

TARTALOMJEGYZÉK

Bevezetés	2
A számítógépes grafika az informatika oktatásában (Tanári szakdolgozat)	4
A grafika oktatásának előnyei és hátrányai más tárgyakkal szemben	8
A tantárgyban megbúvó lehetőségek	9
A grafika kapcsolata más tantárgyakkal	10
A használt programokról egy pár szó	12
Technikai dolgok	13
FELADATOK	14
I. Elméleti kérdések és feladatok	14
II. GIMP - feladatok	15
III. Flash - feladatok	16
IV. Java - grafika	17
MEGOLDÁSOK	18
I. Elméleti kérdések és feladatok	18
II. GIMP - feladatok	28
III. Flash - feladatok	37
IV. Java - grafika	51
Összefoglalás	70
Köszönetnyilvánítás	72
Irodalomjegyzék	73

Bevezetés

Szakdolgozati témám választásakor két dolog vezérelt. Az első az volt, hogy olyan munkát készítsek, melyet a későbbiekben a tanítás során is hasznosan forgathatok. Ez a magyarázata a feladatgyűjtemény választásának.

A másik ok pedig konkrétan maga a grafika. Azt hiszem, ez az egyik leglátványosabb, legizgalmasabb számítógépes alkalmazási terület. (Persze ezen lehet vitatkozni, de nekem az.) Tizenhatodik éve tanítok informatikát, így sok témakörrel hozott össze a sors. Ezek közül egyik kedvencem a programozás, másik a grafika. (Dolgozatom IV. fejezetében ötvözöm is e két területet.)

Mivel középiskolában dolgozom, ezért a feladatok is ezt a szintet célozzák meg. Vagyis a Kedves Olvasó ne várjon tudományos újdonságokat dolgozatomtól!

Maga a grafika témaköre (is) rettentő összetett, nagyon sok szegmense van. Számítógépi grafikáról nem beszélhetünk addig, míg az elméleti alapokat le nem rakjuk.

Szakdolgozatom 4 témakörre osztható. Feladatgyűjteményem első fejezetében az elméleti alapismereteket foglalom össze, természetesen kérdések, feladatok segítségével. Az elméleti alapok kérdései az általam tanított, a középiskolában leggyakrabban előforduló témaköröket próbálja átfogni.

A második részben olyan feladatokat válogattam, melyeket az egyre népszerűbbé váló pixelgrafikus képszerkesztő és rajzoló program, a GIMP segítségével oldható meg. Ez a program a nagy tudása mellett az ingyenességével és a magyarországi két legnagyobb számítógépes platformra való verziójával nyeri el egyre többek tetszését. Ja, és magyar nyelven is elérhető!

A harmadik részbe olyan feladatokat gyűjtöttem össze, amelyek a szintén nagyon népszerű, – igaz nem ingyenes – FLASH programmal oldhatók meg. Ezen feladatok némelyikének megoldásához egy kis programozási ismeret is szükségeltetik.

A negyedik rész egy kicsit kilóg a sorból. A középiskolai grafika, mint tantárgy oktatásába szerintem a Java grafika nem fér bele. Viszont a programozás témakörében én nagyon fontosnak tartom a grafikai feladatokat. Nemcsak azért, mert a középiskolai matematikát is kiválóan lehet vele föleleveníteni, gyakorolni, hanem mert úgy látom, hogy a diákokat ez villanyozza föl a legjobban. Választhattam volna Pascal grafikát is, de (elnézést a

nagyképűségért!) az nem lett volna kihívás. Mivel a Java programozási nyelvvel az egyetemi tanulmányaim során találkoztam először, úgy gondoltam, hogy a programok „kiszívásával” ragad majd rám valami. Nos, azt hiszem, néhány rajzoló feladattal jövőre biztosan megörvendeztetem tanulóimat.

Nem törekedtem a grafika összes ágának lefedésére. A feladatok a tanítási tapasztalataim alapján készültek. Természetesen ezért előfordulhat, hogy grafikát tanító kollégáim bizonyos feladatokat túl egyszerűnek, vagy éppen túl bonyolultnak, esetleg nem a középiskolai grafika tananyagába illőnek tartanak. Ezt a lehetőséget vállalva, mégis úgy gondolom, értekezésem tekinthető talán majd egy olyan alapnak, amire építeni lehet az iskolákban.

A feladatok sorrendjének megválasztásakor törekedtem a fokozatosságra, illetve az egymásra építhetőségre. Ez egyrészt jelenti azt, hogy egy későbbi feladatban már ismertnek feltételezek olyan dolgokat, melyeket egy előző feladat megoldásakor fölhasználtunk, másrészt pedig azt, hogy a feladatok egyre nehezebbek, összetettebbek.

Szeretném hangsúlyozni, hogy a feladatgyűjtemény nem teljes, de azt hiszem, hogy egy ilyen munka soha nem is lehet az. A feladatok megválasztása is szubjektív. Véleményem szerint ez így van rendjén. Ugyanakkor szívből remélem, hogy lesznek, akik haszonnal fogják olvasni a következő oldalakat!

Sőt, abban is bízom, hogy lesznek a dolgozatomban olyan elemek is, amelyek új ismereteket nyújtanak majd!

A számítógépes grafika az informatika oktatásában

(Tanári szakdolgozat)

A XX. század végén, a XXI. század elején az emberiség igen jelentős társadalmi, gazdasági és politikai változáson ment és megy keresztül. Tagadhatatlan, hogy ezt elsősorban a számítógép és a hozzá oly szorosan kapcsolódó informatika generálja. Magyarország ilyen szempontból is szerencsés, hiszen az e téren végbemenő változásokban lépést tudunk tartani a legfejlettebb társadalmakkal.

Az előbb leírt folyamatnak szoros velejárója, hogy az oktatásnak is reagálni kellett és folyamatosan kell a kialakult helyzetre, s nem szabad szemtől hunynia a változások fölött. Ez érzékelhető abban, hogy mára a számítástechnika – informatika páros minden iskolába oktatott tantárgyként is beköltözött.

Az e területen folytatott tanítás alapvetően más, a hagyományostól eltérő, néha merőben új oktatási módszereket igényel. Itt nem elsősorban a technikai eszközökre, számítógépekre és a hozzá kapcsolódó kellékekre gondolok, hanem a pedagógusi szerepkör megváltozására is.

Az informatika rohamos fejlődésével szinte lehetetlen lépést tartania egy hétköznapi halandónak. Törekedni viszont kell rá. Nem lehet ma már csak DOS-t tanítani. A jó tanárnak szakmailag megingathatatlan tudással kell rendelkeznie. Nem halálos bűn, ha a legeslegújabb technikákkal, megvalósításokkal nincs tisztában, nem ismeri az adott szoftver legfrissebb verzióját, de megbocsáthatatlan, ha nem is próbál haladni a korrallal. Itt nem lehet, mint pl. a matematikánál – és most senkit sem akarok megsérteni – 10-20 évig abból megélni, amit az ember a középiskolában, egyetemen tanult. Hol van már az, mikor én 1997-ben informatika tanári diplomát szereztem Nyíregyházán? Mit tanítottam akkor? És most? De elég csak azt megnézni, hogy mit tanítottam 4-5 évvel ezelőtt. Ezért tartom ijesztőnek azt, hogy a tanárok kötelező óraszámát az 1993-as pályakezdésem óta már heti 4 órával emelték meg.

De térjünk vissza a pedagógiai módszerekre! Azt hiszem, az informatika tárgynál mindenféle megbotránkoztatás nélkül elmondható, itt nem szégyen, ha a diák tanít meg valamit tanárának. A mai gyerekek lassan többet ülnek a számítógép, mint a tévé előtt, pedig sajnos az az idő sem kevés. Így természetes, hogy bizonyos dolgokban nagyobb jártasságra tesznek szert, mint oktatójuk. Ez egy új felállás lehet, amely a tanár-diák kapcsolatra is jó hatással van, ha a tanuló valamilyen plusz ismerettel rendelkezik tanárához képest, s ezt meg is oszthatja

nevelőjével, diáktársaival. Az viszont szégyen bármely tanárra nézve, ha ez a fordított szereposztás rendszeressé válik.

Az teljesen nyilvánvaló, hogy aki jól akar tanítani, annak tisztában kell lenni azzal, hogy *kiknek, mikor, hol, mit*, milyen ismereteket akar átadni és mindezt *mivel*, milyen eszközökkel teszi, teheti meg. Nézzük sorban ezeket!

Kiknek?

Nem mindegy, hogy felnőttek oktatásáról van-e szó, avagy másik végletként esetleg óvodásoknak akarunk informatikát tanítani. Nem hiszem, hogy részleteznem kellene a kettő közti óriási különbséget. Mivel én középiskolában tanítok, s szakdolgozatomban is elsősorban az idejáróknak való feladatokat gyűjtöttem össze, ezért az életkort már be is határoltam. Igen ám, de az életkor nem minden. Ezzel még nem tisztáztam a „*kinek*” kérdését.

Tudni kell, hogy iskolámban nem kimondottan az ún. elité a főszerep. Remélem, sikerült finoman fogalmaznom. Arról van szó, hogy nálunk nincs felvételiztetés, így bárki bekerülhet akár az informatika szakmacsoportos osztályunkba is. Elő is fordult úgy 6 évvel ezelőtt, hogy az osztályomnak két olyan gyerek is tagja lett, akik SNI-s osztályból jöttek. (SNI = sajátos nevelési igényű) Nos, nekik bármilyen számítógépes grafikával (nem is beszélve most a programozásról) kapcsolatos feladat kihívás volt. Ez persze egy végletes példa, de az oktatásban ehhez hasonló eset újra megtörténhet.

A „*kinek*” kérdésre adott válasz során bátran kijelenthetem, hogy nem mindegy például egy osztály összetétele sem. A tanár feladata az év elején minél hamarabb fölmérni a helyzetet, s aszerint tervezni az egész évi munkát. Természetesen kell, hogy legyen ekkorra egy előzetes elképzelés és egy olyan minimum szint, amelyből már nem lehet engedni, de a „többet és még többet megtanítani” mindig ott kell, hogy legyen egy pedagógusban. Itt a **megtanítani** szóra helyezném a hangsúlyt. Vallom, inkább kevesebbet tanítsunk, de azt sajátítsák is el a tanulók, méghozzá úgy, hogy a későbbiekben alkalmazni is tudják a tanártól kapott ismereteket, módszereket.

Mikor?

Ez sem lényegtelen kérdés. Tudja mindenki, hogy teljesen más egy 1., megint más egy 3., és megint más egy 8. óra. Hacsak tehetjük, a nap vége felé inkább a gyakorlaté legyen a főszerep, mintsem az elméleté! Persze, egy pezsdítő testnevelés óra után következhet az elmélet.

A grafika minden évfolyamon szóba jöhet, de például az ellipszist megrajzoló program elkészítését nem várhatjuk el egy 9. évfolyamos gyerektől. Ehhez mindenképpen szükség van a koordináta-geometriai ismereteire, amit viszont csak a 11. évfolyamon kap meg matematikából.

Nem érdemes a Flash oktatásába sem belevágnunk, míg a tanulók nincsenek tisztában a rasztergrafika és vektorgrafika közti különbséggel. Mindenképpen érdemes először egy pixelgrafikus képszerkesztő programmal megismertetni őket, mert ott kiderülnek az ilyen programok hátrányai, lehetőségének korlátai, és persze előnyei is.

Hol?

Nyilván az iskolában. Igen ám, de a számítógépes szaktantermek sem mindig állnak rendelkezésre, mert mondjuk egy másik csoportnak van ott órája. Egyértelmű, hogy ilyenkor számítógépes feladatmegoldásról igen nehéz beszélni. Marad az elmélet, melyet szerintem mindig lehet csiszolni, és vallom, hogy nagy szükség is van rá a későbbi komolyabb munkákhoz. Tehát az óra- és teremrend ismeretében szükség lehet a tanmenet átdolgozására is, hogy ilyen órákra kerüljenek az elméleti anyagok ismertetései.

Mit?

Talán ez a legnehezebb. Fontos, hogy mire fogják fölhasználni a kapott ismereteket a tanulók és fontosak az előzőekben felsoroltak. A tantervek és a vizsgakövetelmények sok szabadságot adnak a tanárnak, aminek vannak előnyei és hátulütői is.

Mindenképpen előny, hogy így nincs a szoftverek között egy üdvöske. Nem is lehet előírni azt egy iskolának sem, hogy fizetős szoftverekből mindig a legfrissebb verziókkal rendelkezzen. Akkor jönnek a nyílt forráskódúak? Nem szeretnék állást foglalni e témában, de tény, hogy sokan vannak, akik a kereskedelmi licenszelésű szoftvereket részesítik előnyben. Őket miért kényszerítenénk egy másfajta program használatára? Egyáltalán, annyi jó szoftver van ma már, hogy nagy hiba lenne csak egynél leragadni.

További előny ebből következően az, miszerint így a pedagógus rá van kényszerítve arra, hogy ne egy program használatát sajátíttassa el, hanem problémamegoldást tanítson. Tehát szerintem teljesen rendben van, ha nincs meghatározva, hogy milyen szoftverrel kell oktatni. Még akkor is jól van ez így, ha az előnyből gyakran hátrány keletkezik. Egy vizsgán föl kell készülni arra, hogy – akár ugyanarra a problémakörre – néha 4 szoftvert is a vizsgázók rendelkezésére kell bocsátani.

Mivel?

A szoftver kiválasztása nyilván csak azután következhet, ha tisztáztuk az elsajátítandó ismereteket, készségeket. Ezt nagyvonalakban a tantervek tartalmazzák. Illetve az érettségizőknek ott van még az „Érettségi vizsgakövetelmények” c kiadvány. Ez utóbbit lásd a [9] internetes elérhetőségen!

A programok kiválasztásánál mindenképpen elsődleges szempont az legyen, hogy mennyire fogja majd segíteni, támogatni az elérendő célt. Másik szempont, mennyire egyedi az adott program. Ami olyan készségek meglétét feltételezi, illetve olyan készségek kialakulását segíti elő, melyekre a számítástechnika más területén nincs vagy csak kevésbé van szükség, nos akkor az a program nem lesz jó. A középiskolákban nem speciális ismereteket kell nyújtunk.

Szakdolgozatom feladatait, reményeim szerint, nemcsak a GIMP-pel való megoldásra lehet fölhasználni. Természetesen a megoldások egyes lépései változhatnak, ha más szoftverrel akarjuk elkészíteni a feladatot, de a lényeg, – melyet én is szeretnék diákjaimnak közvetíteni – az nem a lépések megtanulása, hanem a lehetőségek fölillantása. Vannak olyan kollégák, akik még ma is abból íratnak dolgozatot, hogy az adott program valamelyik menüjében milyen parancsokat lehet megtalálni. Nos, szerintem az ilyen módszer, előbb-utóbb megutáltatja a tanulókkal az adott tantárgyat. A lényegét én abban látom, hogy tudnia kell a gyerekeknek megtalálni a megfelelő eszközt, s nem bebifláznia. Ha sokat fogja használni a programot sem biztos, hogy vissza tud arra emlékezni, melyik utasítás hol található, de ha leül a gép elé, akkor pillanatok alatt képes előhalászni az adott eszközt. Nagyon sokszor tapasztalhatjuk azt is, hogy egy jól bevált, sokat használt alkalmazásnak újabb kiadása jelenik meg, s szép lassan (vagy gyorsan) elkezd terjedni ez az új verzió. Ilyenkor mit sem érhet a bemagolt ismeret, mert lehet, hogy bizonyos eszközöket a program megalkotói más helyre pakoltak az adott szoftverben.

A grafika oktatásának előnyei és hátrányai más tárgyakkal szemben

Azt hiszem, a számítógépes grafika, mint tantárgy nagyon sok jó tulajdonsággal rendelkezik.

- Rögtön az első, hogy igen látványos tud lenni.
- A második, hogy könnyen sikerélményhez juttathatja a diákokat, s mint tudjuk, ez inspirálja majd őket a továbbhaladásban. Például a GIMP-pel való rajzolás megismerése után, – mikor már a GIMP más pozitív tulajdonságai is felszínre kerültek – nem egy diákom e-mail-ben kérte tanácsomat, hogy az otthoni, maguktól kitalált munkájukban segítsék. Ez a csoda más tantárgynál ritkán következik be. Másik példa, hogy a mostani 12. évfolyamosok rendszeresen érdeklődnek arról, amit a 13. évfolyamos leendő webmesterekkel tanulunk.

Ezek nagyon fontos visszajelzések számomra, csakúgy, mint mikor a tanítási órák során valami kis boldogságot vélek fölfedezni tanítványaim arcán. Sokszor látom, valami megcsillan a szemükben. A grafika oktatásakor ezek eléréséhez nem is kell olyan sok.

- A harmadik jó tulajdonság, hogy bizonyos speciális matematikai ismeretektől eltekintve, akár már a 9. évfolyamon is elkezdhetjük oktatni.
- A negyedik előnyös dolog a számítógépes grafika oktatásában, hogy a növendékek az itt szerzett tudást nagyon sok területen föl tudják használni, azaz látják a hétköznapi hasznát.
- Az ötödik pozitív vonása, – mely már inkább a tanár szemszögéből nézve az – hogy a diákok által oly becsben tartott puskázás lehetősége a gyakorlati feladatoknál minimális, hiszen nagyon sok feladatnál nem is lehet két egyforma megoldás.

Azt nem tudom, melyik jó tulajdonsághoz kellett volna besorolnom, hogy a grafika órákon nagyon ritkán fordul az elő, miszerint a diák nem azzal foglalkozik, amit kértem tőle. Ilyenkor nincs a máskor oly gyakran előforduló internetezés vagy játék. Teljesen leköti őket a munka.

Mivel „nincsen rózsza tövis nélkül”, ezért a tantárgyból adódó nehézségekről is ejteni kell pár szót.

- Egyik ilyen, hogy elég nehéz az objektív tanulói értékelés. Az elmélet számonkérésekor nincs gond a tárgyilagossággal, de a gyakorlati feladatok osztályzásakor már sokkal nehezebb korrekt módon minősíteni a munkákat, szülehetnek sajnos nem teljesen elfogulatlan érdemjegyek is. Ügyelni kell, hogy ne mindig a külalakot nézzük csak, hiszen egy-egy feladat értékelésénél azt kell figyelniünk, mennyire tudta a tanuló alkalmazni az ismereteit. Azt, hogy mennyit „pepecselt” egy adott alakzat megrajzolásával, ne díjazzuk,

vagy éppen ne büntessük! Talán az a legszerencsésebb, ha számonkéréskor előre megadott állományokkal dolgoznak a diákok, már persze, ha a feladat jellege ezt megengedi.

- A grafika tanítása egy másik szempontból is mindenképpen keményebb dió, mint más informatikával kapcsolatos tantárgyaké. Ez a szempont pedig a tanulói hibák korrigálása. Itt a tévesztések javítása jóval nehezebb, összetettebb. Például a programozásnál egy pár éves tapasztalat után sokszor a forrásba való belekukkantással könnyű észrevenni a hibát, míg a grafikában sokszor csak 58 művelet után szól a tanuló, hogy elakadt, vagy valami nem úgy van, ahogy azt szeretné. Tény ugyanakkor, hogy a grafika látványosabb, s a gyengébb képességű tanulók is tudnak sikereket elérni. A programozásról ez nehezen mondható el.

- A harmadik gondot az okozhatja a tantárggyal kapcsolatban, hogy eléggé szoftverhez kötött. Ez csak akkor gond, ha pl. egy olyan szoftverrel oktatjuk a tárgyat, amelynél nincs esély arra, hogy legalísan legyen meg a gyerek otthoni gépén. Ilyen pl. a Macromedia Flash 8, melyet iskolánk 20 licensszel vásárolt meg, de a hallgatóknak otthon maximum csökkentett módú demó verziójú programjaik lehetnek. Akkor miért ezt a szoftvert választottam? Mert aki a web világában akar a kor követelményeinek, szintjének megfelelőt alkotni, annak ismernie kell ezt a programot. Sajnos itt nem volt alternatíva.

- Még egy problémás dolgot említenék. Nevezetesen, hogy néha bizony hajlamos az ember a sok lehetőség között elveszni. Annyi minden jó dolgot lehet ezen szoftverekkel csinálni, hogy az ember képes belefeledkezni, s már nem is a tananyagnál járunk. De nagyon sokszor maguk a diákok állnak elő egy-egy otthon felmerült problémával. Ez pedig már újra egy nagyszerű dolog.

A tantárgyban megbúvó lehetőségek

A számítógépi grafika a látásra épül. Mindenképpen foglalkozni kell hát az emberi látás sajátosságaival. Nagyon gyakran csapjuk be látószervünket különféle illúziókat keltve. Ezek néha tréfások, de néha végzetesek is lehetnek. A szem roppant fontos érzékszervünk, s pont ezért nagyon-nagyon vigyáznunk kell rá. Meg kell hát tanítani a diákoknak azt, hogy egy esetleges hosszabb lélegzetvételű számítógépes munka során, – ami a grafikai feladatoknál nem ritka – hogyan tudják pihentetni szemüket, illetve azt, hogy miként lehet a munkát jól megszervezve az elkészítéshez szükséges időt lerövidíteni.

