Debreceni Egyetem

Informatikai Kar

GRAFIKUS JAVA ALKALMAZÁS

Témavezető:

Dr. Tornai Róbert

Egyetemi adjunktus

Készítette:

Pick Szabina

Programtervező Informatikus

Debrecen

2011

Tartalomjegyzék

1. Bevezetés

Szakdolgozatom témája a Java SE grafikus felületébe való bevezetés egy képeket szerkesztő program segítségével. Azért választottam ezt a témát, mert érdekel a grafikus ábrázolás objektum orientált környezetben. Célom egy java alkalmazás megalkotása, melyben bemutatom a pixelgrafika alapjait, melyekkel fotókat, ábrákat és hasonló objektumokat lehet manipulálni. A régi képek, melyeket nagyanyáink korában készítettek már elmosódtak, elhalványultak. Sokfajta alkalmazással lehet helyrehozni ezeket a beszkennelt képeket, amelyeken újra látni lehet a körvonalakat ezután. Én is egy ehhez hasonló program írását tűztem ki célul java Swinges felületen. Ez kiválóan alkalmas desktop alkalmazások készítésére és könnyebben, gyorsabban alkothatók meg vele a szoftverek, mint C nyelven. Egy olyan alkalmazás megalkotása a célom, mellyel helyre lehet hozni elmosódott illetve zajos képeket. A programomban a „Zajok eltűntetése” menüpont kiválóan alkalmas egy ábrán található só, bors és egyéb zajok eltűntetésére. A kép élesítésére, melyet megtehetünk egy erősebb, illetve egy enyhébb hatású algoritmussal, az „Élesít (erősen), valamint az „Élesít (enyhén)” menüpontok használhatók a szoftveremben. A fotóknak ki lehet emelni az éleit, hogy fekete háttéren a kép élei látszódjanak fehérrel, valamint lehet választani olyan funkciót is, mellyel ugyanez látható, csak az élek színesek lesznek, az eredeti ábrával egyező színűek, mindezt a példaprogramom „Élek” és „Színes élek” menüpontjai segítségével tehetjük meg. A szoftveremmel ezen kívül olyan változtatást is lehet eszközölni a képen, melynek eredménye egy módosított ábra lesz, amely úgy néz ki, mintha egy kőbe karcolták volna. A programom magában foglal képmanipuláló funkciókat, melyek közül a legalapvetőbbek a kép szürkeárnyalatossá tétele, illetve a kép csak fekete valamint fehér színekkel való ábrázolása. Ezenkívül tartalmaz olyan eljárásokat, mint a világosítás, sötétítés, kontrasztosítás és olyan további pixelgrafikai algoritmusokat, mint a fotó két színnel való megjelenítése: fehér illetve egy felhasználó által kiválasztott szín által, ahol a fehér lesz a képen található legsötétebb, a választott szín pedig a legvilágosabb árnyalat. Legvégül a programom tartalmazza az alapvető fájlkezelési eljárásokat (megnyitás, mentés, kilépés) mindezt grafikus felületen. Meglévő matematikai eljárásokat felhasználva (például konvolúció) alkottam meg ezt a szoftvert. Elsőként a fénnyel és annak a szemre gyakorolt hatásával, a látással foglalkozok, mivel a szem által érzékelhetjük a képeket, amelyeken a későbbiekben a példaprogramom segítségével változtatásokat lehet végrehajtani.

2. A fény

A fény már régóta izgatja az emberiség fantáziáját, például a tüzet kísérő fény, a villám és az általa gyújtott tűz, az égitestek [1]. Ezen jelenségek vallási kultuszok sorozatát hozta létre. A fény forrásában és az, ezáltal kibocsátott fényben istenséget láttak, például a napimádó ősi Egyiptomban, melynek főistene a napisten, Ré, vagy az ókori görögök főistene, Zeusz a villámokat (is) uralta. Ezen vallási hiedelmek voltak az első fénymagyarázatok.

Ugyanakkor az ókor bölcselői már olyan kérdéseket vetettek fel, hogy „hogyan látunk?", „hogyan jön létre a látás?". Erre a kérdésre Platón és Euklidész válasza az, hogy a látást a szemből kiinduló fénysugárnak a tárgyakkal való ütközése kelti. Arisztotelész viszont elveti ezt az elvet, mivel ezen feltevés alapján a sötétben is látnánk, tehát soha nem lenne sötét. Szerinte a szem és a látott tárgy között valamilyen közvetítő anyagnak kell lennie.

Az atomisták azt vallották, hogy a látást a valaki szemébe érkező, a testekről minden pillanatban leváló leheletvékony rétegek keltik. Úgy gondolták, hogy a testről finom anyagi részecskék áramlanak ki és ezen részecskék összessége a fény.

Tulajdonképpen Newton is anyagi részecskék kisugárzásának tekintette a fényt. Azonban ezzel az elmélettel nem tudta megmagyarázni, hogy a fény bizonyos új közeg határához érve az a közeg miért csak a fény egy részét engedi a közegbe, a többit pedig visszaveri. Ezt a problémát úgy magyarázta, hogy a fényrészecskéknek kétféle állapotuk van. Az egyik állapotban képesek beszivárogni az átlátszó anyagba, a másikban nem. Egy fényrészecske mindkét állapotot felveheti, sőt fel is veszi periodikusan.

A hullámelméletnek is sok korai képviselője ismert. Leonardo da Vinci szerint a fény, a hang és a vízhullám alapvető törvényei közösek. Grimaldi véleménye az volt, amelyet 1665-ben nyilvánosságra is hozott, hogy ha fényhez fény adódik, akkor ennek következménye sötétség is lehet. Robert Hooke kifejtette, hogy fény transzverzális rezgés, bár nem bizonyította. Gaston Paradies szerint, mint a halála után megjelent művében is leírta, a fényrezgések úgy terjednek az általa feltételezett éterben (amelyet igen finom anyagi közegnek képzelt), mint a hang a levegőben. A hullámelmélet megalapozójának tartott Huygens pedig az általa alkotott törvény szerint azt vallotta, hogy a fényforrás egy pontjából kiinduló fényrezgés az éterben minden irányban elterjed, és bizonyos idő múlva egy felületre érkezik. Úgy képzelte, hogy az éter részecskéi rugalmas golyócskák, és a fény terjedését az egyenes mentén sorakozó

