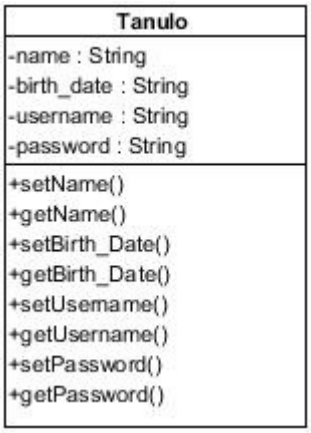


A fenti ábra segítségével viszonylag könnyen áttekinthetjük, hogy az alkalmazásunknak nagyjából milyen feladatai vannak, és az egyes részfeladatok hogyan kapcsolódnak az egészhez. A továbbiakban nincs más dolgunk, mint a már meghatározott részfeladatokat megoldani.

Fizikai terv

Ebben a fejezetben meghatározásra kerül az eltárolt adatok konkrét szerkezete. A rendszerben több olyan adat is szerepel, aminek meg kell határozni az eltárolási módját. Az alkalmazás működéséhez alapvetően háromféle adathalmaz kezelése és tárolása szükséges: a tanulók adatainak, a tananyagoknak, illetve a tananyagokhoz tartozó feladatsoroknak a kezelése, valamint tárolása az, ami elengedhetetlenül fontos.

Tanulók:



Ahogyan az a fenti osztálydiagramon látszik, a tanulóknak csupán azon adataira van szükségünk, amelyek alapján egyértelműen meg tudjuk őket különböztetni. Ezen adatok közé tartozik a felhasználónév (username). Ennek mindenképpen egyedinek kell majd lennie, hiszen a program a felhasználónév alapján tesz különbséget a klienst használó személyek között.

A név (name), illetve a születési idő (birth_date) adattagok a felhasználók, mint tanulók azonosítását könnyítik meg. Ugyanis mindenképpen szükséges információ a tanár számára, hogy be tudja azonosítani, hogy az egyes felhasználónevekhez mely diákok tartoznak.

A jelszó mezőre a tanulók verifikálása érdekében van szükség. Ugyanis nem lenne túl jó, ha bárki be tudna lépni egy-egy tanuló nevében, hiszen a feladatsorokon elért eredmények tárolásra kerülnek, így valaki más eredményei jelennének meg az adott felhasználó neve alatt, ami bonyodalmakhoz vezetne. Mivel a jelszót (szerencsés esetben) csak a felhasználó ismeri, így nem valószínű, hogy ilyen problémák felmerülnek.

A már előzőekben bemutatott osztálydiagramon látszik, hogy a Tanulo osztályban más adatot nem is tervezünk eltárolni, ennek megfelelően az előzőekben ismertetett négy adattagon kívül, illetve az azokhoz tartozó set és get metódusokon kívül másra nincs is szükség ezen osztályon belül.

Tananyag:

A tananyag tárolásának kérdése sarkalatos pont az egész alkalmazás kialakításának szempontjából. Az előre vele szemben megfogalmazott követelmények között a talán legfontosabb szempont a könnyű bővíthetőség, átalakíthatóság. Erre amiatt van nagy szükség, mert a tananyagok bizony gyakran változhatnak, új ismeretek kerülhetnek bele, esetleg egyes régebbi, nem annyira fontos ismeretek kerülhetnek belőle ki. Nem beszélve arról, hogy a középiskolai matematika tananyag meglehetősen bő egy ilyen alkalmazás számára, vagyis semmiképpen nem szeretném azt teljes mértékben felölelő témakörökkel én magam feltölteni a programot.

Mivel a középiskolai matematika meglehetősen változatos témakörökkel rendelkezik, ezért az is lényeges, hogy olyan struktúra kerüljön kiválasztásra, amely képes alkalmazkodni valamennyi igényhez, és lehetővé teszi nemcsak az algebrai részek, hanem a geometriai, trigonometriai és egyéb témakörök bemutatását is.

Ez mindenképpen megköveteli azt, hogy szövegek megjelenítésén kívül ábrák, táblázatok, geometriai szerkesztések lépéseinek a megjelenítése is lehetséges legyen. Ezért úgy döntöttem, hogy a tananyagok megtekintése során egy képernyőhöz egy kép (jellemzően valamilyen ábra, geometriai szerkesztés, táblázat vagy bonyolult képlet), illetve egy hozzá tartozó leírás (a tanár által fontosnak tartott hozzáfűznivaló) kerül megjelenítésre. A kép .jpg kiterjesztésű kell, hogy legyen, míg a szöveg .txt fájlokból kerül beolvasásra. Vagyis ha például egy 8 képernyőt átölelő, rövid tananyagot akarunk készíteni, akkor ahhoz 8 képfájl, és nyolc szöveges állomány fog tartozni.

Mivel hosszabb tananyagok esetében meglehetősen sok állomány válhat szükségessé a tananyag lefedéséhez, ezért az is kívánatos lenne, hogy a tananyagokat valamilyen módon együtt tudjuk kezelni, vagyis a kliensekre történő átmozgatásokkor egy állományként tudjunk vele bánni. Ezt kiválóan elérhetjük, ha a tananyagokhoz tartozó képeket és szöveges állományokat egy zip fájlban helyezzük el. Így a tananyagok

egyben, együtt kezelhetőek, és megnyitásukkor egy átmeneti könyvtárba kicsomagolva megtekinthető a tartalmuk.

A képeket és a hozzájuk tartozó szövegeket érdemes azonos névvel ellátni, biztosítva, hogy azok biztosan egymáshoz tartozóan kerüljenek megjelenítésre. Tovább növeli az áttekinthetőséget, ha az egyes állományok elnevezéséhez sorszámokat használunk (pl.: 01.jpg és 01.txt, 02.jpg és 02.txt, stb.). Ezáltal a diák sorrendjét is könnyen meg tudjuk határozni.

Ezen struktúrát használva gyakorlatilag mindenféle témakörhöz tartozó tananyag megjelenítésére lehetőség nyílik, hiszen elmenthetjük egy geometriai alkalmazásban készített szerkesztés képét .jpg formátumban, mint ahogyan képleteket és más dolgokat is. A .txt állomány pedig alkalmas arra, hogy a hozzáfűzött megjegyzést, magyarázatot abban tároljuk. Ráadásul ezáltal megvalósul a legfontosabb célkitűzés is, vagyis a könnyen kezelhetőség, a könnyű bővíthetőség. Hiszen nem szükséges magas fokú számítógépes ismeret egyszerű képek és szöveges állományok készítéséhez, illetve azok becsomagolásához egy zip állományba.

Feladatsor:

A feladatsorok esetében hasonló követelmények merültek fel, mint a tananyagok esetében. Ezeknek is gyorsan és egyszerűen módosíthatóknak kell lenniük, és az újabb feladatsorok elkészítésekor is előny, ha azok egyszerűen, különösebb ismeretek nélkül elkészíthetőek. Éppen ezért itt is hasonló elgondolás alapján érdemes megvalósítani a tárolást.

A tananyagoktól eltérően itt azonban nincs szükség képekre, hiszen nem szükséges ábrákat megjeleníteni. Ezért itt egy-egy kérdést egy-egy szöveges állománnyal meg tudunk oldani, és így elegendő X darab .txt állomány X darab kérdés megvalósításához. Ezeket hasonlóan a tananyagokhoz aztán elhelyezhetjük egy zip fájlban, és így egyben kezelhetjük a teljes feladatsort.

Azonban elérően a tananyagoktól, a feladatsorok esetében a txt állományok felépítésére szükséges megszorításokat tennünk. Erre azért van szükség, mert itt nem csupán bemutatni, hanem feldolgozni is akarjuk őket, azaz pontosabban fogalmazva, a

kérdésekre adott válaszokat. Vagyis a kérdések szövegén kívül más információk megadására is szükség van, melyek a következők:

Sor	Megnevezés	Leírás	Példa
1. sor	Kérdés szövege	Ide kerül a megjelenítendő kérdés szövege.	Mennyi egy háromszög belső szögeinek összege?
2. sor	A-válasz	Az első válaszlehetőség.	360°
3. sor	B-válasz	A második válaszlehetőség.	270°
4. sor	C-válasz	A harmadik válaszlehetőség.	180°
5. sor	D-válasz	A negyedik válaszlehetőség.	90°
6. sor	Jó válasz	A jó válasz sorszáma.	3
7. sor	Pontérték	A kérdésre adott jó válasz esetén ennyi pont jár érte.	2

Ahogyan az a fenti táblázatból is jól látszik, a szöveges állománynak 7 sorból kell állnia, amely soroknak rendre a következőket kell tartalmaznia: kérdés szövege, a négy válaszlehetőség, jó válasz, pontérték. Ezen információk alapján lehetőség nyílik arra, hogy az összeállított kérdéssorokat ne csak megjelenítsük a felhasználók számára, hanem az adott válaszok alapján kiszámítsuk az elért eredményt is.