Meg kell tanítani, hogy a lehetőségek garmadájában ne vesszenek el. Gyakran sokkal jobb, szebb az egyszerűbb, puritánabb grafika, mint az agyoncicomázott, mindenféle extrával

felruházott, viszont a lényegét elrejtő kép. Itt lehet kapcsolatot keresni a rajz és vizuális kultúra tárgygal.

A grafika témakörbe szerintem nagyon szépen be lehet csempészni a számítógépes adatbiztonság témakörét és a titkosítást. Ez utóbbihoz szorosan kapcsolódik a következőkben említett szerzői jogok kérdése is.

Tisztában kell lennie a tanulóknak (is), hogy az Interneten található képek sokasága nem biztos, hogy szabadon felhasználható, mondjuk egy saját grafikai munka elkészítésekor. A képek tartalmazhatnak olyan elsőre nem látható információkat is, mint pl. amilyen egy ún. vízjel. De persze különböző titkosítási eljárásokkal olyan információ is fölvihető egy képre, amely láthatatlan és semmi köze magához a képhez. Ez mindig eléri hatását, döbbenet szemléljük, ahogy egy képben valamiféle szöveget rejtek el.

Még két nagyon fontos területét említeném meg az informatika, azon belül pedig a grafika oktatásának. Ezek pedig a már fentebb említett szerzői jogok kérdése és a Netikett.

A szerzői jogokról szerintem méltatlanul kevés szó esik az oktatásban. Sokszor dicsekszenek a gyerekek, hogy ilyen és ilyen programjaik vannak, és ilyen meg ilyen zenét hallgatnak, filmet néznek otthon. Egy negyedórás beszélgetéssel föl lehet nyitni a szemüket e témát érintően is. Nagyon sokszor a tanártól hallanak erről először. Mindig meghökkennek, hogy törvénytelen lehet akár egy darab fájl illegális letöltése is.

A Netikett is egy olyan téma, melyről még a végzős diákok sem mindig tudják, hogy mit is értünk alatta. A legtöbbször elfogadja ezen szabályokat. Sőt, nemcsak hogy elfogadják, hanem követendőnek is tartják, mert tudják, hogy az emberi érintkezésnek – történjen az akár számítógép, ill. valamilyen más elektronikus kommunikációs eszköz segítségével – megvannak az írott és íratlan szabályai. Persze, mint az élet más területén, itt is voltak, vannak és lesznek olyanok, akik semmibe veszik ezen szabályokat. Nekünk, pedagógusoknak az egyik feladatunk harcolni azért, hogy minél kevesebben legyenek, akik így gondolkodnak.

A grafika kapcsolata más tantárgyakkal

Maga a számítógépes grafika nemcsak a ma oly divatos multimédia, de a számítástechnika többi területén is nagyon fontos szerepet tölt be. A grafika, a képi megjelenítés mindenhol jelen van környezetünkben. Az ember vizuális úton szerzi ismeretei döntő részét. Tanítási tapasztalatból tudom, hogy a tanulók az ilyen információt jegyzi meg a legjobban, a képi információk vannak rájuk a legnagyobb hatással. Az informatika oktatásán

belül nagy hangsúlyt kell helyezni arra, hogy a diákok minél szélesebb körű ismeretekkel rendelkezzenek a számítógépes grafika területén.

Az informatikai tárgyak közül nem is tudok olyat mondani, ahol valamilyen szinten ne esne szó a grafikáról. Napjaink informatikai alkalmazásai rendkívüli módon előtérbe helyezik a grafikus felhasználói felületre épülő rendszerek gyakorlati alkalmazását. Mindez megmutatkozik abban is, hogy az informatika érettségi vizsgán is külön témakört kapott a számítógépes prezentáció és grafika.

Mivel a számítógépes világ elsődleges outputja ma a képernyő (de akár lehet az a nyomtató is), ezért a rajta előállított kép milyensége, illetve az, hogy az a látvány miként is jön létre, mindig téma. Nagy segítség lehet a sokak által kárhozott számítógépes játékok tömkelege. Igaz, ilyen informatikai tantárgy a diákság legnagyobb bánatára nincs, viszont a grafika témakörébe nagyon jól becsempészhetők a játék közben szerzett tapasztalatok. (Akár tetszik nekünk, akár nem, ez ma már egy külön iparág, s a tanulókat nem lehet a „fogyasztásáról” letiltani.)

Írásomban már többször említettem a matematikát. Nem véletlen, hiszen a gépi grafika megvalósítása, kivitelezése kemény matematikát kíván. Középiskolában értelemszerűen csak a középiskolai matematikára tudunk támaszkodni. A hatás e tantárggyal kölcsönös lehet, lásd a feladatgyűjtemény IV. fejezetét, ahol a Java nyelvet hívom segítségül koordináta-geometriai problémák megoldásához.

A biológiával való kapcsolatra is felhívtam már a figyelmet, itt csak ismételni tudom magam. Az emberi látás sajátosságai nagyban befolyásolják azt, hogy miként kell bizonyos dolgokat élethűen megjeleníteni.

Szoros a kötelék a grafika tantárgy és a rajz és vizuális kultúra, valamint a médiaismeretek tantárgyakkal is. Mivel mindegyik a látásra, illetve a látványra épít, ezért nem meglepő a szoros kapcsolat.

Összefoglalva az eddigieket, megállapítható, hogy a számítógépes grafika az iskolákban is elkezdett teret nyerni, mára már nem hagyható ki egyetlen számítástechnikai könyvből sem. Nagyon sok helyen külön tantárgy is lett belőle. Nem lehet megkerülni a témát akkor sem, ha a grafikával kapcsolatban sokszor csak a legszükségesebb, legelemibb képi dolgokra jut idő és lehetőség az informatikán belül.

A használt programokról egy pár szó

A GIF formátum nagyon jó a kisméretű (5-10 képkockából álló) és a kevés színt használó animációkra. Ilyenek szerkesztésére nagyon sok ingyenes program létezik, de remek szolgálatot tesz e célra is a GIMP.

A Flash viszont már egy teljesen más dolog, az előzőeknél jóval többet kínál e tekintetben.

A Flash-ben ha megadjuk a kezdő kockán a kiindulási pontot, a végsőn pedig a végpontot, a program kiszámolja és megrajzolja a mozgást. Ezt pedig még rengeteg effekttel meg lehet bolondítani. Például egy objektum a helyváltoztatása közben ugrálhat, csavarodhat, pattoghat, stb. Mindezekből következően nem véletlen, hogy a Flash-t elsősorban webes környezetben használják, mivel kis mérete ellenére nagyon látványos. (Klasszikus példa ma már erre a weboldalakon látható animált reklámok tömkelege.) Apró méretét annak köszönheti, hogy vektorgrafikus leírást használ.

A Flash-ben a különböző objektumokat programozni is lehet, ami annyit tesz, hogy nagyon sokrétűen meg tudjuk mondani egy alakzatnak, hogy mikor, mit és hogyan tegyen. Ez lehetőségeink tárházát jelentősen bővíti.

Tehát ha az animáció kicsi és egyszerű, akkor GIF-et, ha összetettebb, akkor a Flash-t érdemes használni.

A GIMP - feladatokat a **GIMP 2.4.7**-es verziójú, Windows platformra megjelentetett magyar nyelvű változatával alkottam meg. A megoldásoknál ezen program menüpontjainak, parancsainak neveit, leírásait használtam.

A következőt a program Súgó → Névjegye → Licenc pontjából idézem:

„A GIMP szabad szoftver: terjesztheti és/vagy módosíthatja a Free Software Foundation által kiadott GNU General Public License második (vagy bármely későbbi) változatában foglaltak alapján.”

A Flash – feladatok megoldását a **Macromedia Flash Professional 8** programmal szerkesztettem, mely programot iskolánk megvásárolta.

A Java nyelven történő programozást a szintén ingyenes **JCreator 4.00 LE** (version 4.00.26) nevű szoftverrel készítettem el. Választásomat az indokolja, hogy munkahelyemen is ezzel oktatom diákjaimat, mivel nem akarok elveszni az ettől a programtól jóval professzionálisabb – s ebből következően jóval bonyolultabban használható – szoftverekben.

Technikai dolgok

A feladatok megoldásának leírásai előtt tisztázni kell egy-két dolgot.

Szakedolgozatomban nem térek ki az adott programok alapvető használatára. Feltételezem, hogy a feladatok megoldásakor ezen ismeretekkel rendelkezik az, aki a példák megoldására vállalkozik.

A feladatokhoz tartozó állományok, melyekkel a munkát el kell kezdeni, a szakdolgozathoz mellékelt CD gyökér könyvtárának Nyers nevű mappájában, azon belül pedig vagy a Flash vagy GIMP nevű almappában találhatók. (Az elméleti kérdésekhez és a Java feladatokhoz nincsenek ún. nyers állományok.) Az állományok neve Nyers_Feladat_sb.kit, ahol „Feladat” a feladat típusát (Flash vagy GIMP), „s” az adott feladat sorszámát, „b” pedig az azon belüli betűjelét jelenti, a „kit” a fájl kiterjesztése. (Pl. a GIMP-feladatok közül az elsőhöz tartozik két ún. nyers állomány, melyeknek a nevei rendre Nyers_Gimp_1a.jpg, ill. Nyers_Gimp_1b.jpg.)

Sok esetben a feladatokat körülményes leírni, ezért segít a megértésben, ha megnézzük a kész megoldást. Ezeket – hasonlóan a nyers fájlokhoz – a CD-n lévő Kesz mappában találjuk. Itt viszont már három almappa van – Flash, GIMP, JAVA. Az elnevezések logikája is hasonló. (Vagyis pl. a Flash-feladatok 8. példájának megoldását a CD-n a Kesz\Flash mappa Kesz_Flash_8 fla állományában találjuk.)

Saját tapasztalatból tudom, hogy érdemes a kész munkát az adott program saját formátumában is megőrizni. Ez azért jó, mert ha valamikor is módosítani kell a kész(nek hitt) munkán, akkor ez könnyen megtehető.

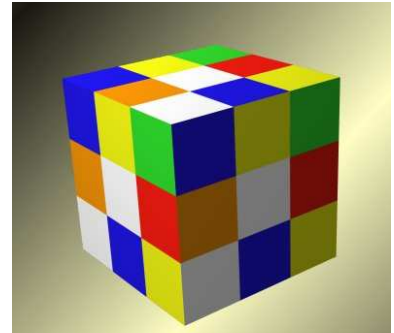
FELADATOK

I. Elméleti kérdések és feladatok

- 1.) Jellemezze a raszter- és a vektorgrafikát!
- 2.) Mekkora annak a képnek a mérete kB-ban megadva, amely 800x600 képpontból áll és 200 színt használ?
- 3.) Döntse el az alábbi állításokról, hogy IGAZ vagy HAMIS?
 - a) A grafikus operációs rendszerek szabványa az X-Window. igaz/hamis
 - b) A számítógépes grafika felhasználási területe a virtuális valóság megjelenítése. igaz/hamis
 - c) A FIREWIRE az univerzális soros busz. igaz/hamis
 - d) Az AGP gyorsított grafikus port. igaz/hamis
 - e) A videokártyák teljesítményét FPS-ban (frame/sec) mérjük. igaz/hamis
 - f) Az LCD monitorok működése a hagyományos televíziókéhoz hasonló. igaz/hamis
- 4.) Röviden foglalja össze az emberi látással kapcsolatos legfontosabb tudnivalókat!
- 5.) Ismertesse a leggyakoribb színkeverési eljárásokat!
- 6.) Mit értünk 3D-2D leképezés alatt?
- 7.) Mi a mintavételezés?
- 8.) Mi a kvantálás?
- 9.) Mi a helyreállítás lényege, mikor van rá szükség?
- 10.) Mi az OpenGL?

II. GIMP - feladatok

- 1.) Tegyen egy nem átlátszó hátterű képet átlátszó hátterűvé!
- 2.) Készítse el az 1. ábrán látható képet! A kész kép szélessége 400, magassága 300 pixel legyen! A munkához használja föl a Nyers_Gimp_2.jpg képet!
- 3.) Egy természeti képre másolja át saját fényképét! (Lásd a 2. ábrát!)
- 4.) Rajzoljon Rubik-kockát a 3. ábrához hasonlóan!



1. ábra GIMP-feladatok_2

2. ábra GIMP-feladatok_3

3. ábra GIMP-feladatok_4

- 5.) Készítse el a 4. ábrán látható képet!
- 6.) Készítsen képet a következő felirattal: „A Juhász család egy boldog pillanata a sok közül”! (Lásd 5. ábra!)



4. ábra GIMP-feladatok_5



5. ábra GIMP-feladatok_6

- 7.) Készítsen egy egyszerű szöveg animációt! (Lásd a 6. ábrát!)



6. ábra GIMP-feladatok_7 (combien és replace)

- 8.) Készítsen képeslapot Hajdúnánásról! (Lásd a 7. ábrát!)
- 9.) Rétegmazsk alkalmazásával készítse el a Nyers_Gimp_9.jpg állományból a 8. ábrán látható képet!
- 10.) Készítse el a 9. ábrán látható képet!



7. ábra GIMP-feladatok_8



8. ábra GIMP-feladatok_9



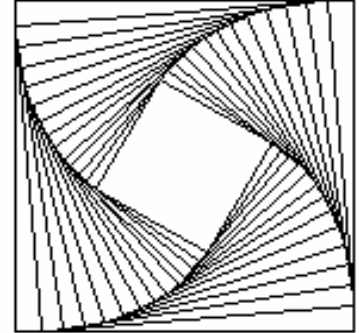
9. ábra GIMP-feladatok_10

III. Flash - feladatok

- 1.) Rajzoljon házikót, fát, Napot, felhőt!
- 2.) Rajzoljon egy pislogó fejet!
- 3.) Rajzoljon egy tetszőleges alakzatot, mely folyamatosan változtatja a színét!
- 4.) Készítsen animációt, melyben a Hold Nappá változik!
- 5.) Mozgasson egy tetszőleges alakzatot egy előre megrajzolt pályán!
- 6.) Készítsen animációt, melyben egy képnek fokozatosan egyre nagyobb része válik láthatóvá!
- 7.) Rajzoljon egy gombot, mely megváltoztatja színét/alakját/feliratát, ha az egeret rámozgatják!
- 8.) Készítsen egy mozgó reklámcsíkot (bannert)!
- 9.) Rajzoljon egy tetszőleges alakzatot, majd a fő égtájak felé mutató nyilakat! Érje el, hogy az alakzat a nyilakra kattintva a megfelelő irányba mozogjon!
- 10.) Készítsen olyan Flash-animációt, melyben 3 golyó mozog egyenletesen úgy, hogy az ablak széleinél mindegyik visszapattan!

IV. Java-grafika

- 1.) Rajzoljon koncentrikus köröket a képernyő közepére, váltott színezéssel!
- 2.) Rajzoljon felülnézetből látható piramist!
- 3.) Rajzolja meg a magyar zászlót!
- 4.) Készítsen sakktáblát kirajzoló programot!
- 5.) Készítse el az alábbi ábrát rajzoló programot! (A feladat ötletét [10] oldalról vettem.)
- 6.) Készítsen programot, mely a felhasználótól bekéri egy egyenes egyenletét, majd megrajzolja az egyenest és annak x-tengelyre tükrözött képét a képernyő közepére helyezett koordinátarendszerben!
- 7.) Kérjük be egy háromszög csúcsainak koordinátáit és egy szög nagyságát! Ezek után rajzoljuk meg a háromszöget, majd a háromszög adott szöggel való origó körüli elforgatottját!
- 8.) Készítsen olyan programot, mely kirajzol egy tetszőleges szabályos sokszöget!
- 9.) Megrajzolandó a háromszög köré írt kör, ha a háromszög csúcspontjainak koordinátáit a felhasználótól kérjük be.
- 10.) Írjon parabolát rajzoló programot!
- 11.) Rajzoljon paraméterezhető arkhimédeszi spirált a képernyő közepére! (csigavonal)



10. ábra *Java-grafika_5*

MEGOLDÁSOK

A feladatok megoldásával kapcsolatos technikai dolgok leírását lásd a dolgozatom „Technikai dolgok” című részében! Az elméleti kérdésekről bővebben olvashatunk [3] megfelelő fejezeteiben.

I. Elméleti kérdések és feladatok

1.) Jellemezze a raszter- és a vektorgrafikát!

A digitális kép olyan információk halmaza, amely képpontokból áll. A képpontot nagyon gyakran pixelnek is mondjuk, az angol *picture elements* kifejezésből alkotott mozaikszó után.

Rasztergrafika: (Másképpen bitmap grafika, pixelgrafika.) Ha a képet alkotó minden pontot önállóan, sorról-sorra jellemezzük, akkor raszteres képről beszélünk. A képpontok megadásakor minden egyes pixelhez egy számot rendelünk. Ezen hozzárendelt bináris szám hossza mutatja meg, hogy mennyi színt is tartalmazhat a kép. Egy bit 2, két bit 4, három bit 8, négy bit 16, öt bit 32, ... n bit 2^n szín megjelenítését teszi lehetővé. Tipikus színmélység a 24 bites ($2^{24} \approx 16,7$ millió), illetve a 32 bites ($2^{32} \approx 4,2$ milliárd).

Előnye az ilyen képnek, hogy viszonylag egyszerű adatszerkezettel reprezentálható. Egyszerű algoritmusokkal gyors a feldolgozása és fotótechnikai trükköknél is igen jól alkalmazható. Hátránya, hogy az adatállomány igen nagy méretű is lehet, továbbá hogy rögzített a felbontás és a nagyításnál a minőség jelentősen romolhat.

Vektorgrafika: Ha a képet alkotó pontok halmaza között valamilyen matematikai összefüggést állapítunk meg, és ezen összefüggések jellemzőit el is tároljuk, akkor vektoros képről beszélünk.

Ilyen jellemző lehet pl. egy pont, illetve egy szakasz két végpontjának a koordinátái, vagy egy kör középpontjának a koordinátái és a sugara. Pontok, szakaszok, görbék, felületek, mint elemi geometriai alakzatok megadásával tetszőleges síkbeli vagy térbeli alakzat megadható, fölépíthető. Mivel a kép nem képpontokból áll, ezért (bizonyos határok között) tetszőlegesen nagyítható/kicsinyíthető. A végeredmény minősége csak a képmegjelenítő eszköztől függ.

A vektoros kép általában kisebb méretű és független a felbontástól, mivel az elemi objektumok tetszőlegesen paraméterezhetők, melynek következtében a méret dinamikusan változhat. Hátránya az ilyen képnek, hogy igen összetett lehet az adatszerkezete. Mivel

egészen bonyolult algoritmusokat használhatnak a kép leírásához, jelentős számítási kapacitást igényelhet az előállítás.

2.) Mekkora annak a képnek a mérete kB-ban megadva, amely 800x600 kép-pontból áll és 200 színt használ?

Az ilyen kép $800 \times 600 = 480000$ pixelből áll. Kétszáz különböző érték (itt szín) tárolásához 8 bit szükséges, mert $128 = 2^7 < 200 < 2^8 = 256$. (Tehát ilyen szempontból mindegy, hogy 129 vagy 256 különböző színből áll a kép.) Ez azt jelenti, hogy minden egyes képpont kinézetét 8 biten tudjuk megadni, vagyis a kép mérete $800 \times 600 \times 8$ bit. Mivel $8 \text{ bit} = 1 \text{ Byte}$, ezért a kép $480000 \text{ B} = \underline{480 \text{ kB}}$ méretű. (A „k” 1000-szerest, a „K” 1024-szerest jelent!)

3.) Döntse el az alábbi állításokról, hogy IGAZ vagy HAMIS?

a) **A grafikus operációs rendszerek szabványa az X-Window.** igaz/hamis

Magyarázat: Az X11 szabványt, azaz az X-WINDOW System v. 11-et 1987. szeptemberében adta ki a Massachusetts Institute of Technology. Az X-Window System hardverfüggetlen, a rastergrafikára alapozott ablakozó megjelenítési rendszer.

Egy grafikus munkahely az X11 szerint képernyőt, billentyűzetet és valamilyen mutatóeszközt kezelő rendszer, melyen az X-szerver program fut. Az X-szerver feladata a grafikus megjelenítés és az adatbevitel fogadása. Az X-szerver hálózati kliensekkel, az ún. alkalmazásokkal tarthat kapcsolatot, melyek számára hozzáférést biztosít a munkahelyhez.

Az X11 felhasználói felülete a Windows-ból is ismert ablakozó rendszer. Az ablakok méretezését és általában menedzselését a Window Manager program végzi, mely az X-szerver számára szintén egy kliens.