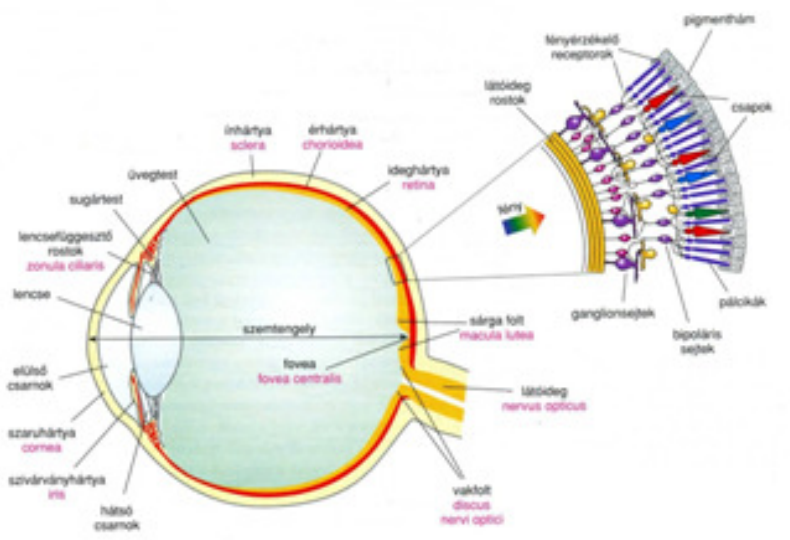
rugalmas golyók ütközésének mintájára feltételezte. A fényt longitudinális hullámnak mondta (tehát a részecskék a terjedés irányába rezegnek). Ez volt Huygens tévedése. Frensel tökéletesítette a Huygens-féle elvet. Úgy gondolta, hogy a pillanatnyi hullámfelületen a fényrezgések ugyanolyan fázisban vannak, tehát rezgési állapotuk megegyező, és hogy a tér valamely pontjában a világosságot az odaérkező fényhullámok találkozásának eredménye, a hullámok interferenciája szabja meg.

3. Emberi szem

Az elektromágneses sugárzás körülbelül 400 nm és 800 nm közötti hullámhosszú tartományát látható fénynek nevezzük, ezen intervallumba eső elektromágneses sugarakat fogja fel az egészséges emberi szem [2]. Az infravörös (800 nm – 1,3 µm) és az ultraibolya (10 nm – 400 nm) elektromágneses spektrumai határolják a látható fény tartományát. (Léteznek azonban olyan állatfajok, amelyek képesek érzékelni a számunkra nem látható ultraibolya sugarakat.) A szem sokféle szerepet tölt be a látásban. A környezet optikai leképezésében, az alkalmazkodásban a változó fényintenzitásokhoz, ezen kívül a fény idegimpulzussá, de előtte elektrokémiai jellé alakításában és a képi információ előzetes kiértékelésében vesz részt.

A szem vázlatos szerkezete:

Egy körülbelül 2,5 cm átmérőjű gömb alakú szerv az emberi szem, amely a belsejében uralkodó 10-22 Hgmm (1,3-2,9kPa) túlnyomásnak köszönheti formáját. A szemgolyó 3 rétegből áll, vázlatos szerkezetét az alábbi ábra mutatja (3.1-es ábra). A 3.1-es ábrát a Damjanovich Sándor, Fidy Judit és Szöllősi János által írt Orvosi biofizika nevű könyvből scanneltem.



3.1-es ábra.

Az ínhártya (sclera) a legkülső erős fehér burok, amely elöl átmegy az átlátszó szaruhártyába (cornea). A szivárványhártya (iris), a sugártest (corpus ciliare) és az érhártya (choroidea) alkotja a középső réteget. A lencsefüggesztő rostok (zonula ciliaris) rögzítik a lencsét, ezek a sugártesthez kapcsolódnak hozzá. A pupilla az iris közepén található nyílás. Az ideghártya (retina) a legbelső réteg, ez tartalmazza a fényreceptorokat is.

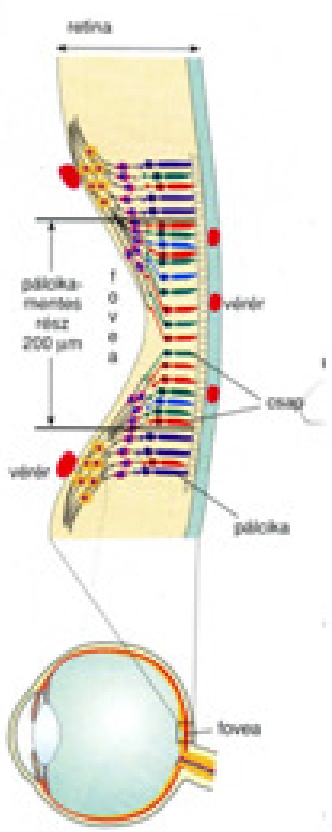
A fényingert az az ideghártya, amely a központi idegrendszer központi részét képzi, fogja fel a fényingert és továbbítja a kiváltott ingerületet az agy felé. Emellett a retina elkezdi a vizuális információ értelmezését is, például felismer mozgásokat és megvilágításbeli kontrasztokat. A látási információ feldolgozásában a retinában több, mint 60 fajta idegsejt vesz részt, amelyek altípusait, és azok elrendeződését az alábbi ábra szemlélteti.

Két csoportját különböztetjük meg azoknak a sejteknek, amelyek a retinára eső fény érzékelését végzik: csap- és pálcikasejteket. Egy egészséges emberi szemben a pálcikasejtek száma 120 millió, a csapoké 6,5 millió. A normális fényintenzitások melletti nappali ($1\text{-}10^5$ lux) látásért a csapok, a pálcikák pedig a szürkületi (10^{-9} – 10 lux) látásért felelősek. A horizontális, bipoláris, amakrin- és ganglionsejtek négy fő típust különböztetjük meg a látási információ előzetes kiértékelésében részt vevő idegsejtek közül. A ganglionsejtek axonjai képezik a szemet elhagyó látóideget. Egyetlen ganglionsejtre jut a retinában több egymás mellett elhelyezkedő receptorsejtből származó inger. Az ingerületi jel konvergenciájának nevezzük ezt a jelenséget. A csapsejtek jele gyengébben konvergál, mint a pálcikasejteké. Több serkentő vagy gátló jellegű szinapszis befolyásolja mindkét típusú receptorsejt működését. A retinális képkiértékelésben van fontos szerepük ezeknek a sejteknek.

A retinában a receptorsejtek az érhártya felöli oldalon helyezkednek el, nem pedig a szem belseje felé fordulnak, az az emberi szem egy érdekessége. A pigmentált epitheliumnak és a fényérzékeny sejteknek az érintkezése fontos a megvilágítás után a fotoreceptorok gyorsabb regenerációjához. Ugyanakkor az ilyen elrendeződés azzal jár, hogy a fénynek ahhoz, hogy a fényreceptorokat elérje, át kell haladnia a retinán. Ugyanez azt is eredményezi, hogy ahhoz, hogy az idegrostok a fényreceptorokat elérjék, át kell haladniuk a fényérzékeny sejteket tartalmazó rétegen. Ugyanez azt is eredményezi, hogy az idegrostoknak, hogy ahhoz, hogy kijussanak a szemből át kell menniük a fényérzékeny sejteket tartalmazó rétegen. Fényreceptorok nem találhatók a látóideg becsatlakozásának helyén. Ezt a vakfoltnak nevezzük. Létét normális körülmények között létét nem érzékeljük Ennek oka, hogy az agy pótolja a vakfoltra eső hiányzó képrészletet annak környezetével, vagy a másik szemből jövő információ alapján (a két szem vakfoltja a látómező más-más részére esik).