Funkcionális elemzés

Az alkalmazás fejlesztése során rendkívül fontos pontosítani azt, hogy milyen feladatokat kell a programnak ellátnia, illetve milyen funkciókkal rendelkezzen az adott alkalmazás.

A szerver oldalon az egyes funkciókat a főablakon menü segítségével lehet elérni. Az alkalmazás többi része (pl.: tanuló felvétele) dialógusablakokban kerül megvalósításra. Ezek a dialógusablakok modálisan jelennek meg, ami azt jelenti, hogy nem tudunk visszatérni a főablakhoz, amíg az aktuális ablakot be nem zárjuk.

Az alkalmazás működéséhez mindenképpen szükséges, hogy a felhasználók (tanulók) adatait fel tudjuk vinni. Ehhez egy külön erre a célra megvalósított párbeszédablakra van szükségünk. Ugyanezen adatok módosítását szintén egy hasonló ablakban tudjuk elvégezni. A törlést is adatmódosításnak tekintjük, ezrért szintén itt van arra lehetőség, hogy egy már korábban felvett tanuló adatait töröljük a rendszerből.

Szükség van továbbá egy Java osztályra, amelyik lekezeli a kliensek részéről érkező csatlakozási kérelmeket, illetve a szerverhez érkező kéréseket. Mivel párhuzamosan van szükség a főablak és a szervertevékenységek folytatására, célszerű a szervert egy külön szálon (thread) elindítani.

A kliens hasonló felépítésű, mint a szerver. A klienst használják a tanulók, vagyis elindulásakor szükséges ellenőrizni, hogy tényleg egy, a szerver oldalon már felvett tanuló akarja-e használatba venni a programot. Ezt egy felbukkanó dialógusablak segítségével meg lehet oldani, amely nem engedi tovább folytatni az alkalmazás futását mindaddig, amíg meg nem adunk neki egy érvényes felhasználónév-jelszó párost.

Amennyiben ez megtörténik, akkor láthatóvá válhat a főablak, amelyen menüből elérhetőek az egyes funkciók. A kliensprogramnak 3 fő funkcióval kell rendelkeznie ahhoz, hogy a feladatát el tudja látni.

Az első, ami nélkül igencsak nehézkes volna bármi is, az új tananyagok és feladatsorok beszerzése, vagyis letöltése a szerverről. Itt a felhasználóknak lehetősége van arra, hogy a szerverről letöltsék azokat a tananyag, illetve feladatsor állományokat, amelyeket

aztán a program feldolgoz számukra. Mint az eddigi funkciók, ez is modálisan jelenik meg.

Második lényeges funkciója a tananyagok bemutatása. Ez teszi lehetővé a tanulók számára, hogy a meglévő tananyagokat megtekinthessék, áttanulmányozhassák. Ezen a ponton először is szükség van arra, hogy a tananyagok közül ki lehessen választani valamelyiket, amelyet aztán meg akarunk tekinteni. Ezt követően kerülhet sor a tananyag tényleges megjelenítésére. A tananyag diáin való lépkedéshez „Előre” és „Tovább” funkciójú gombok szolgálnak.

A harmadik lényeges funkció a feladatsorok kitöltése. Ez teszi lehetővé a tanulók számára, hogy ellenőrizzék, milyen mértékben sikerült elsajátítaniuk egy-egy tananyagot. A tananyagokhoz hasonlóan először itt is lehetőséget kell biztosítani a tananyagok közötti választásra, s ezután kerülhet sor a kérdések és a hozzájuk tartozó négy válaszlehetőség megjelenítésére. A feladatsor végén sor kerülhet az adott válaszok alapján a teljesítmény kiértékelésére is.

A szerver programhoz hasonlóan itt is szükség van egy Java osztályra, amely a hálózathoz kapcsolódó feladatokat végzi. Mivel itt is szükséges eközben, hogy a főprogram használható maradjon, ezért a kliens esetében is elengedhetetlen a szálak (thread) használata. Ez a kliens osztály bonyolítja az összes olyan feladatot, melynek során a szervert is be kell vonni a munkába.

Inputok és outputok megtervezése

Az inputok és outputok megtervezése szintén fontos része egy alkalmazás elkészítésének, hiszen a felhasználó ezeken keresztül tud kapcsolatot teremteni az alkalmazással. Mind az inputok, mind az outputok esetében alapvető követelmény, hogy azok tartalma rendezett, könnyen áttekinthető legyen. Ugyancsak lényeges, hogy minden elem szerepe egyértelmű legyen, s hogy a lehető legegyszerűbbé tegye a felhasználó munkáját.

Az inputképernyők lehetővé teszik, hogy a felhasználó adatokat szolgáltasson a program számára, illetve hogy a tárolni kívánt adatokat felvigye. Az alkalmazás lehetővé teszi a felhasználó számára azt, hogy ezeket az adatokat gyorsan és egyszerűen tudja a gépbe vinni.

Ezen inputképernyők közé tartozik azon párbeszédablak is, amelyen keresztül a szerverprogramon fel tudjuk venni az új felhasználókat.

Amint az a képernyőről készült fenti képen is látszik, egy nagyon egyszerű, könnyen áttekinthető ablakról van szó. Minden adatmező előtt fel van címkézve a sor, ezáltal világos, hogy milyen adatot vár az adott mezőben az alkalmazás, az adatok bevitelére pedig szövegmezők (textfield) szolgálnak. Miután a felhasználó minden adatot felvitt, két lehetősége van: a „Felvesz” feliratú gombra kattintva bekerül az adott tanuló a felhasználók listájára, feltéve, hogy minden szükséges adat meg lett adva ehhez, míg a „Mégse” gombra kattintva eltekinthet a bevitt adatok rögzítésétől.

Egy másik, szintén a szerverhez tartozó ablak input és output célokra is szolgál egyben. Ez a tanulók adatainak módosítására szolgáló párbeszédablak, ahol egyrészt meg lehet tekinteni a már felvitt tanulók adatait, másrészt pedig, amennyiben szükséges, módosítani is lehet azokat.

A tanuló adatainak megjelenítésére szolgáló rész nagyon hasonlít a tanulók felvételére szolgáló ablakéhoz, ami által azonnal át lehet látni az adatokat. Annyi eltérés azonban felfedezhető, hogy itt már nincs lehetőség a felhasználói név módosítására, mivel arra elsődleges kulcs jelleggel tekintettünk, vagyis az szolgál az egyes felhasználók azonosítására, és módosítása több kellemetlen hatással is járna. A többi adaton tetszés szerint módosíthatunk.

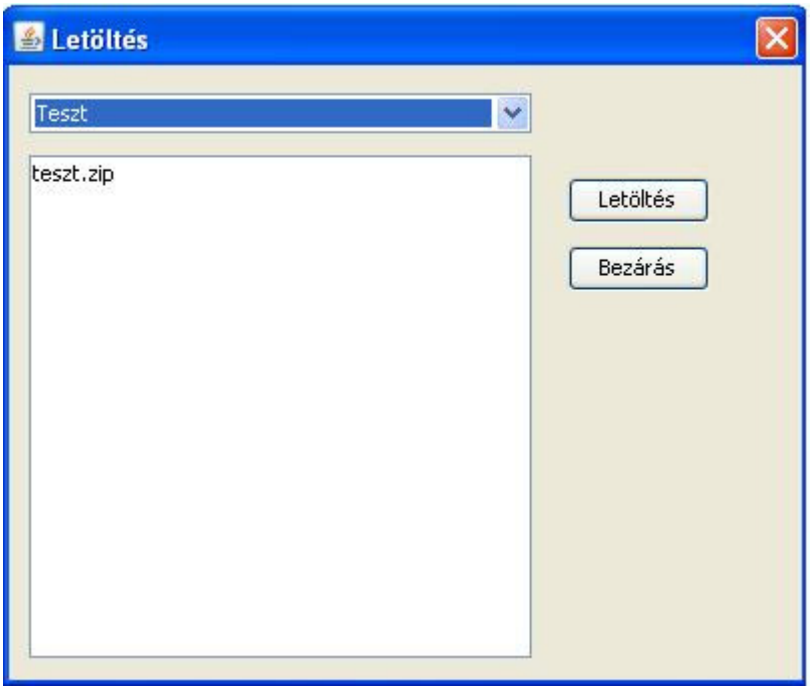
A képernyő bal oldalán a már korábban a felhasználók közé felvitt felhasználók listáját tekinthetjük meg. A listáról egy nevet kiválasztva azonnal láthatóvá válnak az adatai a jobb oldalon. Amennyiben változtatunk a felhasználó adatain, a „Módosít” gombra kattintva véglegesíthetjük azt. A „Törlés” gombbal az éppen kijelölés alatt álló felhasználót törölhetjük a tanulók listájáról, míg a „Mégse” gomb a módosítási szándékunktól történő elállást teszi lehetővé.