A mai grafikus operációs rendszerek jelentős többsége ezen szabványt használja. A felület minden esetben objektumorientált, eseményvezérelt. Az objektumok egy üzenetkezelő rendszeren keresztül tartanak kapcsolatot egymással.

b) **A számítógépes grafika felhasználási területe a virtuális valóság megjelenítése.**

..... igaz/hamis

Magyarázat: A számítógépes grafikának nagyon sok felhasználási területe van. Ezek közül néhány:

- Grafikus operációs rendszerek
- Képfeldolgozás

- Szöveg- és kiadványszerkesztés
- Grafikus prezentáció
- Reklám, művészeti grafika
- Térinformatika, térképészet
- Játékok
- Számítógéppel segített tervezés és gyártás (CAD/CAM – Computer Aided Design/ Manufacturing)
- Orvostechika
- Környezet szimuláció
- Virtuális valóság

c) **A FIREWIRE az univerzális soros busz.** igaz/hamis

Magyarázat: **Grafikus kommunikációs vonalak, busztípusok**

A képi információk adatátviteli követelményei miatt meg kellett növelni a számítógépes átviteli vonalak sebességét, mivel a régiek már nem voltak képesek megfelelő sebességgel ellátni grafikus adatokkal a feldolgozó programokat. Új busztípusok jelentek meg, melyet a grafika mellett a nagy adatmennyiséget feldolgozó új perifériák (pl. digitális videokamera, fotónyomtató, stb.) terjedése is szükségessé tett.

A számítógépes grafika szempontjából az új kommunikációs vonaltípusok közül az USB, a FIREWIRE és az AGP a legfontosabbak.

USB (Universal Serial Bus = univerzális soros busz): Az USB egy olyan szabványosított csatlakozóaljzat és összeköttetés, amely a billentyűzetcsatlakozót, az egércsatlakozót és a soros valamint a párhuzamos portot egyetlen nagysebességű, soros átvitelt biztosító összeköttetéssel helyettesíti.

Legfontosabb tulajdonságai a következők:

- 12 Mbit/sec átviteli sebesség,
- maximum 127 USB eszköz kiszolgálása,
- soros adatátvitel,
- lehetővé teszi a plug and play perifériatelepítést,
- az eszközök tápellátása USB kábelén keresztül lehetséges.

FIREWIRE: A FIREWIRE létrejöttének oka, hogy a multimédiában és grafikában egyre nagyobb szerephez jutnak a közepes teljesítményű átvitelt igénylő perifériák. Ilyenek pl. a videokamerák, a nagy felbontású nyomtatók, a nagy kapacitású streamerek.

A FIREWIRE jellemzői a következők:

- átviteli teljesítmény 400 Mbit/sec,
- max. 16 eszköz kezelését teszi lehetővé,
- a plug and play telepítést támogatja.

AGP (Accelerated Graphics Port = gyorsított grafikus port): A 3D-s megjelenítés évről évre több memóriát igényelt, mellyel a grafikus kártyák kapacitása egyre nehezebben tudott lépést tartani. Az AGP közvetlen összeköttetéssel 266, illetve 533 Mbyte/sec átviteli sebességet biztosít a grafikus kártya és a RAM között (2x, 4x-es AGP).

A kétszeres AGP esetén az új sín átviteli teljesítménye kb. kétszerese a PCI sínének. Ez a busz csak a grafikus adatok átvitelére szolgál, így tovább növelve a grafikus rendszerek teljesítményét.

Az AGP alkalmazásához a RAM-ból természetesen megfelelő mennyiségű memóriát kell lefoglalni. Ez a grafikus hardvertől és az alkalmazástól függően 24–64 Mbyte is lehet.

d) **Az AGP gyorsított grafikus port.** igaz/hamis

Magyarázat: Lásd az előző kérdésre adott választ!

e) **A videokártyák teljesítményét FPS-ban (frame/sec) mérjük.** igaz/hamis

Magyarázat: A videokártyák teljesítményét FPS-ban (frame/sec) adjuk meg, melyen a másodpercenkénti teljes (raszteres) képernyő újrarajzolásainak számát értjük. Ahhoz, hogy folyamatos mozgóképet érzékelhessünk, legalább 25 FPS sebesség kell. A monitorvezérlő kártyák teljesítményében a műszaki paraméterek mellett meghatározóak a drájverek is.

f) **Az LCD monitorok működése a hagyományos televíziókéhoz hasonló.** igaz/hamis

Magyarázat: A monitorokat működési elvüket tekintve 3 csoportra lehet osztani: katódsugárcsőes (CRT), folyadékkristályos (LCD), plazma (PDP).

CRT: A monitorok közül ez terjedt el először. A működés elve röviden: egy elektronágyú által kilőtt elektronsugár soronként végigpásztázza belülről a képernyőt. A monitor belső felületén fénypor (sokszor hibásan foszfornak mondják!) van. Az elektron becsapódásakor ez villan föl. A pásztázás másodpercenként minimum 60-szor megy végbe, így a szemünk összeköti a felvillanásokat, és képet látunk, amit fekete-fehér monitorok esetén a sötét és világos pontok rajzolnak ki.

A színes monitoroknál színkeverést alkalmaznak. A piros (R), zöld (G) és kék (B) színt 3 egymáshoz nagyon közeli pontban hozzák létre, így szemünk összekeveri őket. Ekképpen áll

elő a monitor által megjelenített valamilyen szín. Ehhez a színes készülékek három elektronágyút és háromféle fényport alkalmaznak.

LCD: Itt minden képponthoz 3-3 folyadékkristály és vezérlő tranzisztor tartozik. A folyadékkristály mindig különböző irányban polarizálva engedi át a fényt, így egy folyadékkristály a ráadott feszültségtől és az elérakott polárszűrőtől függően vagy átengedi a hátsó megvilágítást, vagy nem.

PDP: Minden képpontban egy a ráadott feszültség hatására világító ún. neoncső van. (Ez vagy neonnal, vagy xenonnal van töltve.) Az előzőekhez képest óriási eltérés, hogy itt nem felvillanásról van szó, hanem folyamatos működésről. Ennek során a szín és a fényerő pixelenként állítható. Nézése nem fárasztó, mert a kép rezgés- és villódzásmentes.

4.) Röviden foglalja össze az emberi látással kapcsolatos legfontosabb tudnivalókat!

Az összes érzékelt információ több mint felét (egyések szerint akár 90%-át is!) látás útján szerezzük.

Az ember szemében kb. 126 millió fényérzékelő receptor található. Ezek az elektromágneses sugárzást felfogva a létrejött ingerületet az idegrendszer útján az agyba továbbítják. (A szemben kb. 1 millió idegszál található.)

A retinában elhelyezkedő érzékelők vagy pálcikák (kb. 120 millió), vagy csapok (kb. 6 millió). A pálcikák a fényerősséget és a világosságot érzékelik. Színeket a P típusú (vörös fényre érzékeny), a D típusú (zöld fényre érzékeny) és a T típusú (kék fényre érzékeny) színérzékelő csapok különböző erősségű ingerlése alapján látunk. A fényt csak a 380 nm-es ultraibolya és a 780 nm-es infravörös hullámhossz tartományban vagyunk képesek érzékelni.

A csapok ingerületi állapotának eredője eredményez egy színárnyalatot. Az emberi szem kb. 200 színárnyalat eltérését képes megkülönböztetni. Ez függ a hullámhossztól. A legnagyobb érzékenység az 555 nm-es zöld szín körül mutatkozik.

A fényerősség, ill. világosság érzését a szemünkbe érkező fény energiája határozza meg. Az emberi szem a nagyon kis teljesítményű, 10^{-6} lumen alatti fényt nem érzékeli, a 10^4 lumen feletti teljesítményű fény pedig már elvakít.

A szín érzékelésére a szín telítettsége is hatással van. A színtelítettség a szín fehér színnel való felhígítottságának, fátyolosságának mértéke. Pl. ha a vörös fény telítettségét csökkentjük, fokozatosan rózsaszínt kapunk. Szemünk egy konkrét színezeten belül kb. 20 telítettségi fokozatot tud megkülönböztetni.

Az ember színlátásában tehát a következők játszanak szerepet:

- színárnyalat vagy színezet (**hue**)
- színtelítettség (**saturation**)
- fényerősség (**brightness**)

A szem felbontóképességének határa 0,4 mm körüli. Ahhoz tehát, hogy két pontot meg tudjunk különböztetni, legalább ekkora távolságra kell lenniük egymástól. Ennek a számítógépi grafikában pl. a monitorok és a nyomtatók felbontóképességénél van nagy jelentősége.

Egy kép tudatos érzékeléséhez legalább 1/15 másodpercig kell látnunk a képet. Ennél rövidebb időre felvillanó képet nem érzékelünk tudatosan. A tudatalatti érzékeléshez kevesebb idő is elég. (Lásd betiltott reklámozási eljárások!)

Mozgóképeknél fontos, hogy egy másodperc alatt legalább 25 teljes állóképet jelenítsenek meg. Ennyi kell ugyanis ahhoz, hogy a szem villogásmentesen, folyamatosnak lássa a mozgást.

5.) Ismertesse a leggyakoribb színkeverési eljárásokat!

Az RGB, a CMY, a CMYK és a HSB színterek

Elsődleges fényforrásnak nevezik azokat az objektumokat, melyek maguktól bocsátanak ki fényt. Ilyen pl. a Nap, a gyertya, a lámpa, stb. A másodlagos fényforrások nem képesek önállóan fényt kibocsátani, csak áteresztik azt, vagy visszaverik.

A színkeverésnek két alapvető módja van aszerint, hogy elsődleges vagy másodlagos fényforrásról van-e szó. Az összeadó (additív) színkeverésnél a vörös, zöld, kék alapszínekből vett meghatározott mennyiségek adódnak össze. Így jönnek létre a színárnyalatok. Ezzel az ún. elsődleges fényforrások színeit lehet előállítani. A kivonó (szubsztraktív) színkeverésnél az alapszínek komplementereiből¹ (ciánkék, bíborvörös, sárga) állítják elő a színeket. Így tudják a különböző tárgyak által visszavert fényt modellezni.

A színes felület a fehér fényből csak azt a színt veri vissza, amit érzékelünk, (vagyis egy kéknek látszó tárgy csak a kéket veri vissza), a többi hullámhosszú fényt elnyeli, abszorbeálja.

RGB színtér: A vörös, zöld, kék alapszínekből kikeverhető színeket tartalmazza. Az additív színkeverés modellezéséhez használják.

¹ Két szín komplementere egymásnak, ha összeadó keverésükkor fehéret, kivonó keverésükkor pedig feketét kapunk.

CMY színtér: A ciánkék, bíborvörös, sárga (cyan, magenta, yellow) alapszínekből kikeverhető színeket tartalmazza. A szubsztraktív színkeverés modellezéséhez használják.

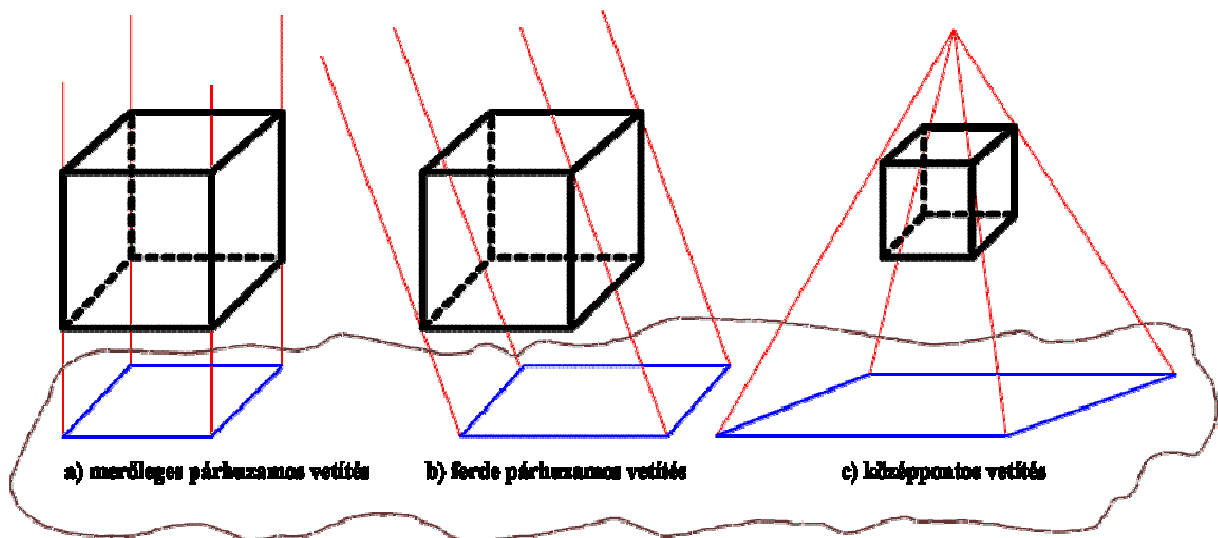
CMYK színtér: Megegyezik a CMY színtérrel, csak még hozzáadódik a fekete szín, mivel a CMY alapszínekből csak szürkét lehet előállítani, a teljesen fekete színt nem. A nyomdatechnikában használják.

Az RGB és a CMYK színterek között nem lehetséges kölcsönösen egyértelmű megfeleltetés. Az RGB alapszínek intenzitását 0 és 255 közötti értékkel, a CMYK alapszíneket pedig 0 és 100 közötti fedettségi értékkel adják meg.

HSB színtér: A színek előállításához az RGB alapszínek mellett a szükséges színtelítettség és világosság értékeket is felhasználják. A HSB színtér jobban alkalmazkodik az emberi érzékeléshez, mint az RGB vagy a CMY.

6.) Mit értünk 3D-2D leképezés alatt?

A leképezés lényege, hogy tér minden egyes (x,y,z) pontjának a sík egy (x',y') pontját feleltetjük meg. (Lásd 11. ábra!)



11. ábra A legfontosabb 3D-2D leképezések

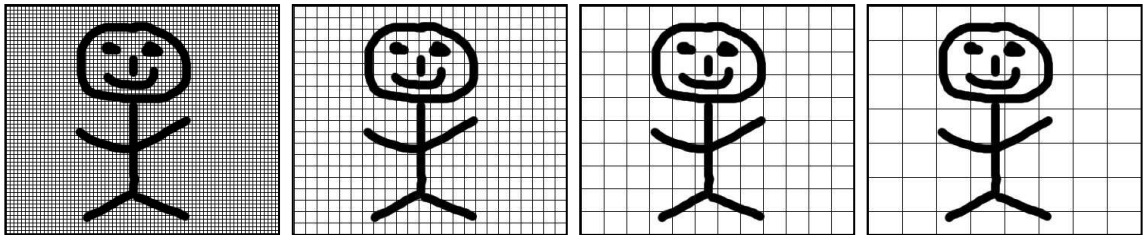
Egy tárgy a fényforrás fényét veri vissza. A visszavert (és ha van, akkor a saját) sugárzás függ az időponttól (t), a megvilágítás szögétől (α), a megfigyelés irányától (γ) és attól, hogy milyen ezen sugárzás hullámhossza (λ). A képképző eljárás ebből az $e(x,y,z,\alpha,\gamma,\lambda)$ folytonos függvényből állítja elő az $f(x,y)$ kétváltozós függvényt.

Ilyen leképező rendszer pl. a fényképezőgép, vagy a kamera is.

Egy fekete-fehér képnél az $f(x,y)$ függvény értéke minden pontban arányos a kép fényességével. Ezt az értéket szokás szürkeségi szintnek nevezni.

7.) Mi a mintavételezés?

A mintavételezés a leképezés utáni lépés. Mintavételezéskor a képsík minden egyes kis területéhez egy számot rendelünk, amely szám az adott tartomány fényességével arányos. Mekkora is legyen egy ilyen kis terület? Nos pontosan ezt mutatja meg a mintavételezés. Segítségül, álljon itt négy ábra!



12. ábra Mintavételezés mérete

Az első képen nagyon picik ezek az említett területek, míg az utolsón elég nagyok. Most minden egyes rácshoz egy számot rendelünk aszerint, hogy a terület közepén milyen színt találunk. Természetesen jobb eredményt kapunk, ha nem a rács közepét nézzük, hanem mondjuk azt, hogy a rácsban melyik szín foglalja el a legnagyobb területet, vagy ha átlagoljuk a négyzetben belüli értékeket. A kapott értékhez digitális képek esetén mindenképpen egy bináris számot rendelünk, vagyis a kép minden rácpontját egy bináris számmal írjuk le.

Nyilván sűrűbb rácsozat esetén lényegesen jobb lesz a kép minősége, mint egy durvább rácsozás esetén. Színes képek esetén a mintavételezés színösszetevőnként végzendő.

Kicsi túlzással a mintavételezés tulajdonképpen a hétköznapi felbontás szónak felel meg.

8.) Mi a kvantálás?

Nézzük meg, hogy az előző feladatban említett hozzárendelt szám milyen értékeket vehet föl! Ezt mutatja meg a kvantálás. Szélsőséges esetben, ha csak kettő színt használunk az ábrázoláshoz, akkor ez a bináris szám vagy 0, vagy 1. (Pl. 0 a fehér, az 1 a fekete színt jelenti.)

Ha már 16 színt tartalmaz a kép, akkor minden képponthoz 4 bitet kell rendelnünk, mert 16 különböző értéket 4 biten lehet ábrázolni.

Fordítsuk meg a dolgot! Ha minden kis négyzetrácshoz 16 bitet rendelünk, akkor decimálisan kifejezve egy képpont 0-tól 65535-ig vehet föl valamilyen egész értéket, vagyis a kép 65536

színt tartalmazhat. 24 bit esetén az ábrázolható színek száma kb. 16,7 millió, míg 32 bitnél ez a szám kb. 4,2 milliárd.

A hétköznapiakban a kvantálás helyett a színmélység szót szoktuk használni.

9.) Mi a helyreállítás lényege, mikor van rá szükség?

Helyreállításra van szükség, ha pl. egy képen valamilyen geometriai transzformációt (nagyítás/kicsinyítés, eltolás, elforgatás, nyírás) hajtunk végre. Minden más olyan esetben is muszáj a helyreállítás, mikor a képernyőn, nyomtatón a megjeleníteni kívánt kép elhelyezkedése, mérete eltér az eredetitől. Ez gyakori művelet, mert a megjelenő kép ablakával végzett bármely művelet is helyreállítást igényel.

A helyreállításkor a digitális képből analóg képet állítunk elő. Ehhez közelítéseket (interpoláció) alkalmazunk. Az eredeti kép viszont több ok miatt sem állítható vissza tökéletesen. Egyik ok a kvantálás. Ez eleve lehetetlenné teszi az eredeti mintavételezett képpont helyreállítását. Az eredeti analóg képfüggvényt tehát csak korlátozott pontossággal lehetséges helyreállítani.

10.) Mi az OpenGL?

Az OpenGL (Open Graphics Library = nyílt grafikus könyvtár) egy grafikus szabvány. Olyan kifejezetten grafikus elemeket magában foglaló eljárások gyűjteménye, mely megkönnyíti a síkbeli és térbeli grafikai programozást. Vektorgrafikus elemeken kívül raszteres és 3D-s animációkat segítő rutinokat is tartalmaz.

Minden operációs rendszertől és grafikus felülettől független. Ha az OpenGL-t egy operációs rendszerben alkalmazni szeretnénk, először meg kell írni a szükséges eljárásokat, melyeket az OpenGL program meghívhat.

Néhány jellemzője:

- geometriai és raszteres primitívek gyűjteménye,
- megvilágítási és áttűnési hatások támogatása,
- textúrák kezelése,
- különböző effektusok (pl. köd, füst) gyűjteménye,
- különböző színterekben való megjelenítések támogatása.

Megalkotásakor ügyeltek a platformfüggetlenségre. Talán ezért, s talán mert tartalmazza a 3D-s mozgások lehetőségét is, napjaink egyik leginkább használatos grafikus környezete lett. (Erről a témáról bővebben lásd [8]!)

II. GIMP - feladatok

1.) Tegyen egy nem átlátszó háttérű képet átlátszó háttérűvé!

A feladat elkészítése előtt tisztában kell lenni azzal, hogy ezt csak úgy lehet megvalósítani, ha a mentéskor GIF vagy PNG formátumba mentjük a kész munkát.

a) Ha GIF lesz az elkészült munkánk kiterjesztése, akkor „csak” egy konkrét színt tudunk lecserélni, azaz átlátszóvá tenni. Ez tökéletes akkor, ha a képünk háttére egyszínű. Ilyen kép a Nyers_Gimp_1a.jpg. A GIF képes tárolni átlátszó képpontokat is, de árnyalni már nem.

A feladat megoldásának lépései:

> Nyissuk meg a képet, majd Réteg → Átlátszóság → Színből alfa...

> A GIMP följánl egy színt, melyet átlátszóvá tesz. Ha ez nem megfelelő, akkor kattintsunk a felajánlott színre. Ekkor megnyílik a „Színből alfa” színpaletta. Itt már kiválaszthatjuk, kikeverhetjük a nekünk megfelelő színt. Arra is mód van, hogy az ún. pipettával (ami a „Mégsem” gomb fölött található) lelopjunk egy színt a képernyő egy pontjáról. (Tudni kell, hogy ilyenkor egy ún. alfa-csatorna adódik hozzá a képhez.)