A látómező közepén elhelyezkedő tárgyak képe a retina a sárga foltnak (macula lutea) nevezett részére képződik le. A fovea az ennek a közepén található mélyedés, ahol a csapsejtek sűrűsége a legnagyobb, és nincsenek pálcikasejtek. A fény akadálytalanabb

érzékelése érdekében erről a helyről a következő ábrán (3.2-es ábra) látható módon oldalra tolódnak a vérerek és az ingerület feldolgozásában részt vevő idegi elemek. A pigmentált epithelium a receptorsejtek mögött helyezkedik el, amely elnyeli a rá eső fényt a nagy melanintartalmának köszönhetően, és így csökkentve a nemkívánatos visszaverődéseket. A 3.2-es ábra a Damjanovich Sándor, Fidy Judit, Szöllősi János: Orvosi biofizika könyvből lett scannelve.



3.2-es ábra

A látási ingerület kialakulása a retinában:

A fotoreceptor sejtek, a retinában lévő idegsejtek, és az agy együttes működésének az következménye a látás. A fotokémiai folyamatok, amelyek a retinára eső fény eredményeként jönnek létre, a fényérzékeny sejtek állandó neurotranszmitter-szekcióját modulálják. Egy elektromos ingerületi hullám jön létre ennek eredményeképpen, amely az agykéreg megfelelő részébe jutva a látóidegen keresztül, ott látásérzetet vált ki. Az agy megfelelő részében mindig

fényérzetet eredményez a retina érzékelő sejtjeinek ingerlése. Ez akkor is így van, ha a kiváltó inger nem fény volt, hanem például a szemet ért nyomás vagy ütés. A képi információ kiértékelése (például fénykontrasztok vagy mozgás detektálása) már a retinában elkezdődik, az ideghártya bizonyos ingereket kiemel, másokat elnyom, mielőtt az agy felé továbbítaná.

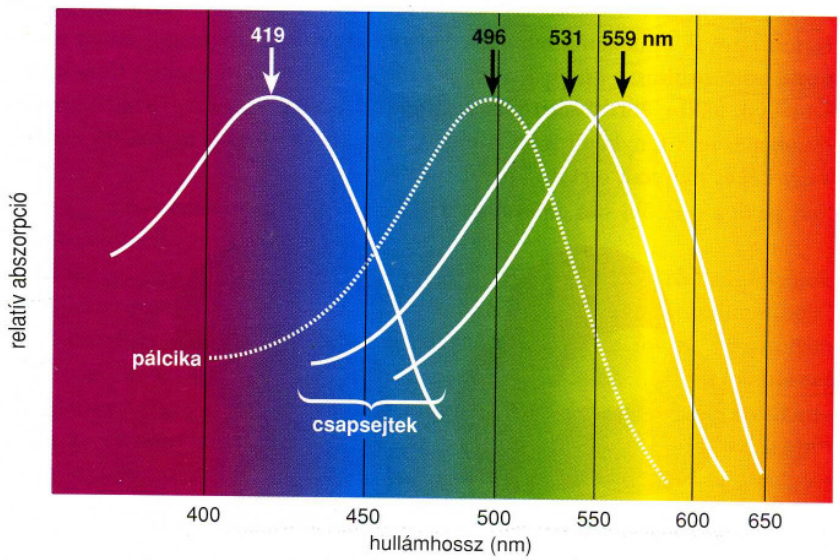
4. Színek

A látható színkép az elektromágneses hullámok rendkívül nagy hullámhossztartományának csak igen kis részét foglalja el. Az infravörös színképtartomány a vizsgálatok szerint a látható színkép hosszú hullámú részeihez csatlakozik, átnyúlván az elektromos hullámok tartományába is, a határon egyenáram állna. Az ultraibolya tartomány a látható színkép rövidhullámú részén túl kezdődik, majd a röntgensugarak és a radioaktív γ-sugarak következnek. A kozmikus sugárzás elektromágneses részének még rövidebb a hullámhossza. A látható színkép határát fiziológiai adottságok határozzák meg, ezért gyakran a láthatóság tartományán kívül eső elektromágneses hullámokat is fénynek hívják (pl. infravörös fény, röntgenfény) [3].

A fehér fény sok színből összetett fény, erre a fontos felismerésre Newton jött rá, a prizmák vizsgálata során [4]. Majd kísérletét folytatva a szétszórt színeket egy másik, fordított prizmán vetette keresztül, ekkor a prizmát elhagyó fény ismét fehér. Ernyővel, melyen kis rést hagyott, a színkép egyes színeit külön vizsgálta. Ha a második prizmára egy színt (monokromatikus színt) vitt, akkor az változatlanul haladt át rajta.

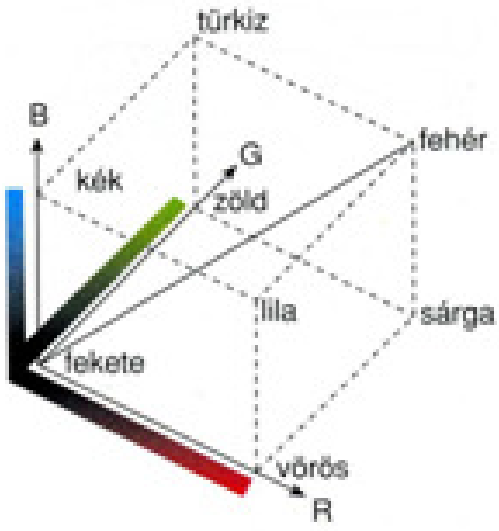
A szín létrehozásának 2 fajtája van, az egyik a szubtraktív, a másik az additív színalkotás. A szubtraktív eljárás az eredeti fehér fényből bizonyos színkomponenseket elnyeléssel, abszorpcióval kivon. A szűrők csak elvesznek. Az additív színalkotás viszont például a színházban, amikor az előadás folyamán valamilyen színes fénnyel világítják be a színpadot, tehát hozzáadnak még valamilyen színű fényt.

Különbséget tudunk tenni a látható fénysugarak között, azok színe, tehát hullámhossza alapján (4.1-es ábra). A spektrum minden egyes hullámhosszához hozzárendelhetünk egy színt, viszont hullámhossz nem minden színhez rendelhető hozzá egyértelműen. A fénysugarak, melyek azonos frekvenciájúak is képesek különböző színérzetet kelteni a fényintenzitás és a környezet megvilágítása szerint. A színek érzékelését három különböző spektrális tulajdonságú csapfajta teszi lehetővé, ez a Young-Helmholtz-elmélet. Az emberi szemben a csapok 64%-a vörös, 32%-a zöld és 2%-a kék. Tiszta, keverék és komplementer színeket különböztethetünk meg. A 4.1-es ábrát a Damjanovich Sándor, Fidy Judit, Szöllősi János: Orvosi biofizika könyvből szkenneltem.