A következő inputképernyő már a kliensprogramhoz kapcsolódik, annak indulásakor szembesül vele a felhasználó, hiszen a bejelentkezési adatok megadására szolgál.

Az ablak felépítése rendkívül egyszerű, két szövegmezőben várja a felhasználónév, illetve a jelszó megadását. A felhasználói név megadásához szokványos szövegbeviteli mezőt használunk, ugyanakkor a jelszó bevitelére a speciálisan erre a célra használatos jelszóbeviteli mezőt (passwordfield) alkalmazzuk. Ez annyiban más, mint az előbbi, hogy a bevitt szöveg helyett csupán '*' karaktereket láthatunk.

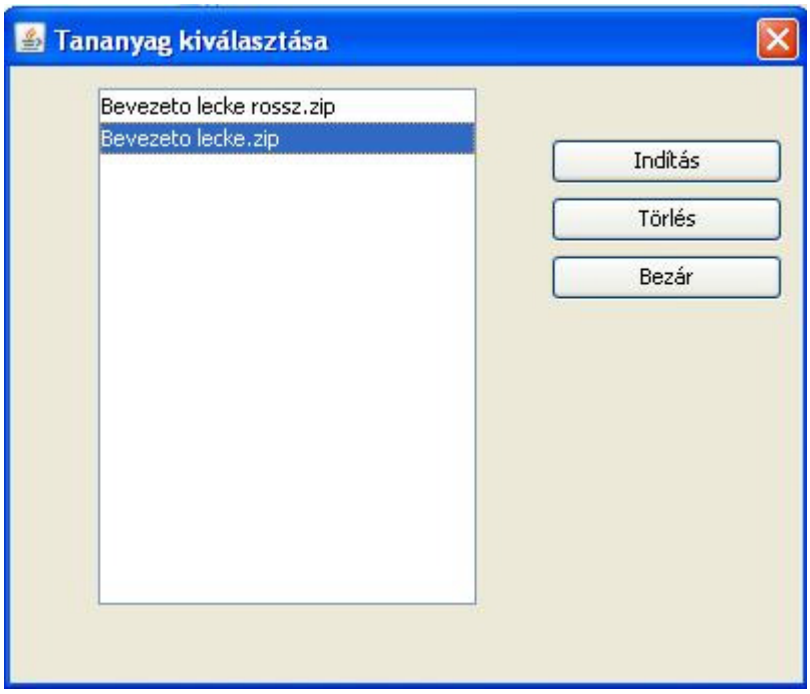
A szükséges adatok megadása után a „Bejelentkezés" gombra kattintva megtörténik a bejelentkezés (amennyiben a szerver helyesnek találja a bevitt adatokat), míg a „Kilépés" gomb hatására befejeződik az alkalmazás futása.

A következő bemutatásra kerülő képernyő szintén a kliensprogramhoz tartozik, azon belül is a tananyagok és feladatsorok letöltésére szolgál.



Az ablakon nem túl sok komponens kapott helyet. Az ablak bal felső sarkában található legördülő listából (combobox) ki lehet választani, hogy tananyag fájlt vagy feladatsort szeretnénk letölteni. Attól függően, hogy melyik lehetőséget választja a felhasználó, az alatta található listában megjelenik a szerveren elérhető tananyagok vagy feladatsorok listája. Ezt követően a megfelelő elemet kiválasztva és a „Letöltés” gombra kattintva a kiválasztott állomány letöltésre, és a megfelelő mappában való elhelyezésre kerül. Amennyiben a felhasználó már nem kíván több állományt letölteni, a „Bezárás” gombra kattintva kiléphet az alkalmazás ezen funkciójából.

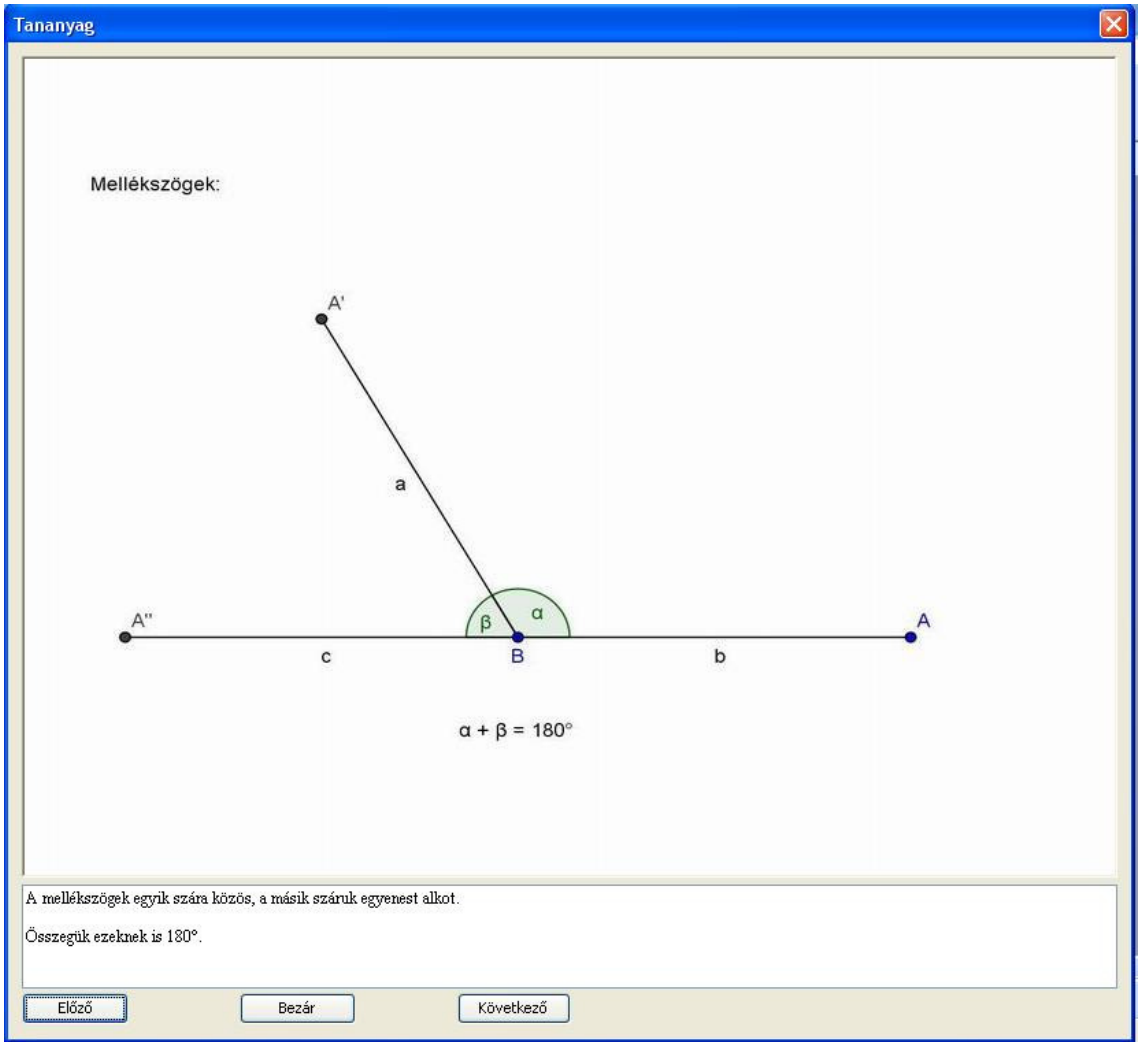
A következőkben két inputmegadásra alkalmas képernyőt egyszerre mutatnék be, mivel azok szinte mindenben megegyeznek. A megjelenítésre kerülő tananyag, illetve feladatsor kiválasztására szolgáló képernyőkről van szó.