> Mentsük a munkát GIF formátumba!

b) Ha képünk háttére nem egyszínű (ilyen a Nyers_Gimp_1b.jpg), akkor vagy több lépésben kell az előzőben leírtakat végrehajtani, vagy PNG formátumba mentve a kész munkát, lehetőség van az áttetszőség mértékének megadására (alfa-csatorna).

2.) Készítse el az alábbi ábrán látható képet! A kész kép szélessége 400, magassága 300 pixel legyen! A munkához használja föl a Nyers_Gimp_2.jpg fájlt!

A feladat során a méretezés, forgatás, tükrözés és az áthelyezés műveleteit kell alkalmazni.

> Itt állítsuk be a vászon megfelelő szélességét, magasságát!

> Nyissuk meg a Nyers_Gimp_2.jpg fájlt és

jelöljük ki az egészet! Másoljuk, majd illesszük be az új képbe az előző kijelölést!



13. ábra GIMP-feladatok_2

- > A létrejött lebegő kijelölést alakítsuk réteggé! Jobb egérgombbal klikkeljünk a Rétegek párbeszédablak lebegő kijelölésén és válasszuk ki az Új réteg... lehetőséget! Érdemes valamilyen beszédes nevet is adni a rétegnek!
- > Válasszuk az **Átméretezés** eszközt és ennek segítségével állítsuk be a kívánt méretet! (Szélesség 400, magasság 150 pixel.) A pontos méretek beállításánál szükség lehet a szélességet és magasságot összekötő láncszem bontására.
- > Válasszuk újra a beillesztést! (A vágólapon elvileg az eredeti képnek kell lennie.) Az új lebegő kijelölésből is készítsünk új réteget! Ezt a réteget először érdemes elforgatni. Ha a bal alsó részt akarjuk elkészíteni, akkor -90 fokot kell választani. Az elforgatás után méretezzük át a réteget! (sz: 200, m: 150) Helyezzük a megfelelő pozícióra a réteget!
- > Az utolsó elem megrajzolása úgy a leggyorsabb, ha az előző réteget a másolás és beillesztés után vízszintesen tükrözzük. A lebegő kijelölésből most is készítsünk új réteget! Ezután már csak a helyére kell cipelni a keletkezett réteget.

3.) Egy természeti képre másolja át saját fényképét!

A leglényegesebb dolog a feladat megoldásakor az átmásolandó rész kijelölése. Erre több lehetőség is van a GIMP-ben. Itt az **Olló** eszközt használjuk! (Sok helyen nem véletlenül ezt „Intelligens olló”-nak is hívják.)

Az **Olló** eszköz kontrollpontok útján kijelölt területről a terület kontúrjai segítségével megpróbálja maga eldönteni, mit is gondoltunk kijelölni. Nagyon sok esetben sokkal pontosabb kijelölést tesz lehetővé, mint más eszközök.

A természeti kép a Nyers_Gimp_3a.jpg, a saját fénykép a megoldásban a Nyers_Gimp_3b.jpg fájl. A feladat megoldásának lépései:

- > Az **Olló** eszköz segítségével jelöljük körbe a képről kivágandó részt! A kijelölés záródjon!
- > Miután kész vagyunk a körülrajzolással, kattintsunk a kijelölés belsejébe, ugyanis csak így lesz belőle valóban kijelölés.
- > Másoljuk a kijelölést (pl. CTRL+C) és illesszük be az alapul szolgáló természeti képbe! (Nyers_Gimp_2a.jpg)



14. ábra GIMP-feladatok_3

- > Létrejött egy ún. lebegő kijelölés. Ebből réteget úgy lehet készíteni, hogy jobb egérgombbal klikkelünk a Rétegek párbeszédablak lebegő kijelölésén és kiválasztjuk az Új réteg... lehetőséget.
- > Ezek után a megfelelő eszközökkel méretezzük, forgassuk a keletkezett új réteget és helyezzük a megfelelő pozícióba!

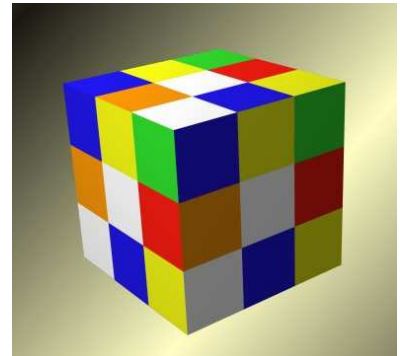
4.) Rajzoljon Rubik-kockát az ábrához hasonlóan!

Ez a feladat jól szemlélteti, hogy mennyi, de mennyi remek dolgot lehet pár kattintással véghezvinni a GIMP-ben.

- > Az új vászon méretét válasszuk úgy, hogy a szélesség és a magasság ugyanakkora legyen és osztható legyen 3-mal! Legyen tehát a kép 390x390 pixeles!

- > Válasszuk ki a Szűrők → Megjelenítés → Minta →

Sakktábla... lehetőséget! A megjelenő párbeszédablakban válasszuk a méretet a vásznunk méretének harmadára! (Tehát most 130 pixel legyen ez a méret!)



15. ábra GIMP-feladatok_4

- > Ha jól csináltuk, akkor most megjelenik egy 3x3-as négyzetrács. A négyzetek az aktuális háttér és előtér színnel rajzolódtak meg. Ezeket kell most a Rubik-kocka színeinek megfelelően 6 különböző színnel kitölteni.

- > Most jön a „kockásítás”. Ehhez válasszuk a Szűrők → Leképezés → Leképezés objektumra... lehetőséget! A párbeszédablak Beállítások lapján a Leképezés erre mezőben válasszuk a téglatest lehetőséget! Az Átlátszó háttér, a Mozaik a forrásképből és az Élsimítás bekapcsolása lehetőségek legyenek kiválasztva! A Mélység legyen 3, a Küszöbszint pedig 0,250 egység!

A GIMP a készülő kockáról egy előnézetet mutat. Érdekes lehet a gyorsabb megjelenítés miatt a Drótvázás előnézet gombot bekapcsolni.

- > Nagyon sok mindent lehet még itt beállítani (pl. fényforrást és helyzetét, fényvisszaverődést, stb.), de mindenképpen érdemes megnézni az Elhelyezkedés lapot. Itt a Helyzet és a Forgatás tulajdonságait valószínűleg állítgatni kell, hogy a nekünk megfelelő eredményt kapjuk. Az „OK” gombra kattintva a GIMP elkezd elkészíteni a Rubik-kocát. Ez a számítógépünk gyorsaságától függően akár több percre is eltarthat.

> Szűrjünk be egy új réteget! Fontos, hogy ez az új réteg az előző, kockát ábrázoló réteg alatt legyen! Ezen lesz ugyanis a háttérünk, melyet színátmenetessé kell tennünk. Ehhez válasszuk ki a **Színátmenet** eszközt! Előtér színének feketét, háttérszínnek pedig halvány sárgát válasszunk! A Színátmenet-nél válasszuk az Előtérből átlátszóba lehetőséget! Az Eltolás mértéke 0, míg a Forma lineáris legyen! Az Ismétlés opciói közül válasszuk a nincs-et és a Színszórás legyen kipipálva!

Egyszínű kocka készítése egy kicsit komplikáltabb. Ehhez két réteg kell. Az elsőn egy fekete négyzetet, a másodikon pedig egy kisebb fehérret kell rajzolni, mindkét esetben 3-mal osztható oldalméretekkkel és a vászon közepére igazítottan. Ezután jöhetnek az előbb felsorolt lépések.

5.) Készítse el az alábbi ábrát!

A feladatban szöveg képre helyezését, ill. lekerekített téglalapok létrehozását lehet gyakorolni.

- > Hozzunk létre egy 400x300-as vásznat!
 - > Válasszuk a Kijelölés → Lekerekített téglalap... lehetőséget! A Sugár (%) értéke legyen 50 és a Konkáv lehetőség NE legyen kiválasztva!
- Töltsük föl a kijelölést kék színnel!



16. ábra GIMP-feladatok_5

- > Hozzunk létre egy új réteget! Ezen a **Téglalap-kijelölési** eszközzel rajzoljunk egy téglalapot, majd ezt is tegyük az előbb említett módon lekerekítetté! (Itt a sugarat válasszuk 65-nek!) Töltsük föl ezt a kijelölést fehér színnel és méretezzük, ill. pozicionáljuk! (Ha lebegő kijelölés keletkezik, akkor jobb klikk rajta és Új réteg... lehetőség választásával egy újabb réteget hozzunk létre belőle!)
- > Most következnek a feliratok elhelyezései. Ezek közül itt csak az egyik elkészítését írom le, mert a többi a beállításoktól eltekintve hasonlóan szerkeszthető meg. Válasszuk ki tehát a **Szöveg** eszközt! Ekkor a képre kattintva beírhatjuk a kívánt szöveget és beállíthatjuk többek között pl. a karakterek típusát, méretét, színét és a betűközt. Ezek után a keletkezett szöveg típusú réteget tetszőlegesen méretezhetjük, pozicionálhatjuk bárhová. Ekkor a szöveget és tulajdonságait újra a **Szöveg** eszközre kattintva tudjuk módosítani. Viszont ha pl. elforgatjuk a szöveget, akkor a réteg megszűnik szöveg-rétegnek lenni, s

ezek után már a szöveg tulajdonságait nem tudjuk módosítani!!! Azt, hogy melyik szöveg melyik másikat takarja, a rétegek sorrendjének változtatásával tudjuk befolyásolni.

> Az árnyékolásokkal lehet még nagyon szép hatásokat elérni. Ehhez ki kell jelölni valamelyik szöveget tartalmazó réteget és a Szűrők → Fény és árnyék → Vetett árnyék... parancsot kell választani. Vigyázzunk, mert az árnyékok új rétegre kerülnek! Piros betűknek kék háttérnél sötétkéék árnyékolást adva nagyszerű térhatást lehet elérni.

6.) Készítsen képet a következő felirattal:

„A Juhász család egy boldog pillanata a sok közül”!

A feladat lényege, hogy szöveget kell egy ún. útvonalra ráilleszteni.

Útvonalat többféleképpen is létre lehet hozni. A legegyszerűbben úgy, hogy az **Útvonal** eszközt választjuk. (Ez tulajdonképpen Bézier-görbék rajzolását jelenti.) Egy másik lehetőség, hogy



17 ábra GIMP-feliratok_6

rajzolunk egy görbét mondjuk az ecsettel, majd kijelöljük azt. Ezután pl. Kijelölés → Útvonallá alakítás. Miután van egy útvonalunk, nézzük a további lépéseket!

> Válasszuk a **Szöveg** eszközt, ahol állítsuk be pl. a szöveg típusát, méretét és írjuk meg a szöveget! Ezután kattintsunk a Szöveg illesztése útvonalra gombra!

> A keletkezett szöveg réteget akár ki is törölhetjük, de mindenképpen tegyük legalább nem láthatóvá a Rétegek ablakban! Ezek után érdemes az Útvonalak lapot láthatóvá tenni. Ezen a lapon most elvileg két réteg található. Egyik az útvonal, a másik maga az útvonal alakját fölvevő szöveg. Legyen ez utóbbi az aktív és kattintsunk az ablak alján található gombok közül a Végigrajzolás az útvonal mentén-re!

> A felbukkanó ablakban adhatjuk meg, hogy a GIMP milyen vastagon rajzolja körül a karaktereket. A körberajzolás színe az aktív előtér színe lesz.

> Miután elkészült a szöveg körberajzolása, jöhet az árnyékolás. Szűrők → Fény és árnyék → Vetített árnyék... Itt az árnyékolás tulajdonságait lehet megadni.

7.) Készítsen egy egyszerű szöveg animációt!

Az animációk készítésének bemelegítőjeként tekinthető ez



a feladat. Az animációról tudni

18. ábra *GIMP-feladatok_7 (combien és replace)*

kell, hogy nem minden képformátum támogatja. A feladatok elkészítése végén én a GIF-be való mentést ajánlom. (A legtöbb szoftver támogatja és az Interneten is az egyik leggyakoribb formátum.) A megoldásban a „HELLO!” szöveg mindenféle cicoma nélküli animálását írom le.

Az animáció egyes képkockáit (frame) külön-külön rétegen kell elkészíteni! Minden egyes frame-hez 3 dolgot kell/lehet megadni a rétegek lapon. Első a frame neve. Ez nem új, minden rétegnek van neve, ha máshogy nem, a program ad egyet. A második jellemzője lehet a rétegnek, hogy mennyi ezredmásodpercig lesz látható, mennyi idő múlva jelenik meg az őt követő képkocka. A harmadik dolog azt fogja megmutatni, hogy miként kerüljenek egymás után lejátszásra a frame-ek. Itt két lehetőség van. Egyik, – ami egyben az alapértelmezett is – hogy az egyes képkockák egymás után úgy jelenítődnek meg, hogy az előző képkockák is láthatóak maradnak. Lehetőség van arra is, hogy az adott képkocka úgy tűnjön elő, hogy az előtte levő már nem marad meg.

Tehát egy képkocka beállítása általában így néz ki: Név (XX ms) (combien/replace). A combien az alapértelmezett, a replace pedig azt jelenti, hogy az adott képkocka megjelenésekor az előző eltűnik. A fenti beállításnál mindig figyeljünk arra, hogy a zárójeleket is ki kell írni!

Az animáció készítésének lépései az utolsó előtti fázisban (a mentés előttiben) térnek el egymástól. Itt kell/lehet/érdemes azt megadni, hogy miként játszódjon le az animáció.

Nézzük tehát a „HELLO!” animálását!

- > Hozzunk létre egy 360x100-as képet! Érdemes a háttérrel átlátszónak választani!
- > Sorban külön-külön hozzuk létre az egyes karaktereket, ill. a nekik megfelelő rétegeket! Az általam elkészített megoldásban a Zombie betűtípust használtam, ahol a méret 100 px. Ekkor a 6 db karakter szépen elhelyezhető egymás mellett.
- > Tulajdonképpen kész is az animáció, mert a többi dolgot már a program fogja megcsinálni, mikor GIF formátumba mentjük a munkát. Ha nem változtatunk semmin és a mentés során ügyelünk arra, hogy a „Mentés animációként” lehetőséget válasszuk (másik lehetőség a „Látható rétegek összefésülése”), akkor a GIMP el is készíti azt az animációt,

mikor a frame-ek úgy jelennek meg egymás után, hogy az előző képkocka is látható marad. Ezt láthatjuk a Kesz_Gimp_7a.gif fájl esetében.

> A másik lehetőség, mikor billentyűzetről bepötyögve külön-külön minden egyes rétegnél beírjuk a (replace) kifejezést. Ennek eredménye a Kesz_Gimp_7b.gif állomány megnyitásával látható.

> Ha az animáció sebességével vagyunk elégedetlenek, akkor ezt sajnos szintén csak az egyes rétegekhez beírt értékekkel tudjuk módosítani.

8.) Készítsen képeslapot Hajdúnánásról!

A feladathoz használjuk föl a Nyers_Gimp_8a.jpg,

Nyers_Gimp_8b.jpg, Nyers_Gimp_8c.jpg,

Nyers_Gimp_8d.jpg és Nyers_Gimp_8e.jpg fájlokat!

Ebben a feladatban tulajdonképpen az előzőeken túl annyi újdonság van csak, hogy a beillesztett képek széleit el kell egy kicsit maszatolni, hogy a képek összeillesztése simábbnak látszódjék. Lehet sokkal szebben is, mint ahogy nekem sikerült.



19. ábra GIMP-feladatok_8

A maszatolás a képek összeillesztésénél a színeket összemosza. Ezt valahogy úgy lehet elképzelni, mintha egy frissen festett képen az ujjunkat elhúznánk a festéken. Hasznos eszköz olyankor is, ha egyszerűbb színösszetételű képen szeretnénk apró részleteket eltüntetni.

> Itt már csak a maszatolás végrehajtását írom le, ami nem lesz túl bonyolult és ebből kifolyólag hosszú sem. Ezt a műveletet a már elmentett JPG kiterjesztésű állományon kell elvégezni, mert itt ugye már csak egy réteg van. A **Maszatolási** eszköz kiválasztása után az összeillesztett képek széleinél kell ezt a műveletet alkalmazni. Beállíthatjuk az eszköz lapján többek között az eszközünk alakját, a nyomásérzékenységet és a maszatolás sűrűségét is. Kísérletezni kell!

9.) Rétegmaszk alkalmazásával készítse el a Nyers_Gimp_9.jpg állományból az ábrán látható képet!

A feladat nem túl nehéz, ha tisztában vagyunk az alfa-csatorna jelentésével.

A rétegek rendelkezhetnek alfa-csatornával. Ha van alfa-csatorna, akkor minden pixel a színén kívül még egy



20. ábra GIMP-feladatok_9

plusz információval is rendelkezik. Ez az információ azt adja meg, hogy mekkora a pont átlátszatlansága, azaz alfa-értéke. Ha ott rajzolunk, ahol átlátszó réteg van, a festett pontokban nő, ha törölünk, radírozunk, akkor csökken az alfa-érték.

Egy rétegmaszkkal mód nyílik arra, hogy ezt az átlátszóságot szerkeszteni lehessen.

A feladat kivitelezése:

- > Nyissunk egy 400x300-as vásznat, majd töltsük föl valamilyen színátmenettel!
- > Ezután a Fájl → Megnyitás réteggént... parancsot választva nyissuk meg a Nyers_Gimp_9a.jpg állományt! Ezt át kell méretezni a vászon méretére: Réteg → Réteg átméretezése...
- > Most készítsünk egy új réteget, melynek neve legyen „háttér2”! Valamilyen módszerrel töltsük ki ezt is! Ahhoz, hogy a fenti mintát megkapjuk, a **Színátmenet** eszközből a Mexican flag-et kell kiválasztani és a rétegen az egeret a jobb alsó sarokból indulva kell a bal felsőig húzni.
- > A „háttér2” nevű réteghez kell most rétegmaszkot hozzáadni. Jobb klikk a Réteg lapon a réteg nevén és Rétegmaszk hozzáadása... Itt válasszuk a Fehér (teljesen átlátszatlan) lehetőséget!
- > Tegyük aktívvá a képet tartalmazó réteget! Másoljuk az egészet a CTRL+C billentyűkombinációval!
- > Újra válasszuk ki a „háttér2” nevű réteget! Jobb klikk ezen a rétegen és Rétegmaszk szerkesztése. Ekkor megjelenik egy fehér téglalap a réteg neve mellett a Réteg lapon. Kattintsunk ezen téglalapra és illesszük be a vágólap tartalmát! (CTRL+V)
- > A keletkezett Lebegő kijelölés-t a Réteg lap legalsó sorában található utolsó előtti (lebegő réteg rögzítése) gombra való kattintással tudjuk a „háttér2” rétegmaszkjába illeszteni.

> Tulajdonképpen kész is vagyunk. A képet tartalmazó réteget ki is lehet törölni. Finomítani lehet még a Színek → Fényerő-kontraszt... pont alatt található beállításokkal. Egy próbát a Színek → Invertálás is megér. Megemlítem még, hogy a rétegek továbbra is szerkeszthetők, tehát lehet nekik új kitöltéseket adni, ezzel is megpróbálni minél jobb hatást elérni.

10.) Készítse el az ábrán látható képet!

A képen egy tavaszi és egy őszi erdő részletei láthatók. A két képet egybemostuk, melyre szintén a rétegmaszkot lehet fölhasználni. És ezt hogyan? A lépések:

> Nyissuk meg a Nyers_Gimp_10a.jpg fájlt és méretezzük át 800x600-asra!

> Nyissuk meg a Nyers_Gimp_10b.jpg fájlt is és ezt is méretezzük át 800x600-asra!

> Ez utóbbi képet teljes egészében jelöljük ki, majd rakjuk a vágólapra! Ezek után akár be is zárhatjuk.

> Illesszük be a vágólap tartalmát a megnyitott Nyers_Gimp_10a.jpg fájlba! A keletkező lebegő kijelölésből készítsünk új réteget! A réteg neve legyen „ősz”!

> Ehhez a réteghez adjunk hozzá rétegmaszkot!

> Most jön a kitöltés színátmenettel. Ennek beállításai: Mód: Normál; Átlátszatl.: 100,0; Eltolás: 0,0; Forma: Sugaras; Ismétlés: Nincs. A Színszórás és az Adaptív túlmintavételezés ne legyen kiválasztva!



21. ábra GIMP-feladatok_10

III. Flash - feladatok

Ahhoz, hogy a Flash-ben készült munkák képét dokumentumba be tudjuk szúrni, pl. JPG kiterjesztéssel kell elmenteni az FLA kiterjesztésű késznek ítélt Flash állományt. Ez a hagyományos File → Save as... módon nem megy. Az exportálást kell ugyanis választani: File → Export → Export Image...

Az 1. feladatnál még részletesen ecsetelem a lehetséges megoldások egyikét, de aztán már csak a leglényegesebb lépéseket fogom leírni. Feltételezem ugyanis, hogy némi Flash előismerettel rendelkezik az, aki a feladatok megoldására vállalkozik.