4.1-es ábra

A következő ábra (4.2-es ábra) a színek additív keverésének illusztrációja a vörös (red, R), zöld (green, G), kék (blue, B) színeket választva alapszíneknek. A 4.2-es ábra a Damjanovich Sándor, Fidy Judit, Szöllősi János: Orvosi biofizika könyvből lett szkennelve.

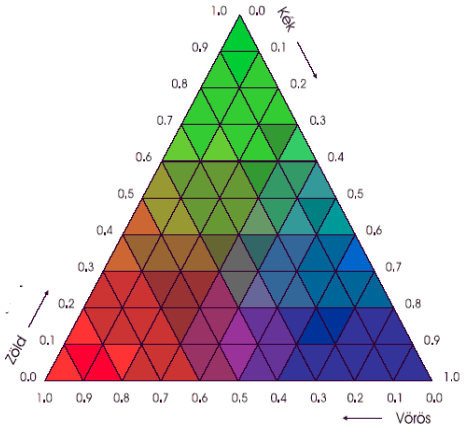


4.2-es ábra

5. Számítógépes színrendszerek

RGB:

Bármely szín előállítható a piros, a zöld és a kék színek bizonyos kombinációjával, egy képpont tehát a piros, a zöld és a kék (RGB (red-green-blue)) árnyalataiból áll. Mind a 3 színkomponensnek 256 esete van, mivel az adott árnyalat a 0 értéktől kezdve 255-ig vehet fel értékeket. Additív színrendszer. A három alapszín (piros, zöld, kék) egyforma keverése fehér, hiányuk feketét eredményez, tehát a fekete: 0,0,0. A kék szín: 0,0,255, a sárga: 255,255,0, a piros: 255,0,0 és a zöld: 0,255,0. Mivel egy színkomponenst 1 byteon ábrázol, ezért egy árnyalatot 3 byteon, azaz 24 biten. Ennek következtében összesen 255*255*255, tehát 16,7 millió szín állítható össze ezen színrendszer segítségével (5.1-es ábra). Az 5.1-es ábrát az internetről szedtem a http://www.mfk.unideb.hu/userdir/dmk/docs/20071/07_1_11.pdf oldalról [5].

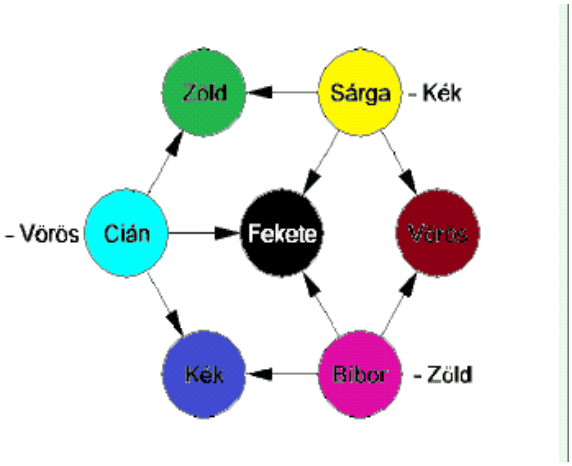


5.1-es ábra. Az RGB színredszer egyik alapmodellje

CMY:

Az RGB színrendszer alap színeinek kiegészítő színeit tartalmazza, tehát a cyan (türkiz), magenta(bíbor) és sárga (5.2-es ábra). A fehér fényből (piros+zöld+kék) a cyan kivonja a komplementer vöröset, így kékeszöldet, azaz cyan-t kapunk. Ugyanez a helyzet a bíborral és a sárgával is. A bíbornál a fehér fényből a bíbor komplementerét, a zöldet vonja ki, itt az eredmény vörös és kék, azaz magenta lesz. Tehát ez szubtraktív színrendszer. A színek hiánya fehéret eredményez. A sárgánál pedig a fehér fényből a komplementer kéket vonja ki és keletkezik a sárga. Itt is mindegyik árnyalatnak 256 esete van. Egy színkomponens 1 byteon

van ábrázolva, tehát 0-tól 255-ig vehet fel értéket. Mivel 3 színkomponens van (türkiz, bíbor, sárga) és mindegyik 1 byteon ábrázolva, tehát összesen 3 byteon, azaz 24 biten, ezért 255*255*255, azaz körülbelül 16,7 millió szín állítható elő ezzel a színrendszerrel. A CMY színeket nyomtatáshoz használják. Mivel a türkiz, bíbor és sárga kivonja a komplementer vöröset, zöldet és kéket és ennek eredménye fekete, ezért lehet fehéret előállítani a CMY alapszínekkel,csak feketét. Az 5.2-es árbát az internetről, a http://www.epab.bme.hu/oktatas/Jegyzetek/visualization/15_CMY.pdf oldalról szedtem [6], [7].



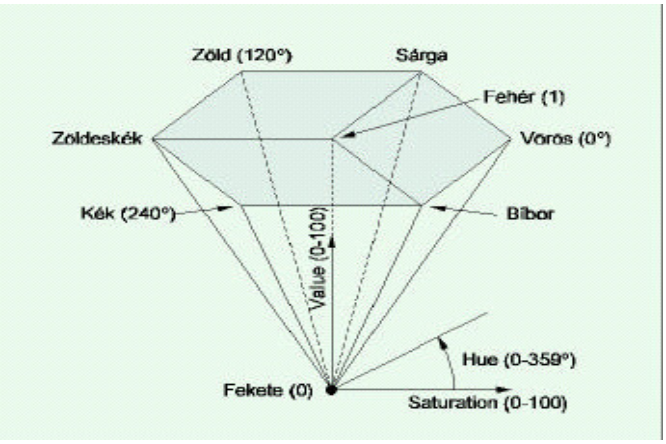
5.2-es ábra. CMY színrendszer

CMYK:

A CMY színrendszer kiegészül a fehérrel és további köztes árnyalatokkal és a feketével, mivel csak elviekben lesz fekete a CMY 3 alapszíne, a gyakorlatban sötétbarna keletkezik. Az elnevezésben a feketét K jelöli. K, mint Key, azaz kulcsszín. Egy színkomponenst 1 byteon ábrázol és 0-tól 255-ig vehet fel értéket. Mivel 4 színkomponens van, és ezeket összesen 4 byteon ábrázolja, tehát 32 biten, ezért 256*256*256*256, azaz körülbelül 4,3 milliárd árnyalatot képes ábrázolni.

HSV:

A színek értékeit színárnyalatból (H (Hue)), telítettségből (S (Saturation)) és világosságból (V (Value)) állítja elő. Ha S = 0 és V = 1, akkor fehéret kapunk, H itt meghatározatlan (undefined). Feketét akkor kapunk, ha V = 0, ekkor az S és a H is meghatározatlan. A következő ábra (5.3-as) a HSV színrendszert mutatja. A következő ábra (5.3-as) a http://www.epab.bme.hu/oktatas/Jegyzetek/visualization/14_RGB.pdf oldalról van.