A tananyag kiválasztását, illetve feladatsorok kiválasztását lehetővé tevő dialógusok nagyon hasonlítanak az azok letöltését lehetővé tevő ablakhoz. Baloldalon egy listában láthatóak az elérhető tananyagok vagy feladatsorok (attól függően, hogy éppen melyik menüpontot választottuk ki a főmenüben), jobboldalon pedig három gomb található. A „Bezár” gomb hatásár visszaléphetünk a főmenübe, míg a „Törlés” gombbal az éppen aktuálisan kiválasztott tananyagot vagy feladatsort törölhetjük. Ez a törlés természetesen csak a klienst érinti, a szerveren továbbra is elérhető marad az adott

tananyag vagy feladatsor. Az „Indítás” gomb hatására megjelenik a tananyagot vagy feladatsort bemutató ablak, és a tényleges munka elkezdhető.

A tananyag elindításának hatására, amennyiben a tananyag megfelelő, megjelenik a tananyag bemutatására szolgáló ablak, és elkezdhető a tananyag megtekintése. Ilyenkor a következő ablakot látjuk:

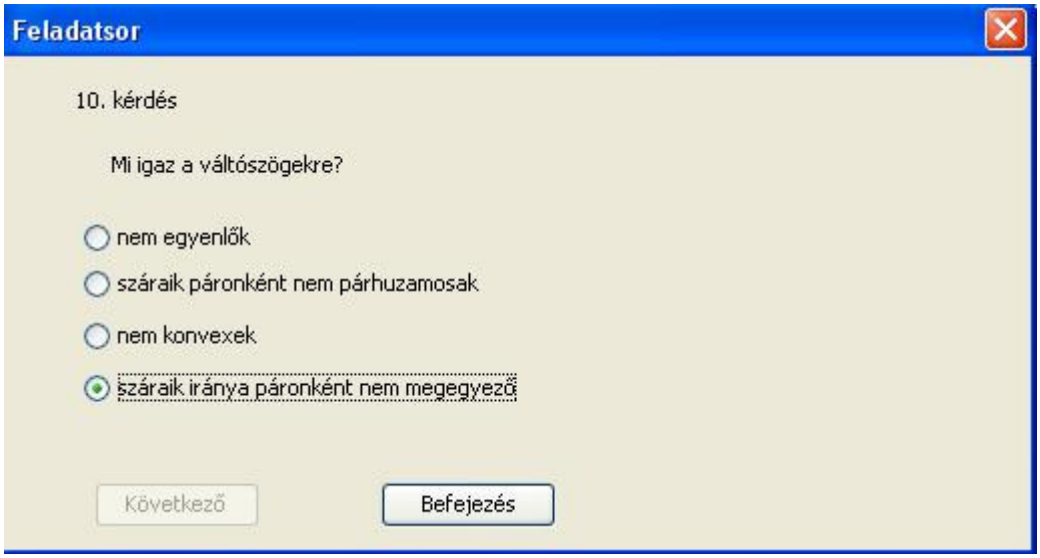


Az ablak felső részében (canvas) kerül megjelenítésre az adott diához tartózó képfájl, míg alatta egy szövegek megjelenítésére alkalmas terület kap helyet (textArea), ahol is az adott képhez tartozó kiegészítő magyarázat jelenik meg.

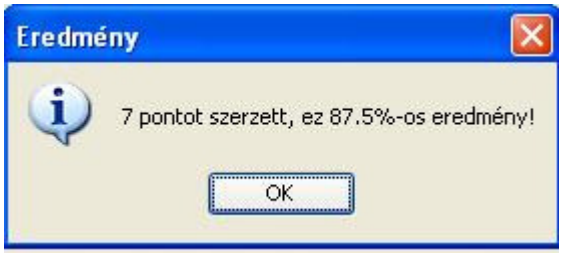
Az ablak alján három gomb található. Az „Előző” gomb akkor aktív, ha épp nem az első diát nézzük, és rákattintva eggyel visszaléphetünk. A „Következő” gomb akkor aktív, ha a megjelenített dia nem az utolsó az adott tananyagban, és rákattintva egyet léphetünk előre a diák sorában. A „Bezár” gomb hatására az aktuális ablak eltűnik, és

visszajutunk a tananyagok kiválasztására szolgáló párbeszédablakhoz, ahol a már korábban tárgyalt lehetőségeink vannak.

Amennyiben feladatsort választunk ki, úgy az adott párbeszédablakon az „Indítás” gombra kattintva megjelenítésre kerül a feladatsor. Ekkor az alábbi ablakot látjuk:



Az ablak felső részén az aktuális kérdés feladatsoron belüli sorszáma látható, míg alatta a kérdés szövege, illetve a négy lehetséges válaszlehetőség található. A tanuló ezek közül választhatja majd ki az általa helyesnek vélt választ, s a „Következő” felirattal ellátott gombra kattintva tekintheti meg a következő kérdést. Az utolsó kérdés esetében a „Befejezés” gombra kattintva van mód a feladatsor véglegesítésére, s ezt követően a felhasználó visszajelzést kap az alkalmazástól a teljesítményét illetően egy információs üzenet keretében:



Az üzenet közli a felhasználóval a megszerzett pontok mennyiségét, illetve azt, hogy ez hány százalékos eredményt jelent a feladatsor tekintetében. Azonban nemcsak a felhasználónak lényeges tudni az elért eredményeket, hanem adott esetben a tanár is kíváncsi lehet a teljesítményekre. Ezért nem csak a kliens részéről kerül sor output üzenet megjelenítésére, hanem a szerver oldalon is output generálódik, mégpedig egy az

aktuális dátumot és időt tartalmazó nevű állomány, amelyben az alábbi adatok kerülnek feljegyzésre: a feladatsort kitöltő felhasználó azonosítja (felhasználónév), a felhasználó által elért pontszám, illetve az összes elérhető pontszám.

Néhány implementációs probléma bemutatása

Ahogyan az már a bevezető részekben is említésre került, az előző fejezetekben megtervezett alkalmazás tényleges megvalósítására Java programozási nyelven került sor, melyhez a NetBeans 6.8-as verziójú fejlesztői környezetét használtam. Az alábbiakban az implementálás során felvetődött lényegesebb kérdéseket, problémákat, érdekességeket szeretném bemutatni.

Az első komolyabb kérdésként az vetődött fel az alkalmazás megvalósítása során, hogy az egyes tanulók adatait hogyan tegyem elérhetővé a program leállása, és újraindítása után, vagyis hogy miként gondoskodjak az adatok tárolásáról. Nos, mivel viszonylag kisebb adatmennyiségről (maximum néhány tucat diákról) van szó, és az egyes diákokat a Tanulo.java oszály példányaiként tároljuk a programban, adta magát a szerializáció[1]. E művelet végzése során a Java objektumok mentésre kerülnek a háttértárolóra, s a program újbóli indítása folyamán onnan a megfelelő változókba visszatölthetők az eredeti értékek. Ezzel tulajdonképpen a probléma megoldottá is vált, mindössze arra volt még érdemes odafigyelni, hogy ne csak a program leállásakor kerüljenek az adatok mentésre, hanem új elem felvétele, régi elem módosítása/törlése esetén is, hogy az esetleges szabálytalan leállás miatt se vesszenek el az adataink.

Mivel az alkalmazást fel kellet vérteznem a hálózati kommunikáció képességével is, így ezzel a kérdéskörrel is kénytelen voltam behatóbban foglalkozni. Olyan szerver, illetve kliens modulra volt szükségem, amely lehetővé teszi egyidejűleg tetszőleges számú kliens csatlakozását, ugyanakkor a háttérben futó hálózatkezelő eszközök nem vonják el az erőforrást a program más elemeitől sem.