Az animációknál a dolgozatomban csak az első és az utolsó kockát illesztettem be.

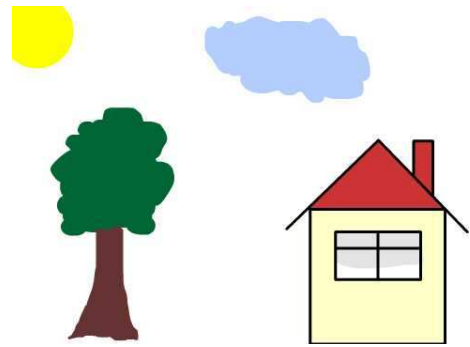
Ha a Flash-ben úgy rajzolunk valamit, hogy az lelóg a vásznonról (rajzlapról), akkor az később a Flash-ben nem jelenik meg. Viszont ha pl. JPG formátumba exportálunk, akkor a keletkező kép mérete úgy változik meg, hogy abba minden „beleférjen”!!!

A Flash az animációk készítésében (is) jeleskedik. Nagyon vázlatosan most összefoglalom a Flash-animációk lényegét. A Flash elméleti háttérét érintő kérdéseket bővebben lásd [6] , [7]! Egy animáció képkockákból áll össze, melyek a Timeline-on (idősáv) kapnak helyet. Képkockából háromféle létezik:

- **Kulcs képkocka:** Ez jelöli az animáció azon pontját, ahol valamilyen változás történik. Ilyen lehet például a mozgás, a méret vagy a színváltozás. Fekete kör a jele.
- **Képkocka:** Semmilyen lényeges dolgot, semmilyen vizuális információt nem tartalmaz. Jele egy üres négyzet. Azt befolyásolhatjuk vele, hogy egy kulcs képkocka mennyi ideig látszódjon.
- **Tweening-et tartalmazó képkocka:** Ezt a program számolja ki a kiinduló és befejező képkockákból. Két fajtája van. Az első a shape tween (formanyújtás), melyet akkor alkalmazunk, ha az animáció során drasztikus alakváltás, színváltás következik be, vagyis ha az objektum paraméterei, egyenlete megváltozik. A másik a motion tween (mozgásnyújtás) használata köztes mozgások megadásakor ajánlatos. Ezt csak szimbólumoknál alkalmazhatjuk, azaz először szimbólummá kell alakítani, amit mozgatni óhajtunk. Világos zöld a képkockák háttere, ha a shape tween-t és világoskék, ha a motion tween-t használjuk. Ha mindent jól csinálunk ezen átmenetek alkalmazásakor, akkor a kezdeti és a végállapotot tartalmazó kulcs képkockákat folyamatos nyíl köti össze.

1.) Rajzoljon házikót, fát, Napot, felhőt!

Ez a feladat a legalapvetőbb dolgokat próbálja gyakoroltatni. Van benne szakasz, kör, téglalap rajzolása, kitöltés, tükrözés, stb. Nézzünk egy lehetséges megoldási menetet!



22. ábra *Flash-feladatok_1*

- > Először rajzoljuk meg a házikót! Rajzolunk egy nagyobb téglalapot (Rectangle Tool), majd ebben egy kisebbet (az ablak kerete). Vonal vastagságát állítsuk kb. 3 egységre! Ezután szakasz rajzolással (Line Tool) készítsük el felosztásait! Ha valami nem oda kerül, ahová szeretnénk, jelöljük ki az objektumot és egyszerűen vonszoljuk arrébb az egérrel!
- > A ház tetejének megrajzolása: Készítsük el a Line Tool-lal először mondjuk a bal oldali tetőt! Jelöljük ki ezt a szakaszt, majd másoljuk és illesszük be a másolatot. Ezt kell most tükrözni a Modify → Transform → Flip Vertical-lal. Ezt a szakaszt cipeljük az előzőhöz, majd jelöljük ki mindkettőt. Így már vonszolható is a tető a helyére.
- > Másodjára rajzoljuk meg a fa törzsét! Válasszuk pl. a Pencil Tool eszközt és állítsuk a színt barnára! A színek kiválasztásakor lehet próbálkozni az ún. Alpha (=alfa-csatorna) megadásával. Az Options-nál az Object Drawing ne legyen kiválasztva! Rajzoljuk meg tehát a fa törzsét úgy, hogy zárt alakzatot kapjunk, vagyis ehhez az alját és a tetejét is meg kell rajzolni. A Paint Bucket Tool kiválasztása után kattintsunk bele a megrajzolt fatörzsbe!
- > Következik a fa koronája. Ehhez a Brush Tool eszköz kell, melynek beállításainál a Paint Normal-t válasszuk! Természetesen a színt valamilyen zöldre kell állítani.
- > A felhő megrajzolása megegyezik a fa koronájának rajzolásával. (Kivéve a színt.)
- > A Napot az Oval Tool-lal lehet megrajzolni. Ez ki fog lógni a vászonról, ami a Flash-ben nem is lenne gond, mert ott ez a kilógó rész nem látszódná. De ha más formátumba exportáljuk a rajzot, akkor ezt a kilógó részt le kell vágnunk. Ehhez nagyítsuk föl az ábrát és rajzoljunk függőleges, ill. vízszintes vonalakat a vászon bal felső határaihoz. Függőleget/vízszintest úgy tudunk rajzolni, ha rajzolás közben nyomva tartjuk a SHIFT billentyűt. Ezután a megfelelő részeket a Selection Tool-lal kijelölve a DELETE billentyűvel tudjuk kitörölni.
- > Hátra van még a ház kiszínezése. A tető, a kémény és a házfalak kiszínezése nagyon egyszerű, mert csak a Paint Bucket Tool-t kell a megfelelő színnel kiválasztani és

kattintani a kívánt részbe. A függöny megrajzolásához a következőt kell tenni. Ki kell választani a #CCCCCC kódú színt és annak átlátszóságát (alfáját) kb. 70%-ra kell állítani. A Brush Tool eszközzel lehet megrajzolni a függönyt, de hogy ez a rajzolás az ablak keretei mögé kerüljön, be kell állítani a Brush Mode-nál a Paint Fills lehetőséget.

2.) Rajzoljon egy pislogó fejet!

Ez a feladat a Flash-animációk készítésének bevezetője. Nagyon egyszerű, de a legalapvetőbb animálási eljárásokat egy ilyen példával lehet bemutatni és magát az animációk létrehozását valami ilyesmivel érdemes kezdeni.

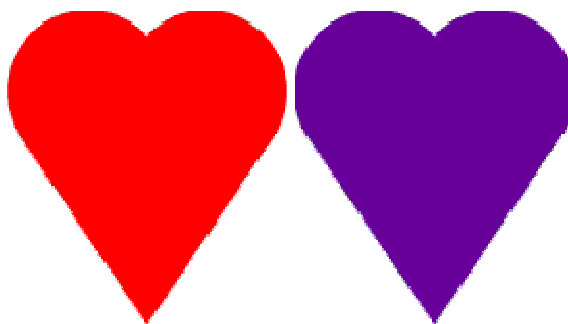


23. ábra *Flash-feladatok_2*

- > Rajzoljuk a rajzlap közepére az első fejet!
- > A Timeline 10-es képkockáját alakítsuk Key Frame-mé, azaz az animáció egy kulcs képkockájává! (Pl. F6 billentyű.)
- > A 15. képkockát is tegyük Keyframe-mé és módosítsuk rajta a rajzot úgy, ahogy fentebb látjuk a 2. ábrán! Tulajdonképpen kész is az animáció. Lejátszani az ENTER megnyomásával lehet.

3.) Rajzoljon egy tetszőleges alakzatot, mely folyamatosan változtatja a színét!

Ezen feladat megoldásában szívek változtatják a színüket, mégpedig a szivárvány színeinek megfelelően. Először vörös, narancs, sárga, zöld, kék, ibolya színsorrendben, majd visszafelé.



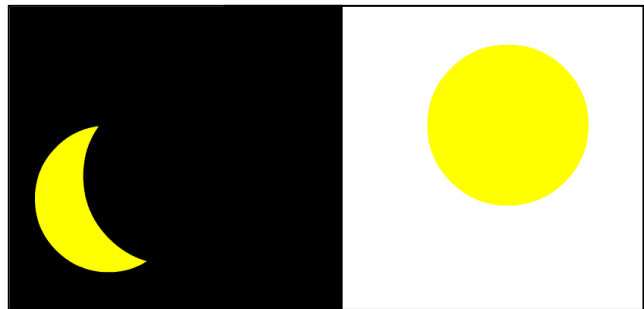
24. ábra *Flash-feladatok_3*

- > Rajzoljuk meg a vörös színű szívet és helyezzük a vászon (rajzlap, stage, színpad) közepére!
- > Kattintsunk a Timeline-on az első képkockára és Properties-nél a Tween mező értékét állítsuk Shape-re!

- > Ezután az 5. képkockát tegyük kulcs képkockává (Keyframe – F6)! Ha minden jól ment, akkor most a Timeline-on az 1. és az 5. képkockát egy halvány zöld háttérű fekete nyíl köti össze.
- > Színezzük át ezen képkockán (tehát az 5.-en) a szívet narancssárgára!
- > Ez utóbbi két lépést kell ismételgetni, míg az utolsó, ibolyaszínű szívet is meg nem rajzoltuk. Ekkor az animáció már lejátszható, de még csak a vöröstől jutunk el az ibolyáig.
- > Visszafelé természetesen nem kell újra megrajzolni a kulcs képkockákat, van egyszerűbb mód is. Jelöljük ki az eddig készített képkockákat! (Ha jól csináltuk, akkor 0-tól 25-ig.) Jobb klikk az utolsó képkockán és a helyi menüből válasszuk ki a Copy Frames parancsot! A 26. képkockára állva újra jobb klikk és Paste Frames. Ezután már csak meg kell fordítani a most beillesztett képkockák sorrendjét. Ehhez jelöljük ki (ha még nem lenne) 26-tól 50-ig az összes képkockát! Jobb klikk ismét és Reverse Frames. Ezzel kész is az animáció.

4.) Készítsen animációt, melyben a Hold Nappá változik!

Az előző feladatban az alakzat színe változott, itt pedig most az alakja és a háttér színe fog változni. Tehát itt is a shape tween-t, valamint újdonságként rétegeket kell alkalmazni.



25. ábra *Flash-feladatok_4*

- > Hozzunk létre egy új Flash dokumentumot, melynek a rajzlapját valamilyen módszerrel fessük feketére! (Pl. rajzolunk egy fekete téglalapot.)
- > Hozzunk létre egy új réteget, melyre megrajzoljuk a Holdat. Ezt lehet úgy, hogy a sárga körből kivonunk egy másik színű kört, majd azt töröljük.
- > Most egy fontos lépés következik. Mindkét rétegen a Tween tulajdonságot állítsuk Shape-re!
- > Mindkét rétegnél pl. a 25. képkockát tegyük Keyframe-mé!
- > A fekete háttérű rétegen most töröljük ki a fekete téglalapot (feltéve, hogy az 1. képkockán ilyen módszerrel hoztuk létre a fekete háttér), a Holdat tartalmazó rétegről pedig töröljük a Holdat és rajzoljuk meg a Napot!

> Ha mindent jól csináltunk, akkor mindkét rétegnél az első és az utolsó képkockát egy világoszöld háttérű folyamatos nyíl köti össze. A kész animáció most már lejátszható.

5.) Mozgasson egy tetszőleges alakzatot egy előre megrajzolt pályán!



26. ábra *Flash-feladatok_5*

A megoldásnál Nyers_Flash_5.gif állományból indultam ki. (Egy legyet ábrázol a kép.)

Ahhoz, hogy irányított mozgást lehessen létrehozni, egy speciális rétegre van szükség. A Flash-ben rétegekből több típus is van.

- **Normal:** Ez az alapértelmezett, semmi különlegességgel nem rendelkezik. Ilyet használtunk az eddigi feladatokban is.
- **Guide layer:** Ez az ún. irányító réteg, ami azt jelenti, hogy az alatta elhelyezkedő, ún. vezérelt rétegen lévő objektum az itt megrajzolt pálya szerint fog mozogni.
- **Guided layer:** Ez a vezérelt réteg. Ennek tartalma fog a fölötte elhelyezkedő, vezérelt rétegen megrajzolt pálya alapján mozogni.
- **Mask:** Meghatározza, hogy mi látszik az alatta lévő, ún. maszkolt (masked) rétegből.
- **Masked:** Ez a maszkolt réteg. Csak az a része látszódik, amit a maszk réteg kitakar.
- **Folder:** Ha sok réteggel dolgozunk, könnyebb közöttük eligazodni, ha mappákba szervezzük őket, hasonlóan az operációs rendszereknek a megszokottakhoz.

Egy réteg tulajdonságait a Modify → Layer menüpontban állíthatjuk be. Ugyanitt szabályozhatjuk a réteg láthatóságát, lezártságát, vagy pl. azt, hogy a körvonalak milyen színnel jelenjenek meg. Ezek után nézzük a feladat megoldását!

- > Egy új Flash dokumentumba a File → Import → Import to Stage... (vagy CTRL+R) segítségével illesszük be a Nyers_Flash_5.gif állomány tartalmát (vagyis a legyet)! Ezt a rajzot méretezzük és helyezzük el valahol a stage jobb oldalán. Innen fog indulni a légy.
- > Kattintsunk a Timeline 1. képkockáján és állítsuk a Tween tulajdonságot Motionra!
- > Az 50. képkockát tegyük Keyframe-mé (F6) és vigyük a legyet az animáció végpontjába!
- > Tegyük ismét aktívvá az 1. képkockát és klikkeljünk a Layer ablak Add Motion Guide gombjára! (Egy kis + jelet keressünk!) Ekkor létre jön a legyet tartalmazó réteg fölött egy

guide layer (irányító, vezérlő réteg). Itt pl. a ceruzával megrajzolhatjuk a mozgás pályáját. Miután ez meg van, a réteg 50. képkockáját kulcs képkockává kell tennünk.

> Már csak az van hátra, hogy az első és az utolsó képkockán a legyet a pályához ragasszuk. Ez nem mindig egyszerű, kísérletezni kell. A kijelölt rajzobjektum közepén található egy kis karika, ezt kell a megrajzolt görbe elejére, ill. végére illeszteni. Ha a réteg Properties ablakában a Snap checkboxot bejelöljük (Snap to guide), akkor az alakzat automatikusan a pályához kapcsolódik.

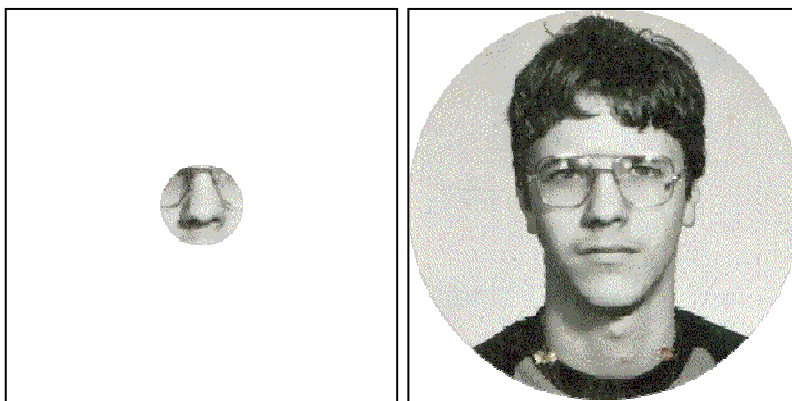
> Arra is lehetőségünk van, hogy a mozgást gyorsulóvá, avagy lassulóvá tegyük. Ehhez a Properties lapon kell az Ease mezőnek értéket adni. Ha ez az érték -1 és -100 közötti, akkor az animáció végéhez közeledve gyorsul a mozgás. Amennyiben még ettől is többet szeretnénk, válasszuk az Ease mező melletti Edit... gombot! Itt tovább szerkesztgethetjük a mozgás sebességét. További lehetőség még az Orient to Path kipipálása, mert ilyenkor az alakzat a mozgása során mindig az útvonal felé „néz”.

> Amennyiben kész az animáció, már csak a vezérlő réteget kell láthatatlanná tenni.

6.) Készítsen animációt, melyben egy képnek fokozatosan egyre nagyobb része válik láthatóvá!

Ehhez a feladathoz az előző feladat bevezetőjében már említett réteg maszkra lesz szükség.

Ha egy képből csak egy részletet szeretnénk megmutatni, akkor van szükségünk



eme ún. mask layer-re. Ezen

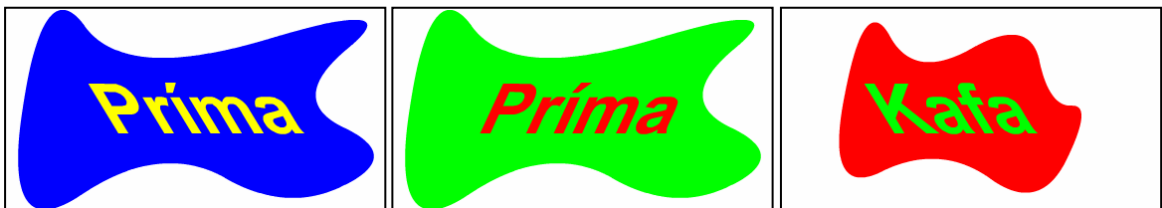
27. ábra *Flash-feladatok_6*

kell megrajzolni a maszkot, ami azt jelenti, hogy az alatta lévő ún. maszkolt rétegből csak ez a megrajzolt alakzat általi terület fog megjelenítődni. Jöjjön a megoldás!

> Importáljuk be egy új Flash dokumentumba a Nyers_Flash_6.jpg fényképet! Ha kell, a fényképet igazítsuk a vászon közepére! Ezt a réteget nevezzük el „fénykép”-nek, ez lesz a maszkolt réteg. Az 50. képkockát tegyük kulcs képkockává!

- > Szúrjunk be egy új réteget, melynek első képkockájának Tween tulajdonságát állítsuk Shape-re! Rajzoljunk meg itt egy akármilyen színű kört, majd igazítsuk ezt a színpad centrumába!
- > Válasszuk ezen réteg 50. képkockáját, melyet tegyünk keyframe-mé és nagyítsuk meg itt a kört!
- > Elvileg már csak az van hátra, hogy ebből a rétegből maszk réteget készítsünk. Jobb klikk a réteg nevén, s a helyi menüből a Mask lehetőséget kell kiválasztani.
- > Még egy dolgot érdemes beállítani, ez pedig a stage háttérszíne. Ehhez klikkeljünk az egér jobb gombjával a vászon egy üres területén és a helyi menüből válasszuk a Document Properties... lehetőséget! A megjelenő ablakban a Background color-nál lehet a háttér színét variálni. Érdemes egy halvány szürkét kiválasztani.

7.) Rajzoljon egy gombot, mely megváltoztatja színét/alakját/feliratát, ha az egeret rámozgatják!



28. ábra *Flash-felatok_7*

Az első ábrán látható az alap gomb. Középen az, mikor az egeret a kép fölé visszük. Az utolsó ábra azt mutatja, hogy hogyan néz ki a gomb, mikor kattintunk rajta.

A Flash-ben olyan animációkat is lehet készíteni, melyek interaktívak. Tipikusan ilyen eset az, amikor a felhasználó a gomb felé viszi az egeret, vagy kattint rajta, akkor a gomb színe/alakja/felirata megváltozik. Erre mutat példát ez a feladat.

A megoldás előtt egy kis elmélet. Eseményeket rendelhetünk kulcs-képkockához, gombhoz, illetve mozikliphez. A gombnak négyféle keyframe-je lehet, amelyek meghatározzák a gomb külsejét. Ezek

- **Up:** Ez egy gomb alapállapota, mely akkor látható, ha az egér nincs a gomb fölött.
- **Over:** Ez határozza meg a küllemet akkor, ha a kurzor a gomb fölött van.
- **Down:** A rákattintáskori kinézetért felelős.

- **Hit:** Ez azt a terület határozza meg, melybe ha belemozgatjuk az egeret, a gombnak reagálni kell. Tehát nem biztos, hogy csak akkor történik változás egy gomb megjelenésében, ha pontosan a gomb fölé kerül ez egér. Lehet ettől kisebb, ill. nagyobb területet is megadni. Eddig csak arról írtam, hogy a külalakja hogyan változhat meg a gombnak. Ez igaz is, de természetesen nem csak a külalak változhat, hanem egy kattintás események sorozatát indíthatja el. Ebben a feladatban csak a külalakkal foglalkozunk.