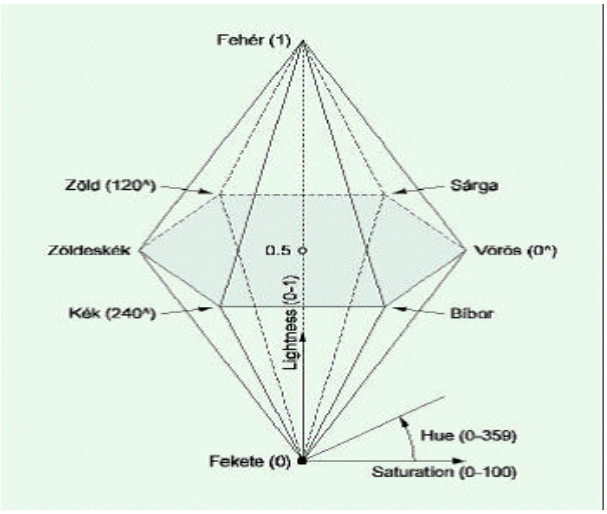


5.3-as ábra. HSV színrendszer

A gúla 1 egység magas és 1 egység sugarú a tetején.

HLS:

Színárnyalat (H (Hue)), világosság (L (Lightness)) és a telítettség (S (Saturation)) hármas határozza meg a színt. Abban különbözik a HVS színrendszertől, hogy a V = 1 síkból a fehér pont kiemelkedik, ezzel egy dupla hatszögletű gúlát hozva létre (5.4-es ábra). Az 5.4-es ábra a http://www.epab.bme.hu/oktatas/Jegyzetek/visualization/14_RGB.pdf oldalról van.



5.4-es ábra. HLS színrendszer

Black and white:

Egy képpont színe vagy fekete vagy fehér, tehát 2 lehetséges állapota van. Egy színárnyalat ábrázolása megoldható 1 biten.

16 Color:

Egy képpont színe ábrázolásának 16 lehetséges állapota lehet, tehát 4 bit szükséges egy árnyalat ábrázolásához.

Grayscale:

Egy képpont színe a szürke 256 féle árnyalatai közül lehetséges, tehát ábrázolásához összesen 8 bit, azaz 1 byte elegendő.

256 Color:

Egy képpont színére vonatkozó információt 8 biten tárolja, lehetséges színe pedig 256 szín közül lehetséges.

6. Java grafika

A következőkben a Java grafikájának egy részét fogja taglalni a szakdolgozatom, melyet a Java 2 Útikalauz programozóknak: 5.0 I. és II. könyvekből vettem [9], [10].

6.1. Grafikus felhasználói felület

A GUI (Graphical User Interface) lehetőséget biztosít a grafikus felhasználói felület használatára, tehát adatok bevitelére, ezekkel számítások elvégzésére és ezek eredményeinek kijelzésére.

6.2. Grafikus komponensek

Grafikus komponenseknek azokat, a rendszerint tovább nem bontható elemeket nevezzük, amelyek kiadják a grafikus felületet összességükben. 3 csoportba sorolhatjuk ezeket a feladatuk alapján: beviteli komponensek, vezérlő komponensek, valamit megjelenítő komponensek. A megjelenítő komponensek a felhasználó felé információt jelenítenek meg, a vezérlő komponensek segítségével a program futását lehet szabályozni, illetve a beviteli komponensekkel adatokat lehet átadni a program számára.

6.3. JFC

A JFC (Java Foundation Classes – Alapvető Java Osztályok) programkönyvtár-gyűjteményét és az alábbi alkotó részeit a Sun hozta létre.

AWT (Abstract Window Toolkit – Absztrakt Ablakozó Eszköztár)

A felületelemek megjelenítéséért nem a Java platform a felelős, hanem az operációs rendszer grafikus ablakozó rendszere, ahogyan erre utal az összetevő nevében is az Abstract szó. Ennek hátulütője viszont, hogy a rendelkezésre álló grafikus komponensek halmaza, az összes aktuális ablakozó rendszer által preferált elég kicsi közös részére tevődik, valamint a komponensek valódi megjelenése az operációs rendszertől függ, ami élesen szemben áll a Java platformfüggetlenségi elvével. Előnye is van viszont, ami az, hogy a Java felhasználói felületek könnyen beilleszkednek a szokásos ablakozó rendszer által létrehozott stílusú munkafelületbe, valamint biztosítja az adatátvitel lehetőségét a különböző ablakok tartalma közti áthúzással. A java.awt csomaghierarchia tartalmazza ezt, a grafikus felhasználói felületek létrehozásához szükséges alap API-t.

Swing

Nevét a JFC grafikus összetevőit fejlesztő munkaprogram kódnevéről kapta. A Swing igen fejlett grafikus programkönyvtár, melyet azért hoztak létre, hogy megszüntessék az AWT hibáit.

2D API

Kétdimenziós grafikai műveletek, illetve képfeldolgozást ismertető és grafikák nyomtatását lehetővé tevő API.

Segítő technikák

Segít mindazon felhasználóknak, akiknek komplikációt okoz a normál grafikus felhasználói felületek használata, lehetővé téve például szövegolvasó berendezések vagy vakírást ismerő billentyűzetek használatát, erre szolgál ez az API.

Lokalizálás

A felhasználó által használt nyelvhez hozzáidomítható a felhasználói felület nyelve, valamint összetett írásrendszerek használata is lehetséges.

6.4. A felhasználói felület életciklusa

A felhasználói felület objektumokból rakódik össze, ahol a grafikus komponenseket egy-egy objektumpéldány helyettesít, mint ez az objektumorientált nyelveknél lenni is szokott. A felhasználói felület életciklusa tehát a következő három szakaszra osztható:

Felület felépítése

Ezen idő alatt kell létrehozni a megfelelő objektumokat, majd beállítani ezek jellemzőit és az egymás között levő kapcsolatokat.

Felület használata

Ekkor tud a felhasználó a programmal kommunikálni a felület segítségével, így ez a felhasználó számára legfontosabb szakasz az életcikluson belül.

Felület bezárása

A felületet felépítő objektumok által lefoglalt erőforrások felszabadítása történik a felhasználói felület életciklusának ezen szakaszában, az előző szakasz feladata, a kommunikáció lezárása után.

6.5. Swing

A Swing független a felhasználói felületet megjelenítő operációs– és ablakozó rendszertől, mivel teljes egésze Javában íródott. A Swing alkotórészei az AWT alkotórészei továbbjavításának tekinthetők. A Swing típusokat a javax.swing csomagok foglalják magukban.

Egyik nagy újítása abban rejlik, hogy a komponensek az adatmodell vezérlés (MVC –Model-View-Controller) architektúrát használják. Ennek következtében logikailag három, egymástól teljesen szétválasztható funkcionális részt lehet elkülöníteni:

Adatmodell: A komponens által megjelenített adatok.

Megjelenítés: A komponens adatmodellje által tartalmazott adatok grafikus megjelenítéséért felelős.

Vezérlés: A felhasználói eseményeket felhasználó logika.