Mivel korábban Java nyelven nemigen volt szükségem ilyen program írására, ezért mielőtt hozzákezdtem volna e probléma megoldásának, igyekeztem megfelelő mértékig tisztában lenni azzal, hogy milyen lehetőségeim vannak az elképzeléseim megvalósításában. Ennek érdekében több forrás alapján is áttanulmányoztam a Java-ban lehetséges hálózatkezelési módokat, mielőtt hozzáfogtam volna a tényleges megvalósításhoz. A témával való ismerkedés megkönnyítése érdekében először az egyszerű TCP szerver és kliens működését tekintettem át [2], s csak ezt követően kezdtem el olyan eszközök és lehetőségek után kutatni, amelyek segítségével a fenti kívánalmaknak megfelelő szervert, illetve klienst hozhatok létre az alkalmazásom

számára. Ezek közül egy „Hálózatkezelés Javában” című írás [3] volt a leghasznosabb számomra, innen vettem az ötletet a szerver külön szálon futtatásához annak elkerülése érdekében, hogy az elvonja az erőforrásokat az alkalmazás fő részeitől is. Ez tulajdonképpen nagyon egyszerűen elérhető, mindössze az alábbi sorok használatára van szükség a szervert megvalósító osztály konstruktorában:

```
life = new Thread(this);
life.setDaemon(true);
life.start();
```

Majd ezt követően a kiszolgáló tevékenység az adott osztály run eljárásában zajlik. Esetemben ez nem volt más, mint egy általános startServer(), illetve startClient metódus meghívása, amely elindítja, és a kéréseknek megfelelően tereli tovább a hálózatos forgalom kezelését.

Ezt, mármint a szerver különböző feladatainak leválogatását úgy oldottam meg, hogy a klienstől beérkezett kéréseknek kódot adtam, amely tulajdonképpen csupán egy szám. Miután a szerver kapott egy ilyen kódot, egy case szerkezetben a kód számának megfelelő ágat lefuttatja, amely tulajdonképpen csupán egy, az adott kérés kiszolgálását elvégző eljárás hívását tartalmazza. Ezáltal a case szerkezet jól átlátható maradt, könnyen meg lehet érteni annak működését. Miután a szerver elvégezte a kérésnek megfelelő tevékenységeket, újabb kódra vár a kliens részéről. Így biztosítva a folyamatosan beérkező kérések teljesítését.

Szintén érdekes kérdést vetett fel a tananyagot megjelenítő ablakon a képek kezelése. Szükség volt arra, hogy kiválasszak egy olyan grafikus komponenst, amelyen egyszerűen és gyorsan jeleníthetem meg a kívánt képeket. Erre legalkalmasabbnak a Canvas elem ígérkezett, de eleinte nem igazán tudtam merre elindulni vele, mivel az eszközei közül hiányzott egy olyan metódus, amivel igazán egyszerűvé tehettem volna a képkezelést. Ezt a problémát egy alapvető funkciókkal felvértezett képnézegető programot bemutató angol nyelvű leírás alapján sikerült áthidalnom [4]. A kérdéses képnézegető esetében létrehoztak egy új Java osztályt, amely kvázi a Canvas kiterjesztése. Így azon túl, hogy lehetőség van használni a Canvas eszközeit, módom nyílt saját eljárások hozzáadására is. Nekem két eljárásra volt szükségem, egy képbetöltő eljárásra (loadImage), illetve egy a képet a canvasra rajzoló metódusra (paint). Ezen eljárások alkalmazása nagy segítséget nyújtott abban, hogy azokon a

helyeken, ahol szükség van a képek betöltésére és megjelenítésére, ott a programkód rövid, áttekinthető, és megfelelően strukturált maradhasson, ugyanakkor a szükséges feladatokat maradéktalanul ellássa.

Miután már a tervezés során nyilvánvalóvá vált, hogy az alkalmazás számos részegységében szükség van arra, hogy tömörített állományokat (zip) tudjunk kezelni, ezért logikus lépésnek tűnt egy a kifejezetten erre létrehozott osztály megvalósítása is (ZipHandler). Ezen osztálynak a tervezés szempontjait figyelembe véve három alapvető feladatnak kellett megfelelnie: első lépésként alkalmasnak kell lennie, hogy egy megadott zip állományt az átmeneti könyvtárba kicsomagoljon feldolgozás céljából, vissza kell tudni adnia a kép, illetve szöveges állományok listáját, valamint használat után gondoskodnia kell a nem kívánatos állományok eltávolításáról.

A zip állományok kicsomagolása, illetve a hátramaradt, a továbbiakban már nem szükséges állományok eltávolítása nem jelentett problémát, azok megoldása viszonylag egyszerű eszközökkel megoldhatónak bizonyult [5].

A különböző típusú állományok kiválasztása azonban első nekifutásra kissé nehézkesebbnek tűnt. Sajnos közvetlen a fájlnevekre nem lehetett kritériumokat tenni, vagyis megszabni például, hogy csak a bizonyos kiterjesztésű állományokkal szeretnénk foglalkozni, ezért más megoldást kellett keresnem. Rövid utánajárás során egyértelművé vált, hogy az az eszköz, amire az adott helyzetben szükségem van, a FilenameFilter nevű interfész osztály [6]. Ezen interfész mindössze az accept() eljárást tartalmazza, amelyben lehetőségünk van a fájlnevekre vonatkozó megszorítások definiálására. Ezt például én a következőképpen tehettem meg:

```
FilenameFilter filter = new FilenameFilter() {
    public boolean accept(File dir, String name) {
        return name.endsWith(".txt");
    }
};
```

Az így létrehozott filtert egy könyvtár list() eljárásának paraméterként megadva már csak azon állományok kerülnek bele a listába, amelyek a filterben található követelményeknek eleget tesznek.

A feladatsorokban megadott kérdések, illetve a hozzájuk tartozó információk (úgymint a válaszlehetőségek, jó válasz, kérdés pontértéke) feldolgozása ha nem is problémát, de

mindenképpen fontos kérdést jelentett. Igyekeztem olyan megoldást találni, amely egyszerre gyors és egyszerű, ugyanakkor nem köti meg a későbbi feldolgozás tekintetében a kezemet. Végül egy külön az erre a célra létrehozandó Question osztály mellett döntöttem, amelyet listába fűzve nagyszerűen kezelhetőnek bizonyult. Miután egy feladatsort tartalmazó állományt beolvastam, egyszerűen az állományok számának megfelelően egy ciklusban sorra létrehoztam a Question osztályú példányokat (maga a konstruktor elvégzi az összes szükséges adattag beállítását), majd azokat egy listába tettem. Ily módon miután az összes állományon végighalad a program, a végeredmény egy kérdéslista, amelyet könnyen fel lehet használni az alkalmazás szükségleteinek megfelelően.

Természetesen az eddigieken kívül számos érdekes, néhol komoly fejtörést okozó kérdéssel és problémával szembesültem az alkalmazás implementálása folyamán, de ezek mindegyikének bemutatása nem volt célom. Mindössze néhány példán keresztül szerettem volna érzékeltetni, hogy a kódolási folyamat során hogyan lehet úgy megoldani a felmerülő gondokat, hogy az az alkalmazás szempontjából a lehető leghatékonyabb, és ami még fontosabb, könnyen használható legyen.

Tesztelés, hibakeresés, javítás

Az inputok és outputok megtervezése után az alkalmazás elkészítése folyamán szükség van arra, hogy megbizonyosodjunk arról, hogy a program helyesen működik, s a meghatározott feladatokat megfelelően végzi el. Erre a tesztelés és a hibakeresés során derül fény. A tesztelés magában foglalja a program funkcióinak a kipróbálását, ideértve az adatfelvitelt, vagy a különböző ablakokon való manipulációs lehetőségek kipróbálását is.

Az adatfelvitel különösen kényes kérdés egy alkalmazás esetében, hiszen itt aztán tényleg rengeteg a hibalehetőség még gondos tervezés esetén is. A bevitel helyességéről extrém adatokkal való teszteléssel győződhetünk meg. Ez azt foglalja magában, hogy olyan adatokat próbálunk meg felvinni, amelyeket a programnak ki kell szűrnie a helyes működés biztosítása érdekében. Megpróbálhatunk üresen hagyni mezőket, s megfigyelhetjük, hogy helyesen reagál-e az alkalmazás az ilyen helyzetekre. Azt is ellenőrizni kell, hogy az egyértelmű azonosításra szolgáló adatelemből (ami az alkalmazásunk esetében például a tanulók esetében megjelenő felhasználónév) lehet e több azonosat felvinni, mivel ez bár a külső szemlélő számára első pillantásra ugyan nem okozna komolyabb nehézségeket az alkalmazás használatában, mégis a program nem megfelelő működéséhez vezetne.