A feladat megvalósítása:

- > Rajzoljuk meg és feliratozzuk a gomb alapállapotát! Ez persze még nem gomb. Átalakítani gombbá úgy lehet, ha kijelöljük az egész rajzot, majd választjuk a Modify → Convert to Symbol... parancsot, vagy egyszerűen lenyomjuk az F8-at. A megjelenő ablakban meg kell adni a gomb nevét (Name), típusát (Type) és azt, hogy hol legyen a gomb abszolút középpontja (Registration). Típusnak mindenképpen a Button-t válasszuk!
- > A létrehozott gombot meg kell nyitnunk, ha szerkeszteni szeretnénk. Ezt egyszerűen megtehetjük, ha duplán kattintunk az ábránkon, vagy ha a Timeline jobb felső sarkában található Edit Symbol gombot választjuk.
- > A Timeline-on most 4 frame-nek kell megjelennie. (Up, Over, Down, Hit) Alakítsuk mindegyiket kulcs képkockává!
- > Az Over frame-en szerkesszük át a gombot olyanra, amilyennek látni szeretnénk, ha az egeret fölé mozgatjuk!
- > A Down frame-en szerkesszük át a gombot olyanra, amilyennek látni szeretnénk, ha az egérrel rákattintunk!
- > Most vissza kell menni a főjelenetbe, melyet a Timeline bal felső részén a Scene 1 feliratú gombra való kattintással tehetünk meg. A CTRL + ENTER-rel már meg is lehet nézni, hogy miként viselkedik a gomb.

8.) Készítsen egy mozgó reklámcsíkot (bannert)!



29. ábra Flash-felatok_8a



30. ábra Flash-felatok_8b

A fenti két ábra csak egy-egy részlet a mozgó reklámból, időrendi sorrendben. A természeti kép fölfelé, majd lefelé úszik a képen, míg a szöveg balról megy jobbra úgy, hogy közben a pirossal írt szöveg is meg-megjelenik.

A feladat megoldásánál a Nyers_Flash_8.jpg állományból induljunk ki!

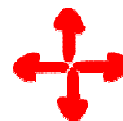
- > Először méretezzük át a színpadot! Ehhez klikkeljünk a jobb egérgombbal a stage-en, majd a helyi menüből válasszuk a Document Properties... lehetőséget! Itt a Dimensions mezőkben lehet beállítani a szélességet és a magasságot. Legyen ez most 1024x100!
- > Importáljuk be a Nyers_Flash_8.jpg fájlt, majd igazítsuk a képet a stage bal felső sarkához! Azért kellett a vászon szélességét 1024-re állítani, mert ekkora a beimportált kép szélessége is.
- > A keletkezett réteget nevezzük el „kép”-nek és az 1. képkockájának Tween tulajdonságát állítsuk Motion-ra!
- > A „kép” réteg 150. képkockáját tegyük kulcs képkockává! Most igazítsuk a képet a stage bal alsó sarkához! Ezután a 300. képkockának kell ugyanolyannak lenni, mint a legelsőnek. Ezzel elértük azt, hogy a háttérként szolgáló kép le-föl fog mozogni a reklámcsíkbán.
- > Következik a balról jobbra mozgó szöveg elkészítése. Ehhez készítsünk egy új „szöveg” nevű réteget! Válasszuk ki a Text Tool-t és szerkesszük meg a szöveget! A szín legyen fehér, melynek alfáját állítsuk 50%-ra. A kész szöveget vigyük le a színpadról a bal oldalra! Erre azért van szükség, mert a szöveg az animáció indulásakor még nem látszik, hanem balról fog majd beúszni.
- > A „szöveg” réteg 1. képkockájának is állítsuk be a Tween tulajdonságát, méghozzá szintén Motion-ra! A 300. képkocka is legyen keyframe! A szöveget vonszoljuk úgy, hogy a színpad jobb oldalán legyen és ne lógjon már be a rajzlapra!

> Elvileg most kész a banner, csak még a piros szöveg (a cím) hiányzik. Ehhez újabb réteget („szöveg2”) hozzunk létre! A 110. frame-et tegyük kulcs képkockává és állítsuk be a Motion tulajdonságát! Készítsük el a piros szöveget, melyet igazítsunk a vízszintesen a színpad közepéhez, függőlegesen pedig a tetejéhez! A 150. képkockát is tegyük keyframe-mé! Mást nem kell tennünk itt a szöveggel, ugyanis az nem fog mozogni. A mozgás látszatát a háttérül szolgáló kép mozgása fogja majd kelteni.

> Utolsó lépésként az előző pontban leírtakat ismételjük meg az animáció végénél, vagyis a 260., ill. a 300. képkockán. Így az animáció során a piros szöveg kétszer fog feltűnni. Mentsük a munkát! Ezt megtehetjük úgy is, ha a File → Publish Preview → HTML-t választjuk. Ekkor egyből egy böngészőben nézhetjük meg az eredményt.

> A feladat (remélhetőleg csak egyik) hibája, hogy a „g” és „j” betűk aljai kilógnak a reklámcsíkból.

9.) Rajzoljon egy tetszőleges alakzatot, majd a fő égtájak felé mutató nyilakat! Érje el, hogy az alakzat a nyilakra kattintva a megfelelő irányba mozogjon!



Ebben a feladatban már bepillantást kapunk a Flash programozásába. Nem bocsátkoznék nagyon bele az elméletbe, hiszen az rengeteg oldallal szaporítaná a dolgozatomat. A lényeg, hogy a Flash programozására az ActionScript használható, mely elég sok dologban hasonlít a JavaScriptre. Ez is a ma oly divatos objektumorientált programozási nyelvek közé sorolható.

31. ábra *Flash-feladatok_9*

Az ActionScript nagyon sokféle eseménykezelővel van felruházva. ActionScriptet rendelhetünk képkockákhoz, gombokhoz és moziklipekhez.

Programunkat az Actions Panel-en írhatjuk meg – az előző mondat alapján – kulcs képkockára, gombra, vagy moziklipre.

Bevezetőnek legyen ennyi most elég! A további fontos információkat a feladatmegoldás lépéseinél írom le.

A megoldásban használjuk föl a Nyers_Flash_9.gif állományt! (De akár ténylegesen rajzolhatunk is egy saját alakzatot.)

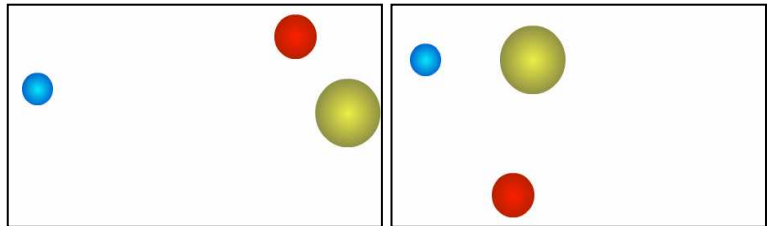
- > Importáljuk a stage-re a `Nyers_Flash_9.gif`-et! Ez lesz az az objektum, amit mozgatni fogunk. Hogy ezt tudjuk programozni, készítsünk belőle moziklipet! Ehhez jelöljük ki, majd nyomjuk le az F8-at! A felkínálkozó lehetőségek közül válasszuk a `Movie clip` típust!
- > A `Properties` lapon (az elvileg a színpad alatt található) adjunk az `Instance Name` mezőben a moziklip típusú objektumunk ezen példányának nevet! Legyen a név „kisfiu”!
- > Most a nyílak megrajzolása és gomb típusú szimbólummá tétele következik. Elég csak az egyik gombot megrajzolni, mert a többit abból elforgatásokkal megkapjuk.
- > Tegyük tehát mind a négy nyílat a vászonra! Válasszuk ki mondjuk a keletre mutatót! Az `Actions` panelen keressük meg a `Statements` → `Variables` → `set variable` parancsot! Ha minden jól ment, akkor a mellette lévő fehér téglalapba írjuk be a következő utasítást:

<pre>on (release) { _root.kisfiu._x= _root.kisfiu._x+10; }</pre>	<p>Ez a kód azt jelenti, hogy a „kisfiu” objektum x-koordinátáját növelni kell 10-zel, amennyiben a gombon kattintanak. Pontosabban kattintanak, majd felengedik a gombot. (release = kiold)</p>
--	--

- > Ha a kód első sorába kattintunk és lenyomjuk a panel jobb felső sarkában lévő `Script Assist` gombot, akkor további eseményeket rendelhetünk a gombhoz. Például, ha kiválasztjuk a `Key Press` lehetőséget és megnyomjuk a fölfelé mutató nyíl billentyűt, akkor módosul a kód is és innentől már majd a nyíl billentyűvel is lehet mozgatni az alakzatot.
- > Ha a kódban a 2. vagy 3. sorra kattintunk, akkor fölötte két mező jelenik meg. A `Variable` mező `Expression` tulajdonságát NE válasszuk ki, viszont az alatta elhelyezkedő `Value` mező ugyanilyen nevű tulajdonsága legyen kipipálva! Ez utóbbi választás azt jelenti, hogy nem egy szöveget, hanem egy kifejezést szeretnénk értékül adni.
- > Hasonló dolgokat kell végrehajtani a másik három nyíl objektumon is. Megjegyzésként még annyit, hogy a lefelé mozgáskor az y-koordináta növekszik.
- > A `CTRL + ENTER`-rel lejátszható a kész moziklip.

10.) Készítsen olyan Flash-animációt, melyben 3 golyó mozog egyenletesen úgy, hogy az ablak széleinél mindegyik visszapattan!

Ez a feladat már komolyabb Flash ismereteket feltételez. Ezen belül itt már nem a rajzoláson van a hangsúly,



32. ábra *Flash-feladatok_10*

hanem az ActionScript használatán, azaz a Flash programozásán.

Az animációk készítése a Timeline segítségével semmilyen programozási ismeretet nem igényel. A Flash a kulcs képkockák közötti dolgokat szépen kiszámolja és ennek megfelelően jeleníti meg a képkockákat. Az ilyen mozgások nagy hátránya, hogy futási időben nem változtathatók. Ezzel szemben az ActionScript roppant sok lehetőséget biztosít objektumok dinamikus, futásidőben történő mozgatására.

A feladat megoldásának leírása előtt következzen újra egy kis elmélet!

Kérdés, hogy miként lehet az ActionScripttel a moziklipet manipulálni. A moziklipeknek vannak tulajdonságaik (Properties) és metódusaik (Methods). Ezek határozzák meg, hogy mit lehet egy objektummal tenni, illetve hogy ő maga mit tehet. Tulajdonságok például:

_x, _y: A moziklip x-, ill. y-koordinátája.

_xscale, _yscale: A moziklip százalékban megadott szélessége, ill. magassága.

_alpha: A moziklip átlátszósága százalékban.

Nemcsak a gomboknak (lásd előző feladat!), hanem a Flash minden programozható objektumának vannak eseményei. Ezek meghatározott sorrendben hajtódnak végre.

Minden moziklip saját eseménykezelőkkel rendelkezhet. Íme néhány:

- **Load/Unload:** A moziklip betöltése, ill. RAM-ból való törlése váltja ki az eseményt.
- **EnterFrame:** Akkor hajtódik végre, ha a moziklip minden képkockája betöltődött.
- **Mouse down/up:** Az egér lenyomása, ill. felengedése váltja ki ezt az eseményt.
- **Mouse move:** Az egér mozgásakor következik be.
- **Key down/up:** Egy billentyű lenyomásakor következik be.
- **Data:** Akkor jön létre, ha egy moziklip információt küld az aktuális moziklipnek.

Nos, ha mozgatni szeretnénk egy alakzatot, akkor annak az x-, esetleg y-koordinátáját kell megváltoztatni. Például a kódban elhelyezett valami._x = valami._x+5; utasítás az objektumot 5 egységgel jobbra teszi. (Ugyanez az eredmény, ha valami._x += 5 formában írjuk az előzőt.) Ekkor viszont még nem lesz folyamatos a mozgás. Ehhez kell az `enterFrame` eseményt használni. Az adott utasítás az esemény bekövetkeztekor annyiszor fut le másodpercenként, amennyi a dokumentum tulajdonságainál beállított Frame rate érték fps-ban megadva. A folytonosság érzetét a 25 körülire beállított értékkel lehet elérni, ami még nem is eredményez túl nagy leterheltséget egy átlagos gép számára sem.

Az ActionScript kódokat létrehozhatjuk akár a Timeline 1. képkockájához rendelve, de magukon az egyes moziklip szimbólumokon is. Ez utóbbit úgy valósíthatjuk meg, hogy jobbklikk az objektumon és a helyi menüből az Actions parancsot kell választani.

További magyarázatot a megoldás egyes részeinél adok. Lássuk hát a megoldást!

- > Egy új dokumentumnak állítsuk be a méreteit és a másodpercenként lejátszott képkockák számát! (500x200 px, 25 fps)
- > Rajzoljunk egy kört a stage-re, majd készítsünk belőle Movie Clip szimbólumot „telekor” néven! A Registration-t állítsuk középre! Formázzuk a szimbólumot Radial típusú színátmenettel, így könnyen kelthetjük egy gömb illúzióját!
- > Húzzunk a szimbólumból 3 példányt a színpadra! Méretezzük őket, majd állítsuk be a színeiket! Ezt a Properties lap Color mezőjében való Advanced kiválasztásával lehet megtenni. A Settings... gombra való kattintás után az értékekkel kísérletezni kell, hogy szép gömböket kapjunk. Nagyon fontos, hogy minden szimbólum példányánál az Instance Name-nek adjuk a „golyo” értéket!
- > Kész vagyunk a golyókkal, most már „csak” a hozzájuk tartozó ActionScriptet kell megírni. Ezt pl. úgy lehet, hogy jobbklikk az egyik példányon, s a helyi menüből válasszuk az Actions parancsot. Illesszük be a következő kódot!

```
onClipEvent(load) {  
    xSeb = 6;  
    ySeb = 8;  
    r = this._width/2;  
}
```

A moziklip betöltődésekor 3 változót inicializálunk. Az „xSeb” a vízszintes sebességet, az „ySeb” a függőleges sebességet határozza meg, az „r” a golyó sugara.

```

onClipEvent(enterFrame) {
    xGolyo = this._x + xSeb;
    yGolyo = this._y + ySeb;
    if (yGolyo > (Stage.height - r)) {
        this._y = Stage.height - r;
        ySeb = -ySeb;
    } else if (yGolyo < 0 + r) {
        this._y = 0 + r;
        ySeb = -ySeb;
    } else {
        this._y += ySeb;
    }
    if (xGolyo > (Stage.width - r)) {
        this._x = Stage.width - r;
        xSeb = -xSeb;
    } else if (xGolyo < 0 + r) {
        this._x = 0 + r;
        xSeb = -xSeb;
    } else {
        this._x += xSeb;
    }
}

```

> Ezt a kódot a többi mozikliphez is hozzá kell rendelni, annyi változtatással, hogy az „xSeb” és az „ySeb” változóknak kell más értéket adni. Ezek után munkánk lejátszható.

Ezek az utasítások a moziklip összes képkockájában végrehajtnak.

Az x- és y-koordináták növelésekor folyamatosan ellenőrizni kell, hogy nem léptünk-e még ki a vásznról. Ha a golyó eléri valamelyik szélét a színpadnak, akkor a sebességet ellentettjére kell változtatni.

Még esetleg magyarázatra szorulhat a sugár. Mivel a labda regisztrációs pontja középen van, az „r” értéket a színpad széleinek koordinátához hozzáadjuk vagy kivonjuk, hogy a golyók szélei pontosan érintkezzenek a stage széleivel.

IV. Java - grafika

A lehetséges megoldások ismertetésekor nem térek ki a Java programozási nyelv ismertetésére, hiszen az dolgozatomnak nem célja. Itt is feltételezem, hogy bizonyos előismeretek már megvannak annál, aki nekirugaszkodik a feladatoknak. Az egyes osztályok, illetve metódusok magyarázatát lásd [1], [2]-ben!

Természetesen a feladatokat ezer meg egyféleképpen el lehet készíteni. Az itt szereplő feladatok megoldásai annyiban megegyeznek, hogy a **main()** metódusuk az adott ablak méretének beállításától eltekintve azonosak. Minden feladatot egy állományban készítettem el, ahol tulajdonképpen a **paint()** metódusokban van a megoldások lényege.

Az összes példánál szerepel a forrás, de az előbbiek miatt a **main()** metódust csak az 1. feladatnál írom le.

A **main()** metódusból külön is kiemelek két dolgot. Az egyik a program terminálása. Ha be akarjuk zárni az ablakot, akkor csak a CTRL+C billentyűkombinációra kapjuk vissza a promptot, a keret jobb felső sarkában való kattintással nem. Ez azért van, mert a program azzal, hogy a kis x gombra kattintunk, még nem fejeződik be. Ilyenkor csak a keret tűnik el.

A termináláshoz a keretet ki kell egészíteni a kilépés gombot kezelő eseményvezérlővel. Ezt a kezelőt a [WindowAdapter](#) osztályból kell származtatni, majd az **addWindowListener()**-rel még hozzá is kell rendelni a kerethez. Egyetlen feladata az, hogy kezelje az ablak bezárását. Ezt a **windowClosing()** eljárás átdefiniálásával tehetjük meg. Az eljárás paramétere a bezárást okozó [WindowEvent](#) típusú esemény, amit mi nem használunk, hanem csak egyszerűen kilépünk a programból.

A másik fontos dolog a **main()**-ben, szintén kapcsolatos az ablakkal. Arról van szó tudniillik, hogy a konstruálás során az ablakok nem láthatóak, azokat a **setVisible()** metódus teszi azzá. A megjelenítés előtt viszont meg kell adni az ablak méretét. A méretet beállíthatjuk fixre a **setSize()**-zal, vagy kiszámíttathatjuk a **pack()** metódussal.

Amelyik metódus még szinte mindegyik példában majd' ugyanaz, az az **init()**. Az applet minden egyes indításkor automatikusan meghívja az **init()** metódust, melyet csak a változók inicializálására szokás használni. Az appletben nem kötelező definiálni, de ha az applet nagy méretű, akkor elvárható egy **init()** a későbbi könnyebb megértés érdekében.

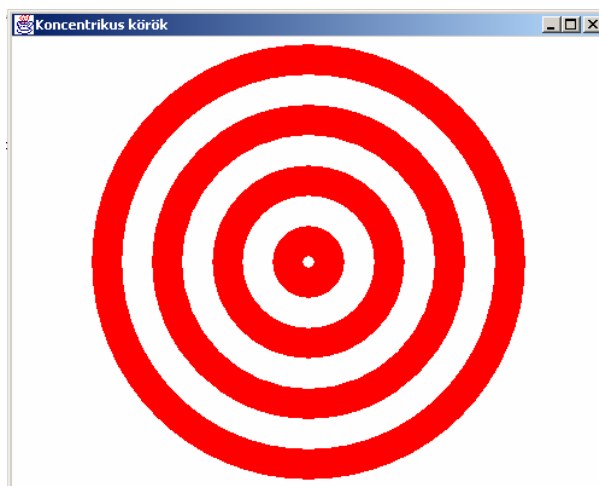
A papírral és festékkel való takarékoság okán csak a legelső feladatnál jelzem az ún. [import](#) utasításokat is.

A feladatgyűjtemény megoldási részében nem térek ki a koordináta-geometriai feladatok matematikai háttérére. Úgy vélem, ez sem dolgozatom témája.

A feladatok elkészítéséhez, megértéséhez úgy gondolom elég tájékoztatást nyújtanak a forrásállományban egyébként is elhelyezett megjegyzések. Ahol esetleg plusz információt éreztem szükségesnek leírni, ott ezt meg is tettem.

1.) Rajzoljon koncentrikus köröket a képernyő közepére, váltott színezéssel!

Ez a feladat feltételezi, a tanulók már ismerik, hogy miként lehet rajzolásra alkalmas grafikus felületet létrehozni Javában. Tudom, hogy az általam választott megoldás – gondolok itt az **Applet** osztály kiterjesztésére, majd annak **paint()** metódusának felülírására, illetve a **JFrame**-ek használatára – nem a legprofibb erre a célra, de legalább biztos vagyok benne, hogy tanulóim megértik.



33. ábra *Java-grafika_1*

A program forrása:

Kesz_Java_01

// A program koncentrikus köröket rajzol, váltott színezéssel.

`import java.awt.*;`

`import java.applet.*;`

`import javax.swing.*;`

`import java.awt.event.*;`

`public class Kesz_Java_1 extends Applet{`

`public void init(){`

`setBackground(Color.white);`

`}`

`public void paint(Graphics g){`

`Dimension d = getSize(); // Ezzel az ablak méreteit lehet lekérdezni.`

```

        int x = d.width/2;    // Az ablak szélességének fele.
        int y = d.height/2; // Az ablak magasságának fele.
        g.translate(x,y); /* A kezdő koordinátát az (x,y) pontba, vagyis a képernyő
közepébe helyezi át. Innentől, ha balra mozdulunk, az negatív x-et, ha föl az negatív y-t
jelent!!! */

        int r=180, i=1;
        Color szin1 = new Color(255, 255, 255); // A fehér szín beállítása
        Color szin2 = new Color(255, 0, 0); /* A piros szín beállítása. (Loki drukkerék
előnyben! :-)) */

        while (r>0){
            if (i%2==0) g.setColor(szin1);
                else g.setColor(szin2);

            i++;
            g.fillOval(-r, -r, 2*r,2*r);
            r -= 25;
        }
    }

    // ----- A main() metódus -----
    public static void main(String[] args) {
        JFrame jf = new JFrame("Koncentrikus körök");
        Applet applet = new Kesz_Java_1();

        jf.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {System.exit(0);}
        });

        jf.getContentPane().add("Center", applet);
        applet.init();
        jf.setSize(new Dimension(500,400)); // Az ablak méreteinek beállítása.
        jf.setVisible(true);

    }
}

```

2.) Rajzoljon felülnézetből látható piramist!