6.5.1. Megjelenítés

A Swing komponensek az MVC egy egyszerűsített típusát alkalmazzák, ahol csak az adatmodell és a grafikus komponensek vannak elkülönítve egymástól: minden grafikus komponens saját maga felel a megjelenítésért és azért, hogy feldolgozzák a felhasználói eseményeket, így a komponens a modell és a grafikus megjelenítés közti kommunikációt saját maga vezérli.

A Swing komponensei a java.awt.Component leszármazottai, tehát az AWT pehelysúlyú komponens lehetőségét használják, ezért a megjelenítésért teljes mértékben saját maguk a felelősek. Emiatt a natív nehézsúlyú AWT komponensek és a Swing komponensek keverése problémákat eredményez, pontosabban a nehézsúlyú komponensek pehelysúlyú komponenseken belül történő megjelenítése váltja ki a hibákat. A probléma oka az, hogy a natív komponensek mindig teljesen látszanak, azoknak legfeljebb a megjelenítési területét (ami általában kisebb, mint a tényleges terület, melyet a komponens foglal el)takarhatja pehelysúlyú komponens. Mindig egy, az ablakozó rendszer által biztosított nehézsúlyú rajzolási felületen belül történik a pehelysúlyú komponensek megjelenítése, ennek következtében míg a pehelysúlyú komponensek takarási sorrendje helyesen megjeleníthető,

addig a pehely- és nehézsúlyú komponensek közti takarási (vagy mélységi) sorrend nem szabályozható.

Lehet átlátszó egy Swing komponens, ezt a setOpaque metódus paraméterével lehet beállítani. Ha hamis az érték, akkor a komponens átlátszó, igaz érték esetén pedig a komponens mindig törli az általa foglalt képernyőterületet a megjelenítésének háttérszínével.

A Swing komponensek megjelenítése az AWT grafikus eshetőségeinek alkalmazásával, annak egyetlen programszálában, az AWT végrehajtó programszálban (AWT dispatch) valósul meg. A megjelenítés mindig a komponenshierarchia legfelső, megjelenítést igénylő eleménél kezdődik, és ezen részhierarchián halad a mélyebben található komponensek felé. Egy komponens ábrázolásának a következőképpen megy végbe:

1. Ha nem átlátszó a komponense, akkor megjelenik annak háttere.
2. A paintComponent metódus meghívásával a komponens megrajzolta saját magát. Ez az egyetlen megjelenítési művelet, melyet a Swing grafikus komponense végez el, nem saját maga.
3. Ha rendelkezik a komponens kerettel (vagy például nála van a beviteli fókusz), akkor annak megjelenítése következik a paintBorder metódussal.
4. Végül ha a komponens további komponenseket tartalmaz, akkor azok fognak sorban megjelenni a paintChildren metódus hatására.

A takarási sorrend helyes megjelenítését ezen sorrend betartása garantálja. Saját komponens megjelenítését a fentebb ismertetett metódusok valamelyikének (általában a paintComponent) felüldefiniálásával valósítsuk meg és ne a paint metódussal.

A Swing komponensek automatikusan vezérlik a saját megjelenítésüket, tehát valamely jellemzőjük vagy az adatmodell megváltozásakor gondoskodnak saját megjelenítésükről. Kérvényezhető újramegjelenítés, mely a repaint metódussal is megvalósítható. Ha a komponens mérete vagy pozíciója megváltozott, akkor célszerű a komponens revalidate metódusát is meghívni még a módosulás előtt, hogy ezt a megjelenítési stratégia helyesen vegye figyelembe.

Minden Swing komponens megjelenítése offscreen technikával történik az alapértelmezés szerint, azaz a Swing először a memóriában felépíti a komponens képét, majd azt a képernyőn

egyben jelenítni meg. Ennek a módszernek a használatát a komponense isDoubleBuffered metódusával lehet lekérdezni, beállítani pedig a setDoubleBuffered metódussal lehet. Ugyancsak a megjelenítés optimalizálását szolgálja a RepaintManager osztály, amely a megjelenítendő komponenseket tartja nyilván, illetve az összegyűjtött információk alapján azok újrarajzolását és offscreen puffer foglalását (offscreen puffer: a komponens képének memóriában történő felépítéséhez szükséges tárterület) optimalizálja. A RepaintManager objektum, amely az adott komponens megjelenítését végzi, ezen osztály currentManager statikus metódusával lehet elérni, a pufferelt megjelenítés használatát pedig annak set/isDoubleBufferingEnablad metódusával lehet beállítanii/lekérdezni. Hozzáfűzendő, hogy a pufferelt megjelenítés kikapcsolása nagymértékben megnöveli a komponensek megjelenítéséhez szükséges időt.

A megjelenítést vezérlő programszál:

Az MVC (Model – View – Controller) architektúra alkalmazásának következménye, hogy a komponens megjelenítése, a modellje tartalma alapján történik. Ha a modell adatai megváltoznak a megjelenítés alatt, akkor a megjelenítés kinézete hibás lehet. Ennek következtében mindenképpen biztosítani kell, hogy a modell változása és a megjelenítés végrehajtása egymáshoz képest szinkronizálva fordulhasson csak elő. Hatékonysági okokból nem építették be a Swingbe ezen szinkronizációt, mivel túlságosan nehézkessé tenné az API megvalósítását a modell- és komponensjellemzők zárolási mechanizmusa vagy a megjelenítendő adatok esetleges másolása. Emiatt a szükséges szinkronizáció implementálása mindig a Swing API használójának a feladata. Ezért azt mondhatjuk, hogy többszálú programkörnyezetben a Swing önmagában nem használható, mindig gondoskodni kell a megfelelő szinkronizációról.

Ennek támogatására a Swingben a felhasználói eseményeket csak egyetlen programszál (az AWT programszála) fogadja és továbbítja az esetleges eseménykezelők felé. Ennek eredményeképpen az eseménykezelő kódok csakis szinkronizálva, egymás után kerülnek meghívásra. Ugyanez a programszál vezérli a komponensek megjelenítését is, mint ahogy azt már említettük, ezért ezen egyetlen programszál használata szinkronizálja a megjelenítést, valamint az eseménykezelő kódok végrehajtása, ahol az adatmodell változtatása rendszerint történik. Ügyeljünk arra, ezt figyelembe véve, hogy gyorsan befejeződjön minden eseménykezelő, mivel blokkolódik a teljes felhasználói felület annak futása alatt. Ha hosszabb

teendőt kell elvégezni, hogy ne blokkoljuk feleslegesen a felhasználói felület megjelenítését és esemény-feldolgozását, az eseménykezelőben indítsunk el egy saját programszálat, amely elvégzi a feladatot. Ehhez persze mindenképp vegyük figyelembe, hogy az újonnan létrehozott programszál örökli az aktuális végrehajtási prioritást (azaz a végrehajtó programszál magas prioritását!), ezért a start metódusával történő indítása előtt mindenképp csökkentsük le azt legalább normális szintre (setPriority(Thread.NORM_PRIORITY)).