Itt el is érkeztünk az alkalmazás szempontjából az első felfedezett hibához. Ugyanis bár úgy gondoltam, hogy levédtem az azonos felhasználónevek felvételének lehetőségét, a program mégis szívfájdalom nélkül engedte felvenni az ugyanazzal a felhasználónévvel megadott tanulók egész sorát. Az ellenőrzést végző programrészre pillantva azonban rögtön nyilvánvalóvá válik a hiba:

```
if (userField.getText().equals("")){
    valid = false;
}
else{
    int n = userList.size();
    for (int i = 0; i < n; i++){
        if(userList.get(i).getUsername()
                                .equals(userField.getText())){
            voltmar = true;
        }
    }
}
```

```
If (valid){
   //tanuló felvétel
}
```

Az iménti kódrészletre pillantva azonnal látható, hogy hol követtem el figyelmetlenséget. A tanuló felvehetőségekor ugyanis csupán azt szabtam meg követelménynek, hogy minden mezőben érvényes adat legyen megadva, azt, hogy egyedinek is kell lenni már nem. A feltételt (valid && !voltmar)-ra javítva már helyesen működik a tanulók felvételének lehetősége.

Egy másik probléma a tanulók adatainak módosításakor jelentkezett. Amikor a listában megjelenő felhasználónevek alá kattintottam, akkor NullPointerException-üzenetet dobott a program, ami azt jelezte, hogy olyan elemre akarok hivatkozni, ami nem is létezik. Ez abból fakadt, hogy a program akkor is érzékelte a listában kiválasztott elem sorszámát, ha az adott elem nem létezett (vagyis üres volt a helye), és utána ezzel a sorszámmal keresett a tanulók között, ami nyilván nem létezett, így hibát generált. Ugyan ha nem a fejlesztői környezetben futtattam a programot, a hiba majdhogynem észrevehetetlen maradt, mégis a javítás mellett döntöttem, mivel időnként előfordult az az anomália, hogy ilyen hiba után több felhasználót engedett kijelölni, ami újabb hibákat eredményezett volna.

A javítás tulajdonképpen nagyon egyszerű volt, mindössze azt kellett vizsgálni, hogy érvényes-e a kijelölt elem, és amennyiben nem, akkor kijelölésként az első elemet határoztam meg. Ezzel ez a probléma meg is szűnt.

Egy következő hibaforrásként jelentkezett, ha az alkalmazást az ablakkezelő gombok Bezárás (X) gombjával léptem ki bizonyos dialógusablakok esetében. Ilyen esetekben ugyanis nem futottak le azok a kódok, eljárások, amik egyébként az ablak bezárására létrehozott gomb használatakor lefutnak. Ez pedig a kérdéses párbeszédablakok esetében a következő használatukkor kiszámíthatatlan viselkedést idézett elő, vagyis időként előfordult, hogy nem került törlésre egy-egy szövegbeviteli mező értéke, vagy az ablak megjelenésekor nem futottak le olyan eljárások, amiknek az ablak tartalmainak inicializálása lett volna a dolga.

Annak érdekében, hogy elkerüljem az ablakkezelő gombbal való bezárás okozta kellemetlenségeket, a form tulajdonságai között beállítottam a defaultcloseOperation

tulajdonság értékéül a „NOTHING” értéket, aminek hatására a kérdéses bezárógomb a továbbiakban nem használható, vagyis ezentúl csak a megfelelő gomb használatával lehet bezárni egy-egy ablakot, így kerülve el az előzőekben tapasztalt hibajelenségeket.

A tananyagok megtekintése esetében szintén sikerült egy kisebb hibára bukkannom, ami azonban jelentősen befolyásolta a program működését. Bezáráskor nem kerültek törlésre a temp könyvtárban ideiglenesen elhelyezett kép- és szövegállományok, így egy újabb tananyagfájlt vagy feladatsort megnyitva előfordultak hibajelenségek, mivel nemcsak az adott tananyaghoz vagy feladatsorhoz tartozó állományok kerültek beolvasásra.

A probléma megoldása a felderítését követően végtelenül egyszerűnek bizonyult, mindössze meg kellett az ablak bezárásakor hívni a már korábban a ZipHandler osztályban megírt delTemp() eljárást.

Az ablakok átméretezésekor is adódtak időnként problémák. Bizonyos esetekben az ablakok átméretezésekor az azon elhelyezett komponensek elhelyezkedése összekavarodott. Ugyan ez az alkalmazás funkcionalitását nem befolyásolta, vagyis a program továbbra is működőképes volt, viszont esztétikailag nem volt túl előnyös ez a jelenség. Ezért a formok elemeinek elhelyezkedésének beállítása után az ablakok resizable tulajdonságát „false” értékre állítottam, így ezek után az ablakok eredeti elrendezése védve van, hiszen a továbbiakban nem engedélyezett az ablakok átméretezése.

A tananyagok és feladatsorok letöltésekor szintén jelentkezett hiba. Ugyanis a szerver nem minden esetben tudta eljuttatni a teljes állomány egyszerre a klienshez, így a kliens időnként hibás állományokat tudott csak létrehozni. Ennek kiküszöbölésére beépítettem a letöltés menetébe egy ellenőrző részt, amely folyamatosan számontartja, hogy a teljes állományból mekkora részt sikerült már letöltenie a kliensnek. Így amennyiben van még letöltésre váró adat, akkor a kliens jelzi a szervernek, hogy meddig sikerült megkapnia az állományt, és attól a ponttól újrakéri a szervertől az állomány töltését.

Ezt a folyamatot addig végzik, amíg a kliens azt nem jelzi a szervernek, hogy az állomány hosszának megfelelő mennyiségű adatot sikerült letöltenie.

A gondos tervezésnek köszönhetően több jelentősebb hibát nem sikerült felfedeznek, ami mindenképpen alátámasztja, hogy hasznos az alkalmazások fejlesztése során megfelelő mennyiségű időt fordítani a tervezés folyamatára. Természetesen nem állíthatom, hogy minden hibát sikerült kiszűrnöm, hiszen ritka az olyan program (ha egyáltalán létezik), amelyik mentes a hibáktól.

Ugyanakkor úgy gondolom, hogy sikerült megszüntetni az összes olyan hibaforrást, amely számottevően rontani tudta volna az alkalmazás használhatóságát.

Az alkalmazás használata

A szerver program

Az alkalmazás használatbavételéhez először is szükség van a szerver elindítására. Ezt a szerver könyvtárban található Oktato_szerver.jar állomány indításával tehetjük meg. Miután megjelent a szerver ablaka, a Fájl menü Indítás menüpontját kiválasztva van lehetőség a szerver tényleges elindítására. A leállítása pedig a Fájl menü Kilépés menüponttal lehetséges.

A szerveren ezenkívül lehetőség van a tanulók felvételére, adataik módosítására, illetve a tanulók törlésére is.

Tanuló felvételére a Tanuló menü Tanuló hozzáadása menüpont segítségével van lehetőség. A megjelenő dialógusablakban a megfelelő adatokat megadva, és a Felvesz gombra kattintva megtörténik a tanuló felvétele (amennyiben a megadott adatok elfogadhatóak). A Mégse gomb hatására a felvételtől eltekintünk, és visszatérhetünk a főablakhoz.

A tanulók adatainak módosítását a Tanuló menü Tanuló módosítás/törlés menüpont alatt végezhetjük el. A menüpontra kattintva megjelenik egy párbeszédablak, ahol kiválaszthatjuk a módosítani vagy törölni kívánt tanulót, és az adatok átírását követően a Módosít gombra kattintva véglegesíthetjük azt. Amennyiben törölni szeretnénk valamelyik felhasználót, úgy annak kiválasztását követően egyszerűen csak a Töröl gombra kell kattintani, és a törlés máris végbemegy. A Mégse gombra kattintva itt is lehetőségünk van a főablakhoz visszalépni.

A szerverprogram bezárása csak a kliensek lecsatlakozását követően.

A kliens indítása és leállítása

A kliens elindítása a kliens könyvtárban található Oktato_kliens.jar állománnyal lehetséges. Elindítás előtt azonban nem árt megbizonyosodni arról, hogy a könyvtárban található host.ini állományban helyesen lett-e beállítva a szerver IP címe. A kliens elindítását követően a bejelentkezési ablak jelenik meg. Itt meg kell adni egy érvényes felhasználónév és jelszó párosát. Amennyiben ez nem sikerül, akkor a program leáll.

Helyes felhasználónév és jelszó megadása esetén a kliens elindul, és használhatóvá válik.

A kliens leállítása a Fájl menü Kilépés menüpontjára kattintva történik meg.