Még ez is igen egyszerű feladat. Viszont a négyzetek rajzolása helyett – szinte mindenkinek ez jut először eszébe – mutatok egy másik megoldást is. A **drawPolygon()** metódus használatát gyakorolhatják vele a tanulók.

a) Megoldás négyzetek segítségével – **drawRect()**

Kesz_Java_02a

// Piramis rajzolása felülnézetből

```
public class Kesz_Java_02a extends Applet{
    public void init(){
        setBackground(Color.white);
    }

    public void paint(Graphics g){
        Dimension d = getSize();
        int x = d.width/2;    // Az ablak szélességének fele.
        int y = d.height/2;   // Az ablak magasságának fele.
        g.translate(x,y);    // A kezdő koordinátát az (x,y) pontba helyezi át.
        int leptek =12;
        for (int i=10; i>=0; i--){
            g.drawRect( -leptek*i, -leptek*i, 2*leptek*i, 2*leptek*i );
        }
    }
}

// ----- IDE JÖN A main() METÓDUS -----
}
```

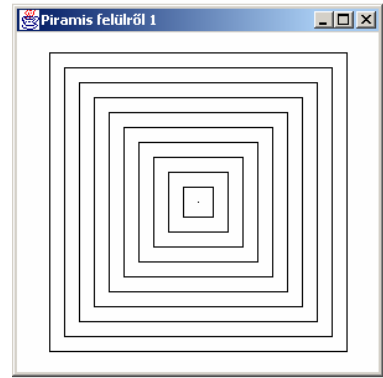
b) Megoldás a **drawPolygon()** használatával

Kesz_Java_02b

// Piramis rajzolása felülnézetből

```
public class Kesz_Java_02b extends Applet{
    public void init(){ setBackground(Color.white); }
    public void paint(Graphics g){
        Dimension d = getSize();
        g.translate(d.width/2, d.height/2);
        int leptek =12;
        for (int i=0; i<9; i++){
            int[] xTomb = {-100+i*leptek, 100-i*leptek, 100-i*leptek, -100+i*leptek};
            int[] yTomb = {-100+i*leptek, -100+i*leptek, 100-i*leptek, 100-i*leptek};
            g.drawPolygon(xTomb, yTomb, 4);
        }
    }
}

// ----- IDE JÖN A main() METÓDUS -----
}
```



34. ábra Java-grafika_2

3.) Rajzolja meg a magyar zászlót!

Ezen feladat megoldásával sem lehet programozói versenyt nyerni, de itt is lehet mutatni egy, a szokásostól – miszerint 3 vagy 2 kitöltött téglalapot kell rajzolni – merőben eltérő megoldást. Ha fehér háttérre rajzolunk jó vastag piros és zöld vonalat, akkor nagyon egyszerűen áll elő egy megoldás.

a) Megoldás négyzetek segítségével – **drawRect()**

Kesz_Java_03a

// Magyar zászló rajzolása

```
public class Kesz_Java_03a extends Applet{
    public void init() {
        setBackground(Color.white);
    }

    public void paint (Graphics g) {
        Dimension d = getSize();
        g.setColor(Color.red);
        g.fillRect(0,0, d.width,d.height/3);
        g.setColor(Color.green);
        g.fillRect(0,2*d.height/3, d.width,450);
    }
// ----- IDE JÖN A main() METÓDUS -----
}
```

b) Megoldás a **setStroke()** használatával

Kesz_Java_03b

// Magyar zászló rajzolása

```
public class Kesz_Java_03b extends Applet{
    public void init() { setBackground(Color.white); }
    public void paint (Graphics g) {
        Dimension d = getSize();
        Graphics2D g2 = (Graphics2D) g;
        // A rajzolt vonal tulajdonságait lehet állítani.
        BasicStroke stroke2 = new BasicStroke(d.height/3); // A vonal vastagsága.
        g2.setStroke(stroke2);
        g.setColor(Color.red);
        // Számolni kell a vonal vastagságával. A koordináták a vonal közepét határozzák meg.
        g.drawLine(0,d.height/6, d.width,d.height/6);
        g.setColor(Color.green);
        g.drawLine(0,5*d.height/6, d.width,5*d.height/6);
    }
// ----- IDE JÖN A main() METÓDUS -----
}
```



35. ábra Java-grafika_3

4.) Készítsen sakktáblát kirajzoló programot!

A látszat ellenére ez már nem is oly triviális feladat. Itt is mutatok két megoldást. Az előző feladatból merítve az ötletet, szaggatott vonalak rajzolásával megkapható a sakktábla. A másik megoldás a fekete-fehér négyzetek rajzolgatása.

a) Megoldás a **setStroke()** használatával

Kesz_Java_04a

// Sakktábla rajzolása

```
public class Kesz_Java_04a extends Applet{
    public void init(){
        setBackground(Color.white);
    }

    public void paint(Graphics g){
        Graphics2D g2 = (Graphics2D) g;
        g.setColor(Color.black);
        // A sakktábla szegélye.
        g.drawRect(50,50, 400,400);

        // A sakktábla bal alsó négyzete sötét!!!
        // 50 - a vastagság
        Stroke stroke = new BasicStroke(50, BasicStroke.CAP_SQUARE,
        BasicStroke.JOIN_ROUND, 0, new float[] {0,100}, 0);
        g2.setStroke(stroke);
        for (int i=0; i<4; i++){
            g.drawLine(125,75+i*100, 450,75+i*100);
        }
        for (int i=0; i<4; i++){
            g.drawLine(75,125+i*100, 400,125+i*100);
        }
    }
}
// ----- IDE JÖN A main() METÓDUS -----
}
```

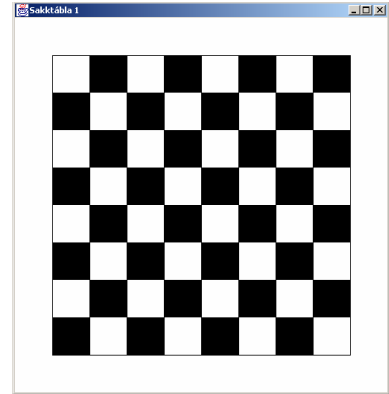
b) Megoldás **drawRect()** segítségével

Kesz_Java_04b

// Sakktábla rajzolása

```
public class Kesz_Java_04b extends Applet{
    public void init(){
        setBackground(Color.white);
    }

    public void paint(Graphics g){
```



36. ábra Java-grafika_4

```

        Dimension d = getSize();
        g.setColor(Color.black);
        // A sakktábla szegélye
        g.drawRect(50,50, 400,400);
        // Csak a páratlan indexösszegű négyzeteket kell rajzolni (azok lesznek feketék)!!!
        for (int i=1; i<=8; i++){
            for (int j=1; j<=8; j++){
                if ((i+j)%2!=0) g.fillRect(i*50,j*50, 50,50);
            }
        }
    }
}
// ----- IDE JÖN A main() METÓDUS -----
}

```

5.) Készítse el az alábbi ábrát rajzoló programot!

Itt sem a grafika az, ami a gondot okozza. A ciklust és még inkább a benne elhelyezkedő utasításokat kell jól kitalálni.

a) Megoldás a **drawPolygon()** használatával

Kesz_Java_05a

// Elforgatott négyzetek rajzolása

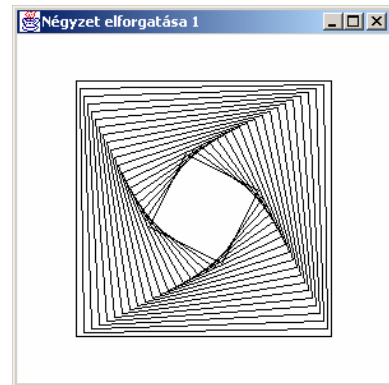
```

public class Kesz_Java_05a extends Applet{
    public void init(){
        setBackground(Color.white);
    }

    public void paint(Graphics g){
        Dimension d = getSize();
        int x = d.width/2;
        int y = d.height/2;
        g.translate(x,y);

        int leptek = 3;
        for (int i=0; i<20; i++){
            int[] xTomb = {-100+i*leptek, 100-i*2*leptek, 100-i*leptek, -100+i*2*leptek};
            int[] yTomb = {-100+i*2*leptek, -100+i*leptek, 100-i*2*leptek, 100-i*leptek};
            g.drawPolygon(xTomb, yTomb, 4);
        }
    }
}
// ----- IDE JÖN A main() METÓDUS -----
}

```



37. ábra Java-grafika_5

b) Megoldás **drawLine()** segítségével

Kesz_Java_05b

// Elforgatott négyzetek - másképpen

```
public class Kesz_Java_05b extends Applet{
    public void init(){
        setBackground(Color.white);
    }

    public void paint(Graphics g){
        Dimension d = getSize();
        int x = d.width/2;
        int y = d.height/2;
        g.translate(x,y);
        int xA=-100, xB=100, xC=100, xD=-100, yA=-100, yB=-100, yC=100, yD=100;

        for (int i=0; i<10; i++){
            g.drawLine(xA+i*20,yA, xB,yB+i*20);
            g.drawLine(xB,yB+i*20, xC-i*20,yC);
            g.drawLine(xC-i*20,yC, xD,yD-i*20);
            g.drawLine(xD,yD-i*20, xA+i*20,yA);
        }
    }
}

// ----- IDE JÖN A main() METÓDUS -----
}
```

- 6.) Készítsen programot, mely a felhasználótól bekéri egy egyenes egyenletét, majd megrajzolja az egyenest és annak x-tengelyre tükrözött képét a képernyő közepére helyezett koordináta-rendszerben!

Ez a „koordináta-geometriás” feladatok bevezetője. Egyik új probléma az adatok bekérése, másik pedig a tengelyek megrajzolása. Mindezek után pedig jöhet a két egyenes ábrázolás a koordináta-rendszerben.

Kesz_Java_06

// Egyenes és x-tengelyre tükrözött képének megrajzolása

```
public class Kesz_Java_06 extends Applet{
    double m, b;
    public void init(){
        setBackground(Color.white);
        adatBekeres();
    }
}
```

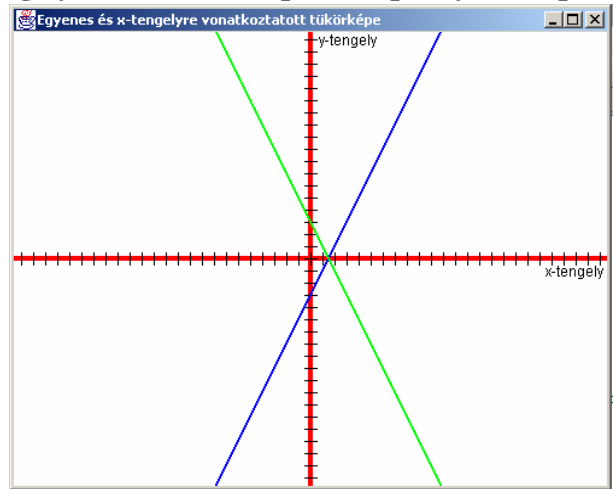
// Adatok bekérése a felhasználótól

```
public void adatBekeres(){
    String s;
    s = JOptionPane.showInputDialog("Mennyi az egyenes meredeksége (m)?");
    m = atalakit(s);
    s = JOptionPane.showInputDialog("Hol metszi az egyenes az y-tengelyt?");
    b = atalakit(s);
}
```

```
public double atalakit(String szoveg){
    double szam = Double.parseDouble(szoveg);
    /* Másképpen is át lehet konvertálni. Pl.:
       Double D = new Double(szoveg);
       double szam = D.intValue(); */
    return szam;
}
```

// A koordináta-rendszer megrajzolása

```
public void tengelyekRajzolasa(Graphics g){
    Dimension d = getSize();
    Graphics2D g2 = (Graphics2D) g;
    int xOrigo = d.width/2;
    int yOrigo = d.height/2;
    g.translate(xOrigo,yOrigo);
}
```



38. ábra Java-grafika_6

```

// A tengelyek megrajzolása
BasicStroke stroke = new BasicStroke(4);
g2.setStroke(stroke);
g.setColor(Color.red);
g.drawLine(-xOrigo,0, xOrigo,0);
g.drawLine(0,-yOrigo, 0,yOrigo);

int egyseg = 10; // lépték
g.setColor(Color.black);
stroke = new BasicStroke(1);
g2.setStroke(stroke);
// Az x-tengely beosztásainak megrajzolása
for (int i=0; i< xOrigo; i+=egyseg){ g.drawLine(i,-5, i,5); }
for (int i=0; i>-xOrigo; i-=egyseg){ g.drawLine(i,-5, i,5); }
// Az y-tengely beosztásainak megrajzolása
for (int i=0; i< yOrigo; i+=egyseg){ g.drawLine(-5,i, 5,i); }
for (int i=0; i>-yOrigo; i-=egyseg){ g.drawLine(-5,i, 5,i); }
g.drawString("x-tengely", 195,15); // Az (195,15) a szöveg alapvonalának bal
szélső pontja.
g.drawString("y-tengely", 7,-175);
}

public void paint(Graphics g){
    Graphics2D g2 = (Graphics2D) g;
    tengelyekRajzolasa(g);
    int egyseg = 10;
    BasicStroke stroke = new BasicStroke(4);
    stroke = new BasicStroke(2);
    g2.setStroke(stroke);
    g.setColor(Color.blue);
    // y = mx + b
    int y;
    for (int x=-200; x<200; x++){
        y = (int) (m*x + b*egyseg);
        y *= -1;
        g.drawLine(x,y, x,y);
    }
    /* Az egyenes tükrözése az x-tengelyre. Ez azt jelenti, hogy mind a meredekségnek, mind az
       y-tengellyel való metszéspontnak venni kell az ellentettjét. */
    g.setColor(Color.green);
    for (int x=-200; x<200; x++){
        y = (int) (-m*x + (-1)*b*egyseg);
        y *= -1;
        g.drawLine(x,y, x,y);
    }
}

// ----- IDE JÖN A main() METÓDUS -----
}

```

- 7.) Kérjük be egy háromszög csúcsainak koordinátáit és egy szög nagyságát! Ezek után rajzoljuk meg a háromszöget, majd a háromszög adott szöggel való origó körüli elforgatottját!

Az előző példához képest a kívánt matematikai ismeretek számítanak újnak a feladatban.

Kesz_Java_07

// Háromszög és elforgatottja

```
public class Kesz_Java_07 extends Applet{
    int egyseg = 20; // A lépték.
    double xA, yA, xB, yB, xC, yC, alfa;

    public void init(){

        setBackground(Color.white);
        adatBekeres();
    }

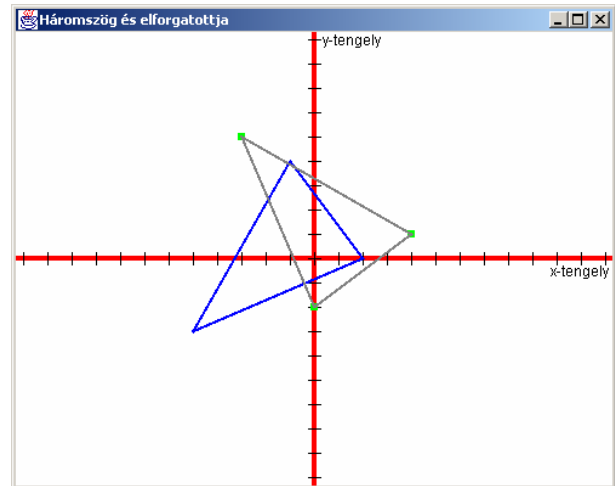
    public void adatBekeres(){
        String s;
        s = JOptionPane.showInputDialog("Az \"A\" csúcs x-koordinátája ");
        xA = atalakit(s);
        s = JOptionPane.showInputDialog("Az \"A\" csúcs y-koordinátája ");
        yA = atalakit(s);

        s = JOptionPane.showInputDialog("A \"B\" csúcs x-koordinátája ");
        xB = atalakit(s);
        s = JOptionPane.showInputDialog("A \"B\" csúcs y-koordinátája ");
        yB = atalakit(s);

        s = JOptionPane.showInputDialog("A \"C\" csúcs x-koordinátája ");
        xC = atalakit(s);
        s = JOptionPane.showInputDialog("A \"C\" csúcs y-koordinátája ");
        yC = atalakit(s);

        s = JOptionPane.showInputDialog("Az elfordulás szöge");
        alfa = atalakit(s);
    }

    public double atalakit(String szoveg){
        double szam = Double.parseDouble(szoveg);
        return szam;
    }
}
```



39. ábra Java-grafika_7

```

// A koordináta-rendszer megrajzolás
public void tengelyekRajzolasa(Graphics g){
    Dimension d = getSize();
    Graphics2D g2 = (Graphics2D) g;
    int xOrigo = d.width/2;
    int yOrigo = d.height/2;
    g.translate(xOrigo,yOrigo);

    // A tengelyek megrajzolása
    BasicStroke stroke = new BasicStroke(4);
    g2.setStroke(stroke);
    g.setColor(Color.red);
    g.drawLine(-xOrigo,0, xOrigo,0);
    g.drawLine(0,-yOrigo, 0,yOrigo);

    g.setColor(Color.black);
    stroke = new BasicStroke(1);
    g2.setStroke(stroke);
    // Az x-tengely beosztásainak megrajzolása
    for (int i=0; i< xOrigo; i+=egyseg){ g.drawLine(i,-5, i,5); }
    for (int i=0; i>-xOrigo; i-=egyseg){ g.drawLine(i,-5, i,5); }
    // Az y-tengely beosztásainak megrajzolása
    for (int i=0; i< yOrigo; i+=egyseg){ g.drawLine(-5,i, 5,i); }
    for (int i=0; i>-yOrigo; i-=egyseg){ g.drawLine(-5,i, 5,i); }
    g.drawString("x-tengely", 195,15);
    g.drawString("y-tengely", 7,-175);
}

public void paint(Graphics g){
    tengelyekRajzolasa(g);
    Graphics2D g2 = (Graphics2D) g;
    BasicStroke stroke = new BasicStroke(2);
    g2.setStroke(stroke);
    g.setColor(Color.blue);

    double x2A, y2A, x2B, y2B, x2C, y2C;

    // A háromszög megrajzolása
    g.drawLine( (int)xA*egyseg, (int)-yA*egyseg, (int)xB*egyseg, (int)-yB*egyseg );
    g.drawLine( (int)xB*egyseg, (int)-yB*egyseg, (int)xC*egyseg, (int)-yC*egyseg );
    g.drawLine( (int)xC*egyseg, (int)-yC*egyseg, (int)xA*egyseg, (int)-yA*egyseg );

    stroke = new BasicStroke(6);
    g2.setStroke(stroke);
    g.setColor(Color.green);
    // Az "A" pont elforgatottja
    x2A = ( xA*Math.cos(Math.toRadians(alfa)) + yA*Math.sin(Math.toRadians(alfa)) );
    y2A = ( -xA*Math.sin(Math.toRadians(alfa)) + yA*Math.cos(Math.toRadians(alfa)) );

```

```

/* Most kell int-té alakítani. Ha az x2A, y2A, stb. pontokat int-ként deklaráljuk, akkor
már az előző két sorban int-té kell alakítani a jobb oldalt. Az viszont nagyon pontatlan
számítást és megjelenítést eredményez.
TEHÁT, NEM MINDEGY, HOGY MIKOR KEREKÍTÜNK!!! */
g.drawLine( (int)(x2A*egyseg), (int)(-y2A*egyseg), (int)(x2A*egyseg), (int)(-y2A*
egyseg) );

// A "B" pont elforgatottja
x2B = ( xB*Math.cos(Math.toRadians(alfa)) + yB*Math.sin(Math.toRadians(alfa)) );
y2B = ( -xB*Math.sin(Math.toRadians(alfa)) + yB*Math.cos(Math.toRadians(alfa)) );
g.drawLine( (int)(x2B*egyseg), (int)(-y2B*egyseg), (int)(x2B*egyseg), (int)(-y2B*
egyseg) );

// A "C" pont elforgatottja
x2C = ( xC*Math.cos(Math.toRadians(alfa)) + yC*Math.sin(Math.toRadians(alfa)) );
y2C = ( -xC*Math.sin(Math.toRadians(alfa)) + yC*Math.cos(Math.toRadians(alfa)) );
g.drawLine( (int)(x2C*egyseg), (int)(-y2C*egyseg), (int)(x2C*egyseg), (int)(-y2C*
egyseg) );

// Az elforgatott háromszög megrajzolása
stroke = new BasicStroke(2);
g2.setStroke(stroke);
g.setColor(Color.gray);

g.drawLine( (int)(x2A*egyseg), (int)(-y2A*egyseg), (int)(x2B*egyseg), (int)(-y2B*
egyseg) );
g.drawLine( (int)(x2B*egyseg), (int)(-y2B*egyseg), (int)(x2C*egyseg), (int)(-y2C*
egyseg) );
g.drawLine( (int)(x2C*egyseg), (int)(-y2C*egyseg), (int)(x2A*egyseg), (int)(-y2A*
egyseg) );
}
// ----- IDE JÖN A main() METÓDUS -----
}

```

8.) Készítsen olyan programot, mely kirajzol egy tetszőleges szabályos sokszöget!