Az előbbieket egy szabályba össze lehet foglalni, amely a következő: miután létrejött egy Swing komponens, minden programkód, ami ezen komponens megjelenítését változtatja, illetve annak állapotát lekérdezi vagy módosítja, csakis az AWT végrehajtó programszálában futtatható. A következő eseteket értünk a komponens létrejötte alatt:

- ha a komponens készen áll a megjelenítésre vagy már megjelent a képernyőn
- ha meghívjuk a komponens show (ablakoknál pack) vagy setVisible metódusát
- ha létrejön a komponens valamely őse
- vagy ha egy, már korábban létrejött konténerbe vesszük fel.

Csak a következő estek jelentenek kivételt ezen szabály alól:

- A main metódusában felépítheti majd megjelenítheti egy alkalmazás a grafikus felhasználói felületet annak pack, majd közvetlenül utána a setVisible metódusának meghívásával. A main metódus ezután viszont nem változhat a grafikus felhasználói felület felépítésében, illetve nem férhet hozzá a felületelemek tulajdonságaihoz sem.
- Egy applet, annak init metódusában felépítheti a grafikus felhasználói felületet, a megjelenítést azonban mindig a böngészőprogram végzi el.
- A repaint és revalidate metódusok, melyek a komponens újrafestését kérik, bármely programszálból hívhatóak, ugyanis az újrarajzolást vezérlő RepaintManager osztály gondoskodik róla, hogy minden csakis az AWT végrehajtó programszálban történik.
- Komponens eseményfigyelőinek listáját akármelyik programszálból lehet befolyásolni.

Ha a fenti szabályokat nem tartjuk be, akkor csak a megjelenítés zavarodik össze jobb esetben, rosszabb esetben pedig akár holtponthoz is vezethet a komponensek több programszálból egyszerre történő elérése/manipulálása.

A SwingUtilities osztály a következő statikus metódusokat biztosítja az AWT végrehajtó programszál helyes használatához:

- is EventDispatchThread: ha az aktuálisan futó programszál az AWT végrehajtó szála, akkor igazat ad vissza. Olyan kódrészleteknél lehet szükséges ennek lekérdezése, melyeket az AWT fő szálán kívül több más programszál is végrehajthat.
- invokeLater: Runnable objektumot, melyet paraméterként kapott meg, úgy ütemezi (FIFO), hogy az AWT végrehajtó programszál hajtsa végre annak run metódusát. A metódus egyből visszatér, miután minden egyéb teendőjét befejezte az AWT végrehajtó szál pedig csak azután hajtja végre a kapott objektum run metódusát. A legújabb ajánlások szerint alkalmazások esetén célszerű már a GUI felépítését is ezen metódussal ütemezni, a következő séma szerint:

  ```
  private static void felépít() {
  …                                    //grafikus felület felépítése és megjelenítése
  }
  public static void main(String argumentumok[]) { //alkalmazás indítása
  …                                                //inicializálások elvégzése
          javax.swing.SwingUtilities.invokeLater(new Runnable(){ //ütemezés
                  public void run() {felépít();}
          });
  }
  ```

  Ennek a megoldásnak az a lényege, hogy biztosan az esemény feldolgozó programszálban történjen a grafikus felület első megjelenítése, hiszen ekkor már esetleges komponenshierarchia eseményfigyelők végrehajtásra kerülnek!
- invokeAndWait: addig vár, amíg a paraméterként magadott Runnable objektum run metódusát az AWT végrehajtó programszál végre nem hajtotta, ezért nem szabad meghívni az AWT végrehajtó szálban, különben kivétel fog fellépni. Már az appletek esetén is érdemes a GUI felépítését ezzel a metódussal ütemeztetni:

  ```
  private void felépít() {
  …                                    //grafikus felület felépítése
  }
  public void init() {                 //applet inicializálása
  …                                    //inicializálások elvégzése
  ```

```
        if(javax.swing.SwingUtilities.isEventDispatchThread()) felépít();
        else javax.swing.SwingUtilities.invokeAndWait(new Runnable() {
            public void run() {felépít();}
        });
    }
```

Ennek a megoldásnak az a lényege, hogy mivel rendszerint nem maga az AWT végrehajtó szál az appletet inicializáló programszál, viszont befejeztekor már megjelenítésre készen kell állni az applet grafikus felületének, ezért ajánlott a szinkronizált várakoztatás.

Az AWT eseménysorának (EventQueue) azonos nevű metódusait hívják meg ezek a metódusok, tehát most már csak kompatibilitási okok miatt maradtak meg a Swing osztályban is.

6.5.2. A megjelenítés stílusának megváltoztatása

Az MVC architektúra egy módosított változatát használják a Swing komponensek, ahol a megjelenítésért különálló grafikus komponensek a felelősek. Ennek nagy előnye, hogy külön szabályozható minden komponens megjelenítése, sőt a megjelenítés akár futás közben is megváltoztatható, mivel az adatmodellben tárolt adatok függetlenek azok megjelenítésétől. Ennek következtében lehetőségünk van arra, hogy egysezrre megváltoztassuk minden komponens megjelenését valamilyen közös esztétikai szempontoknak megfelelően, ezt a megjelenítés stílusának nevezzük (LAF – Look-And-Feel).

A javax.swing.plaf csomagban és annak alcsomagjaiban találjuk a megjelenítés stílusának kezeléséhez szükséges típusokat. A ComponentUI az ősosztálya a megjelenítést végző komponenseknek. Tehát futási időben is könnyedén elvégezhető a megjelenítési stílus váltása, ezért is szokás a megjelenítési stílusra a lecserélhető jelzőt alkalmazni (PLAF – Pluggable Look-And-Fell), innen ered az említett alcsomag neve is.

Az absztrakt LookAndFeel reprezentálja a megjelentés stílusát, ami tulajdonképp csak a szükséges metódusokat definiálja. A javax.swing.plaf.basic csomagban található a leszármazottja, a szintén absztrakt BasicLookAndFeel már ad implementációt minden Swing komponens megjelenítéséhez. Ennek az osztálynak a következő három leszármazottai reprezentálják a JDK-val együtt adott megjelenítési stílusokat:

- MetalLookAndFeel: Java Look-And-Feel-nek is nevezik (JLF), mert a Swing komponensek platform-független megjelenési formáját biztosítja. Ez az alapértelmezett megjelenítési stílus. Megvalósítását a javax.swing.plaf.metal csomag adja.
- GTKLookAndFeel: a GTK+ megjelenítési formáját teszi lehetővé a Swing komponenseknek. A grafikus komponensek implementációjára a com.sun.java.swing.plaf.gtk csomagban bukkanhatunk. Ezen megjelenítési stílus használatának előfeltétele a legalább 2.2-es verziójú GTK+, ilyenkor Linux és Solaris operációs rendszereken az a platformfüggő megjelenítési stílus.
- MotifLookAndFeel: a Motifos megjelenítési formáját teszi lehetővé a Swing komponenseknek. A com.sun.java.swing.plaf.motif csomagban találjuk a grafikus komponensek implementációját. Ez a platformfüggő megjelenítési stílus Linux és Solaris operációs rendszereken, ahol nincs (megfelelő) GTK telepítve.
- WindowsLookAndFeel: a Windows megjelenítési formáját teszi lehetővé a Swing komponenseknek. A com.sun.java.swing.plaf.windows csomagban található a megvalósítása. Csak a Windows operációs rendszereken használható ezen megjelenítési stílus.