Tananyagok és feladatsorok letöltése

A kliensen lehetőség van a szerveren található tananyagok és tesztsorok letöltésére. Ezt a funkciót a Tananyag menü Letöltés menüpontjának kiválasztásával érhetjük el. Ekkor megjelenik a letöltéseket lehetővé tevő ablak. Itt felül ki kell választani, hogy tananyagot vagy feladatsort akarunk letölteni. Miután ez megtörtént, megjelenik a kiválasztott típusú állományok listája. Ezen listából kiválaszthatjuk a nekünk tetszőt, majd a Letöltés gombra kattintva letölthetjük az a kliens megfelelő könyvtárába. A Bezárás gombra kattintva visszaléphetünk a kliens főablakába, ahonnan aztán újra választhatunk az elérhető funkciók közül.

Tananyag indítása, törlése

Miután már rendelkezésre állnak tananyagok a kliensprogram számára, a Tananyag menü Indítás menüpontját választva lehetőségünk nyílik egy tananyag megtekintésére. Először megjelenik egy ablak, ahol a rendelkezésre álló tananyagok közül választhatunk. Miután kiválasztottuk a kívánt tananyagot, az Indítás gombra kattintva elindul a lejátszás. Amennyiben törölni akarunk egy tananyagot, arra szintén van lehetőségünk, csupán a tananyag kiválasztása követően nem az Indítás, hanem a Törlés gombra kell kattintanunk.

Miután elindítottunk egy tananyagot, megjelenik a tananyag bemutatása ablak. Itt három lehetőség közül választhatunk. A Következő gombra kattintva megtekinthetjük a tananyag következő diáját, míg az Előző gombra kattintva eggyel visszaléphetünk. Természetesen az első és az utolsó dia esetében csak az egyik választási lehetőség adott ezek közül. A Bezárás gombbal pedig visszaléphetünk a tananyag kiválasztásához.

Teszt kitöltése

A tesztek kitöltésére hasonló módon van lehetőségünk, mint a tananyagok megtekintésére. A funkció eléréséhez a Tananyag menü Teszt menüpontját kell kiválasztanunk. Ekkor megjelenik egy a tananyagok kiválasztásához hasonló ablak, ahol

a kitölteni kívánt teszt kiválasztása után az Indítás gombra kattintva máris kezdhetjük a feladatsor megoldását. A Törlés gombbal itt is van lehetőségünk a már felesleges tesztfájlok törlésére.

A teszt elindítása után rögtön láthatjuk az ahhoz tartozó első kérdést. Ebben az esetben nincs lehetőségünk a visszalépésre, csupán előre haladhatunk. Miután a négy válaszlehetőség közül kiválasztjuk az általunk helyesnek véltet, kattintsunk a Következő gombra, s ekkor láthatjuk az újabb kérdést. Üres válasz esetén az alkalmazás hibás válasznak tekinti az adott kérdésre adott válaszunkat. Az utolsó kérdéshez érkezve a Befejezés gomb aktiválódik, s rákattintva rögtön láthatjuk az általunk elért eredményt is egy információs ablakban.

Tananyagok és feladatsorok összeállítása

A tananyagok és feladatsorok előállítását a könnyű, bárki által elvégezhető bővíthetőségük miatt elkülönítettem az alkalmazástól, lehetővé téve, hogy szinte bármilyen tananyag összeállítható legyen.

Tananyagok

A tananyagokra vonatkozóan nagyon kevés megszorítás van. Mindössze azt kell figyelembe venni, hogy egy-egy megjeleníteni kívánt diához 1 képre, illetve 1 szöveges állományra van szükség. A képnek jpg kiterjesztésűnek kell lennie, míg a szövegnek a txt kiterjesztés az elfogadott.

Az állományok elnevezésekor érdemes követni néhány alapelvet. Az állományok lehetőleg 01*.jpg, 02*.jpg, … illetve 01*.txt, 02*.txt, … elnevezést kövessenek. Erre azért van szükség, hogy biztosítva legyen a képek és a hozzájuk tartozó szövegek sorrendje.

Az elkészült fájlokat egy zip állományba kell tenni, és a zip fájlt be kell másolni a szerver /tananyagok könyvtárába. Ezt követően a tananyag elérhetővé válik.

Feladatsorok

A feladatsorok készítése szintén nagyon egyszerűen történik. A feladatsorok egy-egy kérdése egy-egy szöveges állományban (txt) kerül eltárolásra. Vagyis egy kérdés megalkotásához mindössze egy szöveges állományt kell írnunk, betartva néhány megkötést.

A szöveges állományoknak az alábbi módon kell felépülniük:

Sor	Megnevezés	Leírás	Példa
1. sor	Kérdés szövege	Ide kerül a megjelenítendő kérdés szövege.	Mennyi egy háromszög belső szögeinek összege?
2. sor	A-válasz	Az első válaszlehetőség.	360°

3. sor	B-válasz	A második válaszlehetőség.	270°
4. sor	C-válasz	A harmadik válaszlehetőség.	180°
5. sor	D-válasz	A negyedik válaszlehetőség.	90°
6. sor	Jó válasz	A jó válasz sorszáma.	3
7. sor	Pontérték	A kérdésre adott jó válasz esetén ennyi pont jár érte.	2

Vagyis minden egyes állománynak 7 sort kell tartalmaznia, amelyek rendre a fentebb felsorolt információkat tartalmazzák. Amennyiben egy 10 kérdésből álló tesztsort akarunk összeállítani, úgy 10 darab szöveges állományt kell létrehoznunk a fenti szabályok szerint. Az állományokat itt is zip állományba kell csomagolni, és ezután azt be kell másolni a szerver /teszt könyvtárába.

Feladatsorok esetén azonban van még egy feladatunk. Amennyiben el kívánjuk tárolni a felhasználók eredményeit, létre kell hoznunk a /eredmenyek könyvtárban egy a tesztsor nevével megegyező nevű könyvtárat. Ez proba.zip esetén a /eredmények/proba.zip/ könyvtárszerkezetet jelenti.

Ezt követően a felhasználók eredményei a proba.zip nevű könyvtárba kerülnek, ahol az állományok neve a következőképpen épül fel:

```
<aktuális dátum>-<aktuális idő>.txt
```

Az állomány tartalmazza a felhasználó nevét, az elért pontszámot, illetve a feladatsorban elérhető maximális pontszámot. Ezen információk birtokában akár értékelni is lehet a tanulók teljesítményét.

Éles teszt a célközönséget bevonva

Már az alkalmazás tervezésének kezdeti szakaszában megfogalmazódott bennem a gondolat, hogy szeretném, ha azt az elkészülte után éles körülmények között, középiskolai keretek között ki is tudnám próbálni.

Erre a feladatra aztán sikerült elintéznem, hogy egy középiskolai osztály 11 tanulója egy délutáni fakultatív foglalkozás keretein belül a segítségemre legyen a program iskolai keretek között történő kipróbálására is.

Miután kiválasztottunk egy gépet a szerverprogram futtatása céljára, elindítottam rajta a szervert. A gépre telepített tűzfal ezzel egy időben kérte, hogy engedélyezzem az adott alkalmazás számára, hogy hozzáférjen a hálózathoz. A tűzfalon történő megfelelő beállítások elvégzése után minden rendben zajlott, a szerveren felvettem a jelenlévő tanulók adatait.

Miután 11 gépre felmásoltuk a kliens programot, és azok host.ini állományába beírták a tanulók a szerver IP címét, a kliens programok gond nélkül elindultak, és szép sorban mindenkinek sikerült a szerverhez csatlakozni.

A tanulók ezután rövid ideig szabadon ismerkedhettek az alkalmazással, ami során azt a megállapítást tették, hogy a program funkcióit könnyű átlátni, hamar rájöttek maguktól is, hogy mi mire való.

A rövid ismerkedést követően megkértem őket, hogy töltsenek le egy tananyagot a gépükre, majd a hozzá tartozó feladatsort is. Tekintve, hogy ezúttal elsősorban nem a matematika oktatása, hanem az alkalmazás kipróbálása volt az elsődleges célom, így egy egyszerű, mindenki által ismert információkat tartalmazó próba tananyagról volt csupán szó.

A tananyagot mindenkinek sikerült megnyitni, végignézni, s azt követően a feladatsorban található kérdésekre válaszolni. Mivel a kérdések is igen egyszerűek voltak, megbeszélés szerint néhányan elrontottak pár tetszőleges kérdést, hogy meg tudjuk nézni, vajon jól számolja-e össze a program a pontszámaikat ilyen esetekben is.