Kicsit kilóg a koordináta-geometriás feladatok közül, de jól szemlélteti a `Polygon` osztály használatát.

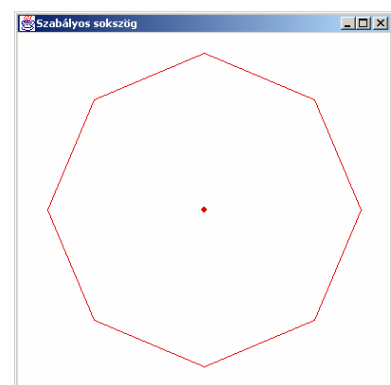
Kesz_Java_08

```

// Szabályos sokszög rajzolása
public class Kesz_Java_08 extends Applet{
    public void init(){
        setBackground(Color.white);
    }

    public void paint(Graphics g){
        Dimension d = getSize();

```



40. ábra Java-grafika_8

```

g.translate(d.width/2, d.height/2);
int n=8;      // A szabályos sokszög szögeinek a száma
int r=165;    // A sokszög köré írható kör sugara
Polygon sokszog = new Polygon(); // Egy példány a Polygon osztályból.
int alfa = 360/n;    // Egy körcikk szöge fokban!!!

for (int i=0; i<n; i++){
    int x=(int)(r*Math.cos(Math.toRadians(i*alfa)));
    int y=(int)(r*Math.sin(Math.toRadians(i*alfa)));
    sokszog.addPoint(x,y); // Pont hozzáadása a sokszöghöz
}
Color szin = new Color(228, 0, 0);
g.setColor(szin);
//g.fillPolygon(sokszog);
g.drawPolygon(sokszog);
g.fillOval(-3,-3,6,6);
}
// ----- IDE JÖN A main() METÓDUS -----
}

```

9.) Megrajzolandó a háromszög köré írt kör, ha a háromszög csúcspontjainak koordinátáit a felhasználótól kérjük be.

Ennél a feladatnál is talán elegánsabb megoldás lett volna a háromszög csúcsainak koordinátáit tömbben, vagy méginkább a Polygon osztályt felhasználva tárolni. Ez legyen két új feladat!

Kesz_Java_09

```

// Háromszög és köré írható köre
public class Kesz_Java_09 extends Applet{
    int egyseg = 20; // A lépték.
    double xA, yA, xB, yB, xC, yC;

```

```

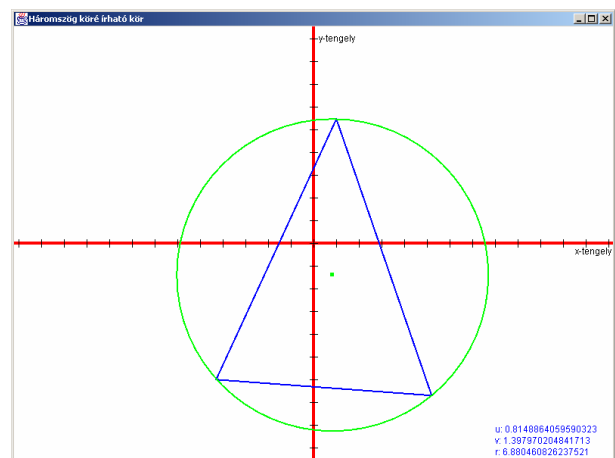
    public void init(){
        setBackground(Color.white);
        adatBekeres();
    }

```

```

    public void adatBekeres(){
        String s;
        s = JOptionPane.showInputDialog("Az \"A\" csúcs x-koordinátája ");
        xA = atalakit(s);
        s = JOptionPane.showInputDialog("Az \"A\" csúcs y-koordinátája ");
        yA = atalakit(s);

```



41. ábra Java-grafika_9

```

        s = JOptionPane.showInputDialog("A \"B\" csúcs x-koordinátája ");
        xB = atalakit(s);
        s = JOptionPane.showInputDialog("A \"B\" csúcs y-koordinátája ");
        yB = atalakit(s);

        s = JOptionPane.showInputDialog("A \"C\" csúcs x-koordinátája ");
        xC = atalakit(s);
        s = JOptionPane.showInputDialog("A \"C\" csúcs y-koordinátája ");
        yC = atalakit(s);
    }

    public double atalakit(String szoveg){
        double szam = Double.parseDouble(szoveg);
        return szam;
    }

    // A koordináta-rendszer megrajzolás
    public void tengelyekRajzolasa(Graphics g){
        Dimension d = getSize();
        Graphics2D g2 = (Graphics2D) g;
        int xOrigo = d.width/2;
        int yOrigo = d.height/2;
        g.translate(xOrigo,yOrigo);

        // A tengelyek megrajzolása
        BasicStroke stroke = new BasicStroke(4);
        g2.setStroke(stroke);
        g.setColor(Color.red);
        g.drawLine(-xOrigo,0, xOrigo,0);
        g.drawLine(0,-yOrigo, 0,yOrigo);
        g.setColor(Color.black);
        stroke = new BasicStroke(1);
        g2.setStroke(stroke);
        // Az x-tengely beosztásainak megrajzolása
        for (int i=0; i< xOrigo; i+=egyseg){ g.drawLine(i,-5, i,5); }
        for (int i=0; i>-xOrigo; i-=egyseg){ g.drawLine(i,-5, i,5); }
        // Az y-tengely beosztásainak megrajzolása
        for (int i=0; i< yOrigo; i+=egyseg){ g.drawLine(-5,i, 5,i); }
        for (int i=0; i>-yOrigo; i-=egyseg){ g.drawLine(-5,i, 5,i); }
        g.drawString("x-tengely", 340,15);
        g.drawString("y-tengely", 7,-275);
    }

    public void paint(Graphics g){
        tengelyekRajzolasa(g);
        Graphics2D g2 = (Graphics2D) g;
        BasicStroke stroke = new BasicStroke(2);

```

```

        g2.setStroke(stroke);
        g.setColor(Color.blue);

// A háromszög megrajzolása

// A koordináta-rendszer sajátossága miatt kell -1-gyel szorozni az y-koordinátákat
g.drawLine((int)(xA*egyseg),(int)(-yA*egyseg), (int)(xB*egyseg),(int)(-yB*egyseg));
g.drawLine((int)(xB*egyseg),(int)(-yB*egyseg), (int)(xC*egyseg),(int)(-yC*egyseg));
g.drawLine((int)(xC*egyseg),(int)(-yC*egyseg), (int)(xA*egyseg),(int)(-yA*egyseg));

// A körül írt kör megrajzolása
double M=xB-xA, N=yB-yA, O=xB*xB + yB*yB - xA*xA - yA*yA,
       P=xC-xB, Q=yC-yB, R=xC*xC + yC*yC - xB*xB - yB*yB;
double u,v,r; // A kör középpontjának koordinátái, ill. sugara
v = (M*R/2 - O*P/2) / (M*Q-N*P);
u = (O/2-v*N)/M;
r = Math.sqrt( (u-xA)*(u-xA) + (v-yA)*(v-yA) );
v *= -1;
g.drawString("u: "+u, 240,250);
g.drawString("v: "+v, 240,265);
g.drawString("r: "+r, 240,280);
g.setColor(Color.green);

// Az ellipszist az (x,y) bal felső koordinátájú, szélesség, magasság téglalap határolja!!!
g.drawOval( (int)((u-r)*egyseg), (int)((v-r)*egyseg), (int)(2*r*egyseg), (int)(2*r*
egyseg) );

// A kör középpontjának kirajzolása
stroke = new BasicStroke(5);
g2.setStroke(stroke);
g.drawLine((int)(u*egyseg),(int)(v*egyseg), (int)(u*egyseg),(int)(v*egyseg));
    }
// ----- IDE JÖN A main() METÓDUS -----
}

```

10.) Írjon parabolát rajzoló programot!

Ez a feladat a legegyszerűbb módon rajzol parabolát. Ez azt jelenti, hogy a felhasználónak a centrum (C) és fókuszpont (F) koordinátáit kell megadnia. Továbbfejlesztendő a feladat, miszerint a vezéregyenes egyenletét és a fókuszpontot kelljen megadni.

Kesz_Java_10

// Parabola rajzolása

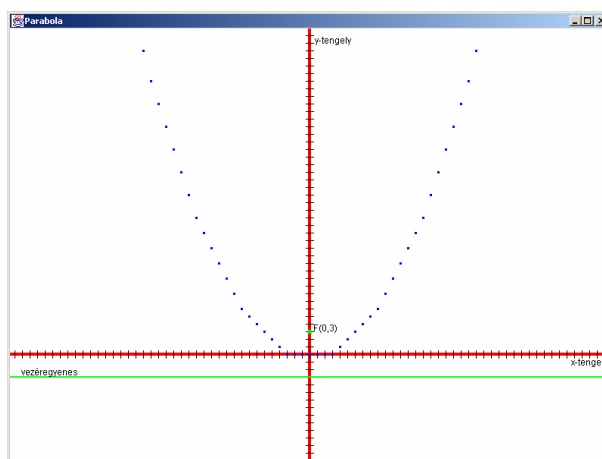
```
public class Kesz_Java_10 extends Applet{
    int egyseg = 10; // A lépték.
    double uC, vC, uF, vF;

    public void init(){
        setBackground(Color.white);
        adatBekeres();
    }

    public void adatBekeres(){
        String s;
        s = JOptionPane.showInputDialog("A centrum (C) x-koordinátája");
        uC = atalakit(s);
        s = JOptionPane.showInputDialog("A centrum (C) y-koordinátája");
        vC = atalakit(s);
        s = JOptionPane.showInputDialog("A fókuszpont (F) x-koordinátája");
        uF = atalakit(s);
        s = JOptionPane.showInputDialog("A fókuszpont (F) y-koordinátája");
        vF = atalakit(s);
    }

    public double atalakit(String szoveg){
        double szam = Double.parseDouble(szoveg);
        return szam;
    }

    // A koordináta-rendszer megrajzolás
    public void tengelyekRajzolasa(Graphics g){
        Dimension d = getSize();
        Graphics2D g2 = (Graphics2D) g;
        int xOrigo = d.width/2;
        int yOrigo = d.height/2;
        g.translate(xOrigo,yOrigo);
```



42. ábra Java-grafika_10

```

        // A tengelyek megrajzolása
        BasicStroke stroke = new BasicStroke(4);
        g2.setStroke(stroke);
        g.setColor(Color.red);
        g.drawLine(-xOrigo,0, xOrigo,0);
        g.drawLine(0,-yOrigo, 0,yOrigo);
        g.setColor(Color.black);
        stroke = new BasicStroke(1);
        g2.setStroke(stroke);
        // Az x-tengely beosztásainak megrajzolása
        for (int i=0; i< xOrigo; i+=egyseg){ g.drawLine(i,-5, i,5); }
        for (int i=0; i>-xOrigo; i-=egyseg){ g.drawLine(i,-5, i,5); }
        // Az y-tengely beosztásainak megrajzolása
        for (int i=0; i< yOrigo; i+=egyseg){ g.drawLine(-5,i, 5,i); }
        for (int i=0; i>-yOrigo; i-=egyseg){ g.drawLine(-5,i, 5,i); }
        g.drawString("x-tengely", 340,15);
        g.drawString("y-tengely", 7,-275);
    }

    public void paint(Graphics g){
        tengelyekRajzolasa(g);
        Graphics2D g2 = (Graphics2D) g;
        BasicStroke stroke = new BasicStroke(3);
        g2.setStroke(stroke);
        g.setColor(Color.blue);
        double yP, p;
        p = 2*(vF - vC);
        for (int xP=-50; xP<50; xP++){
            yP = 1.0/(2*p) * (xP-uC)*(xP-uC) + vC;
            g.drawLine(xP*egyseg,(int)-yP*egyseg, xP*egyseg,(int)-yP*egyseg);
        }

        // A fókuszpont (F) és a vezéregyenes (d) megrajzolása
        stroke = new BasicStroke(5);
        g2.setStroke(stroke);
        g.setColor(Color.green);
        g.drawLine((int)uF*egyseg, (int)-vF*egyseg, (int)uF*egyseg, (int)-vF*egyseg);
        stroke = new BasicStroke(2);
        g2.setStroke(stroke);
        Dimension ablakmeret = getSize();
        g.drawLine(-ablakmeret.width/2,(int)(p/2.0)*egyseg, ablakmeret.width/2,
(int)(p/2.0)*egyseg);
        g.setColor(Color.black);
        g.drawString("F(" +uF+ ", " +vF+ ")", 5, (int)-vF*egyseg);
        g.drawString("vezéregyenes", -ablakmeret.width/2+10, (int)vF*egyseg-2);
    }
    // ----- IDE JÖN A main() METÓDUS -----
}

```

11.) Rajzoljon paraméterezhető arkhimédeszi spirált a képernyő közepére! (csigavonal)

Ennél a feladatnál már egy pont polárkoordinátáit használjuk.

Az elméletet lásd [11]-ben!

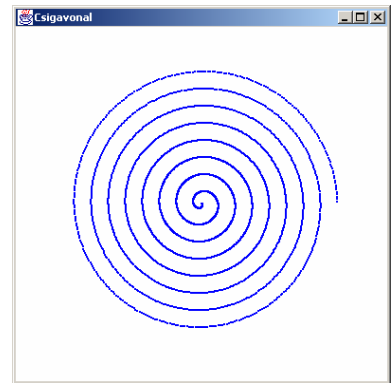
Kesz_Java_11

// Csigavonal kirajzolása

```
public class Kesz_Java_11 extends Applet{
    public void init(){
        setBackground(Color.white);
    }

    public void paint(Graphics g){
        Dimension d = getSize();
        g.translate(d.width/2, d.height/2);
        Graphics2D g2 = (Graphics2D) g;
        double x, y, r;
        final double k = 0.05;
        BasicStroke stroke = new BasicStroke(2);
        g2.setStroke(stroke);
        g.setColor(Color.blue);
        for (int alfa=0; alfa<8*360; alfa++){ // A 8 a körfordulások száma.
            r = k*alfa;
            x = r*Math.cos(Math.toRadians(alfa));
            y = r*Math.sin(Math.toRadians(alfa));
            g.drawLine((int)x, (int)y, (int)x, (int)y);
        }
    }
}

// ----- IDE JÖN A main() METÓDUS -----
}
```



43. ábra Java-grafika_11

Összefoglalás

Dolgozatom végén kijelenthetem: a feladatgyűjtemény még nincs kész! Azt hiszem, soha nem is lesz. Persze valahol sejtettem én már ezt a legelején, de bíztam benne, hátha sikerül egy kerek példatárat összeállítanom. Nos, munkám végeztével be kell ismernem, hogy ez nem sikerült. Még sincs bennem semmi rosszérzés, mert tudom, hogy olyan feladatgyűjtemény, mely minden igényt kielégít nincs is. Úgy vélem, ha másra nem is jó ez a néhány feladat, az én munkámat biztosan hatékonyabbá teszi a jövőben.

A feladatok kiválasztásában azt hiszem érhetnek jogos bírálatok, de ezen 41 feladat majdnem mindegyikét a gyakorlatban is kipróbáltam már tanulóimmal. Azt hiszem, az ő lelkesedésük igazolja, hogy vannak olyan ismeretek, melyeket a tanulók az iskolai tantárgyakra egyébként jellemző „nyűg” szó nélkül próbálnak elsajátítani.

Az első fejezetben található elméleti kérdések ugyanolyan fontos részét képezik a tananyagnak, mint a gyakorlati feladatok. Diákjaim visszajelzéséből tudom azt is, hogy azok, akik rendszeresen dolgoznak valamilyen grafikai programmal, hasznosnak ítélik az elméleti megalapozást, s megértik, miért is van nagy jelentősége az azokra fordított időnek.

A második fejezetben található feladatok segítségével a tanulók megismerkedhetnek egy professzionális rajzoló és képszerkesztő program használatával. Az itt megszerzett tudás segítségükre lesz – hogy csak három dolgot említsek – bonyolultabb számítógépi grafikák elkészítésében, esetleg egy másik hasonló szoftver használatában vagy éppen fényképek retusálásakor.

A harmadik fejezet példáival áttekintettük a web technológiák egyik nagyon fontos területét, melyek után a tanulók képesek lesznek a flash-technika alkalmazásával tetszetősebbé tenni weboldalaikat. A Flash segítségével a diákok kedvük szerint formálhatják honlapjaikat, de akár komolyabb multimédiás programokat is készíthetnek. A szoftver segítségével megalkotott objektumok mozoghatnak, átszíneződhetnek és még különböző eseményeket is kiválthatnak, melyek alapját képezik az interaktivitásnak. Ennek a fejezetnek egy nagy előnye még, hogy bepillantást nyújt a vektorgrafika világába, jártasságot adva a tanulóknak egy ma oly hatásos technikában.

A legtöbb GIMP és Flash gyakorlati feladatot a jövőben úgy kívánom még érdekesebbé tenni, ha a tanulók hozott anyaggal, azaz saját fényképeikkel dolgoznak majd. Meggyőződésem

ugyanis, hogy ez egy olyan motivációs lehetőség a grafikában, mely más tárgyaknál nincs meg. Ezt pedig nem szabad kihasználatlanul hagyni.

A negyedik fejezetben a diákok megtanulják a Java grafikus alkalmazásainak alapjait. Az igazság az, hogy legnagyobb hiányérzetem a szakdolgozatomban ezzel a fejezettel kapcsolatban van. Egyrészt nekem is rettentő sokat kell még fejlődnöm ebben a programozási nyelvben, másrészt valóban olyan megoldásokat kell keresnem, melyek megértése elvárható az általam tanított diákoktól.

Az előzetes várakozásaimmal ellentétben, élveztem ezt a munkát. Az elején féltem, hogy túl nagy fába vágom a fejszém, de aztán a feladatok hasznossága és az, hogy én is egyre jobban belemélyedtem az adott témákba, élvezetessé tette számomra a diplomamunka elkészítésének idejét.

Végezetül úgy gondolom, szép feladatot vettem nyakamba a példatár kidolgozásával, melynek sosem lesz vége – legalábbis míg a tanári pályán maradok. De már most azon töröm a fejem, hogy a folytatásban 4 külön részre fogom bontani a feladatgyűjteményt, s külön-külön folytatom majd a munkát.

Köszönetnyilvánítás

Elsősorban köszönöm feleségemnek, Editnek, hogy áldozatkészségével megfelelő időt, körülményt tudott teremteni szakdolgozatom elkészítéséhez. Sokszor annyi türelmet és biztatást kaptam tőle, hogy nélküle biztosan nem tudtam volna talán még elkezdni sem a munkát.

Másodsorban köszönöm 14 hónapos kislányomnak, Petrának, hogy valahányszor leültem a számítógép elé, ő mindannyiszor mellettem termett, s „óriási szakértelmével” sikerült nem beletörölnie a készülő munkába. Sőt, az egeret és billentyűzetet is – kisebb harcok árán ugyan, de – sikerült nem tönkretennie. Ha pedig néha fáradtnak és lehangoltnak éreztem magam, ő két másodperc alatt mindig fölvidított, ezzel erőt adva a továbbiakhoz.

Harmadsorban köszönet illeti dolgozatom témavezetőjét, Tomán Henriettát, aki építő kritikáival, precizitásával és segítőkész hozzáállásával támogatta munkámat.

Irodalomjegyzék

- [1] Angster Erzsébet: Objektumorientált tervezés és programozás 1, 4KÖR Bt., 2002.
- [2] Angster Erzsébet: Objektumorientált tervezés és programozás 2, 4KÖR Bt., 2002.
- [3] Berke József, Virág Miklós: Számítógépes grafika és prezentáció, Keszthelyi Akadémia Alapítvány, 1998.
- [4] Carey Bunks: Egy korty GIMP, Typotex Kft., 2002.
- [5] Gurdy Leete, Elle Finkelstein: Flash 5 – Dummies könyvek, Kossuth Kiadó, 2002
- [6] Jan P. Cibula, Simon Kagi, Sebastian Michel: Macromedia Flash MX Professzionális webdesign akciószkriptekkel, Extra-Plan Kft., 2002.
- [7] Sziklai János: Flash 8 egyszerűen, Perfact, 2006.
- [8] www.inf.u-szeged.hu/oktatas/jegyzetek/KubaAttila/opengl/alt.xml
- [9] www.om.hu/letolt/kozokt/erettsegi2005/tanaroknak/informatika/informatikabe.htm
- [10] mathworld.wolfram.com/Whirl.html
- [11] www.hmg.hu/tanarok/vz/arany.pdf