Jelentősebb hibára szerencsére már nem derült fény, egy-két ablakméretezéssel kapcsolatos probléma azonban akadt. Ilyenkor jellemzően az fordult elő, hogy nem túl

esztétikus elrendezésbe kerültek az ablakon az elemek, azonban a működést ez érdemben nem befolyásolta. Mindenesetre a probléma megoldása kívánatosnak tűnik, így ezt mindenképpen tervbe vettem.

A feladatok végeztével a kijelentkezések is rendben mentek, sem a szerver, sem a kliensek oldalán nem jelentkezett érzékelhető hiba. Néhány tanulót megkértem, hogy próbaképpen jelentkezzenek be újra, ami minden további nélkül sikerült is, vagyis a jelek szerint megfelelően működtek a programok.

Szeretném megköszönni minden érintettnek, hogy idejüket nem sajnálva a segítségemre voltak abban, hogy ki tudjam próbálni az elkészült alkalmazást ilyen körülmények között is.

Fejlesztési lehetőségek

Bár az elkészült program kielégíti a felhasználásával kapcsolatban támasztott követelményeket, mint minden program esetében, úgy itt is van mód az alkalmazás továbbfejlesztésére.

Az egyik ilyen lehetőség a tananyagok és feladatsorok összeállításának programba építése. Lehetne készíteni egy olyan funkciót az alkalmazáshoz, ahol lehetőség lenne kép- és szövegállományok hozzáadogatásával is készíteni egy tananyagot, amely tananyagot aztán az elkészültét követően a program önállóan elhelyezné a megfelelő könyvtárban. Ezáltal egy kissé tovább egyszerűsödne a tananyagok és feladatsorok elkészítése az alkalmazás számára.

Amennyiben igény merülne fel rá, úgy a tesztsorok esetében is meg lehetne oldani, hogy a kérdés mellett képet is meg tudjunk jeleníteni. Ez lehetővé tenné azt, hogy a megjelenített ábrával kapcsolatban tegyen fel kérdést a program. Szintén a feladatsorokkal kapcsolatos esetleges fejlesztési lehetőség az időkorlát bevezetése. Egy számláló alkalmazásával korlátozni lehetne a tanulók által egy kérdésre fordítható idejét például azáltal, hogy a kiszabott idő leteltével a program automatikusan a következő kérdésre ugrana.

A tananyagok interaktívvá tétele is tovább fejlesztené a programban rejlő lehetőségeket. Meg lehetne oldani például, hogy bizonyos diák esetében kérdések és válaszlehetőségek is megjelenítésre kerüljenek, ezáltal a tananyag áttanulmányozása közben is gondolkodásra lehetne késztetni a tanulókat. Be lehetne építeni plusz információkat, a tananyaghoz nem kötelező jelleggel kapcsolódó ismereteket is, amelyeket csak akkor jelenítene meg a program, ha a tanuló érdeklődést mutat, kifejezetten kéri azt.

Ezek mind-mind olyan lehetőségek, amelyek szintén nagyon hasznosak lehetnének, ám ebbe az alkalmazásba igény, idő, vagy valami más híján nem kerültek bele, viszont némi fejlesztői munkával beépíthetőek volnának, ezáltal még hasznosabbá és hatékonyabbá válhatna velük az alkalmazás.

Összegzés

A feladatok végeztével elérkezett az ideje az eddig taglalt fejezetek, gondolatok, és a feladat elvégzése közben szerzett tapasztalatok összegzésére. Úgy gondolom, hogy az eddigi fejezetekből kellőképpen fény derült arra, hogy mennyire összetett és sokrétű egy konkrét feladat, egy olyan probléma megoldása, mint például egy ilyen oktatóprogram szerves és kliens részének megírása. Nem elég, ha csupán azzal vagyunk tisztában, hogy mi az, amit végeredményül szeretnénk kapni. Szükség van arra is, hogy ismerjük azon lépések sorozatát is, amelyeket követve eljuthatunk a célunkig, vagyis a majdan használatba veendő alkalmazásig.

Egy számítógépes alkalmazás fejlesztése számos lépésből áll, amelyek mindegyike igen lényeges, vagyis időt kell rá szánnunk, hogy mindegyiket érinthessük. Ezen lépések végigjárásával eljutottunk egy olyan alkalmazás megvalósításához, amely végső soron megfelelően kiszolgálja azokat az igényeket, amelyeket a fejlesztés kezdeti szakaszában magunk elé állítottunk.

Fontos érzem tanulságként megjegyeznem, hogy egy alkalmazás implementálása során nem az implementálásra fordított idő az, ami a legnagyobb hányadát kiteszi a munkának. Sokszor órákig, s időnként napokig is eltart, mire egy feladatot sikerült annyira körbejárnom, hogy átláttam az előttem nyitva álló lehetőségeket, s azok közül ki tudtam választani az általam legjobbnak tűnő megoldást. S miután megismerkedtem egy-egy technológia, eszköz működésével, szinte pillanatok alatt meg tudtam oldani az addig hegyként tornyosuló problémát. Vagyis fontos időt szánni arra, hogy megismerjük a lehetőségeinket, megpróbáljuk azokat beleképzelni a már elkészült funkciók sorába, hogy aztán kiválaszthassuk azt, amelyik a leginkább beleillik a már készülőfélben lévő rendszerbe.

Jó például szolgált erre a tananyagok kérdése. Igen sok időmet lekötötte, hogy kitaláljam, milyen struktúrát lenne érdemes használnom annak érdekében, hogy a tananyagba be lehessen építeni a változatos matematikai témakörök mindegyikét, ugyanakkor pedig nagyon egyszerű legyen összeállítani őket, hogy egy csekély informatikai ismeretekkel rendelkező személy számára se okozzon különösebb problémát azok összeállítása. S miután már számtalan megoldási módot elvetettem, s igen sok időt veszítettem a megoldás keresésével, egyszer csak sikerült rátalálnom az

alkalmazásban használt módra, amellyel aztán végül sok időt sikerült megtakarítanom, s el tudtam érni a fentebb kitűzött célokat is velük kapcsolatban.

Végezetül meg kell említenem, hogy bár az elkészült alkalmazás megfelel azoknak a céloknak, amelyeket kitűztem magam elé, mégis úgy érzem, hogy nem sikerült kiaknáznom minden benne rejlő lehetőséget. A fejlesztési lehetőségek, mint minden alkalmazás esetében itt is jelen vannak, s azokat megvalósítva tovább lehetne csiszolni a programot, hogy még értékesebb segédeszköz váljon belőle.

Irodalomjegyzék

[1.] Vég Csaba: Instant Java/Java EE/SOA, Logos 2000 Bt, 2007, 311.oldal

[2.] http://www.inf.u-szeged.hu/~bilickiv/h_op/halozat_java.html
(Utolsó látogatás dátuma: 2010. április 22.)

[3.] http://www.cab.u-szeged.hu/WWW/java/kiss/network.html
(Utolsó látogatás dátuma: 2010. április 22.)

[4.] http://www.eclipse.org/articles/Article-Image-Viewer/Image_viewer.html
(Utolsó látogatás dátuma: 2010. április 24.)

[5.] Vég Csaba: Instant Java/Java EE/SOA, Logos 2000 Bt, 2007, 230.oldal

[6.] http://www.java-samples.com/showtutorial.php?tutorialid=384
(Utolsó látogatás dátuma: 2010. április 24.)

[7.] Czapáry Endre: Matematika I., Nemzeti Tankönyvkiadó, 1996

[8.] GeoGebra, geometriai szerkesztőprogram

[9.] Brian Goetz, Joshua Blotz, Joseph Bowbeer, David Holmes, Doug Lea, Tim Peierls: Párhuzamos Java-programozás a gyakorlatban, Kiskapu kiadó, 2009

[10.] Dirk Louis, Peter Müller: Java – belépés az Internetprogramozás világába, Panem, 2002

[11.] Csizmazia Balázs: Hálózati alkalmazások készítése, Kalibán, 1999

[12.] http://java.sun.com/docs/books/tutorial/networking/index.html
(Utolsó látogatás dátuma: 2010. május 2.)

[13.] http://www.davidreilly.com/java/java_network_programming/
(Utolsó látogatás dátuma: 2010. május 2.)

[14.] http://www.ibiblio.org/java/books/jnp/javanetexamples/index.html
(Utolsó látogatás dátuma: 2010. május 2.)

[15.] http://home.cogeco.ca/~ve3ll/jatutorg.htm
(Utolsó látogatás dátuma: 2010. május 2.)