Tehát csak az alapértelmezett Metal, valamint a Motif tekinthető valóban platformfüggetlen (és mindig elérhető) stílusnak.

Érdekes lehetőségeket kínál fejlesztői szempontból a javax.swing.plaf.multi csomagban található stílus, amely lehetővé teszi stílusok keverését (segítségével például felolvasási lehetőséget biztosíthatunk), valamint a javax.swing.plaf.synth csomag, ahol a megjelenítést teljes egészében magunk szabályozhatjuk teljes egészében akár egy XML leírófájl felhasználásával. Az aktuális megjelenítés stílusát az UIManager osztály tartja nyilván.

6.5.3. Felsőszintű Swing konténerek

A grafikus felhasználói felület ablakozó rendszerhez történő kapcsolódását a felsőszintű konténerek valósítják meg.

Azon Swing konténereket nevezzük felsőszintű Swing konténereknek, amelyek még nem pehelysúlyú komponensek., viszont már tartalmazhatnak Swing komponenseket. Pontosabban csak egyetlen gyermeket tartalmaz minden felsőszintű Swing konténer, amely gyermek

mindige egy getRootPane objektum, amit automatikusan létrehoz és beállít a konstruktor. A Swing komponensek további hierarciáját ezen objektum tartalompanelje fogja tartalmazni

Párbeszédablakok

A JDialog osztály reprezentálja a párbeszéd- vagy dialógusablakokat. Ez az osztály a java.awt.Dialog leszármazottja és azt tulajdonképpen csak azon metódusokkal egészíti ki, amelyek a RootPaneContainer interfészt valósítják meg. Egy adott komponensen beül a párbeszédablak megjelenítését a setLocationRelativeTo metódussal lehet középre igazítani. Azt, hogy a dialógus a becsukásakor hogy viselkedjen, a get/setDefaultCloseOperation metódusokkal lehet szabályozni, az alapértelmezett a HIDE_ON_CLOSE viselkedési mód.

Beviteli dialógusdoboz

Dialógusdobozok létrehozása legegyszerűbben a JOptionPane osztály segítségével történhet. Az így létrehozott dialógusdobozt vagy egy JDialog vagy egy JInternalFrame jelenítheti meg, ami a következő komponenseket tartalmazza:

- A dialógusdoboz bal felső sarkában elhelyezkedő ikon, amely az üzenet típusát reprezentálja. A get/seticon metódusokkal lehet elérni az ikont.
- Egy Object által reprezentált üzenet, amely az ikon melletti területen olvasható. Akkor fog egyértelműen megjelenni, ha ez egy grafikus komponens, ha egy ikon, akkor az azt tartalmazó címke fog megjelenni, egyéb típus esetén pedig a megjelenítendőüzenetet az objektum szöveges reprezentációja fogja jelenteni. A get/setMessage metódusokkal lehet kezelni az üzenetet. A get/setMessageType metódusokkal lehet lekérdezni/megadni az üzenet típusát, ahol a következő konstansok használhatók:
    - ERROR_MESSAGE (hibakijelző dialógusdoboz)
    - INFORMATION_MESSAGE (információt kijelző dialógusdoboz)
    - WARNING_MESSAGE (figyelmeztető dialógusdoboz)
    - QUESTION_MESSAGE (kérdő dialógusdoboz)
    - PLAIN_MESSAGE (tetszőleges üzenetet kijelző dialógusdoboz)
- Az üzenet szövege alatt különböző kiválasztható opciók jelennek meg a beviteli dialógusdobozok esetén. A get/setWantsInput metódussal lehet lekérdezni/szabályozni ezen opciók láthatóságát. A beviteli opciók vizuális reprezentációja rendszerint egy legördülő lista, vagy túl sok elem esetén egy gördíthető lista. Egy objektumtömb adja

meg a kiválasztható opciókat, amely akár null is lehet, ami azt jelzi, hogy tetszőleges adatot megadhat a felhasználó rendszerint egy beviteli szövegmező segítségével. A get/setSelectionValues metódussal lehet lekérdezni/megadni a kiválasztható opciókat. Az alapértelmezett módon kiválasztott elemet pedig a get/setSelectionValue metódussal lehet megtudni/beállítani.

- Legalul pedig olyan nyomógombok jelennek meg, amik különböző lehetőségek kiválasztását teszik lehetővé. Egy objektumtömb reprezentálja ezen opciókat, melynek adott eleme
    - ha egy grafikus komponens, akkor az egyszerűen meg fog jelenni,
    - ha egy ikon, akkor az a nyomógomb fog megjelenni, amely azt tartalmazza,
    - egyéb típus esetén pedig az objektum szöveges reprezentációját, mint feliratot használó nyomógomb lesz látható.

A set/getOptions metódusokkal lehet lekérdezni/beállítani az opciókat. Az opció típusának megadásával lehet megjeleníteni opciók előredefiniált csoportját. A get/setOptionType metódusokkal lehet ezen típust lekérdezni/megadni, ahol a következő konstansok használhatók: public static final int

- DEFAULT_OPTION: egy OK gomb jelenik meg
- YES_NO_OPTION: egy Igen és egy Nem gomb jelenik meg
- YES_NO_CANCEL_OPTION: Igen, Nem és egy Mégsem gomb jelenik meg
- OK_CANCEL_OPTION: egy Igen és egy Mégsem gomb jelenik meg

A get/setInitialValue metódussal lehet megtudni/beállítani az alapértelmezett módon kiválasztott opciót. Az így beállított alapértelmezett opció kiválasztását a selectInitialValue metódussal lehet elvégezni. A get/setValue metódussal pedig a felhasználó által ténylegesen kiválasztott opciót lehet lekérdezni/megadni, melynek null az eredménye, ha a felhasználó nem választott ki semmit és becsukta a dialógusablakot, ha még nem történt semmi kiválasztás, akkor UNINITIALIZED_VALUE, egyébként pedig a kiválasztott objektum lesz. Jóváhagyást kérő dialógusdobozok visszatérési értéke egy egész szám lesz, melynek értékét a következő konstansok adják meg:

- YES_OPTION: a felhasználó az Igen gombot nyomta meg
- NO_OPTION: a felhasználó a Nem gombot nyomta meg

- CANCEL_OPTION: a felhasználó a Mégsem gombot nyomta meg
- OK_OPTION: a felhasználó az OK gombot nyomta meg
- CLOSED_OPTION: a felhasználó gombnyomás helyett bezárta az ablakot

Mindig meg kell adni egy szülőkonténert is a megjelenítéshez,(ennek hiányában egy alapértelmezett JFrame objektumot használ, amelyet kezelni a get/setRootFrame statikus metódusokkal lehet). A szülőkomponensben középre igazítva, modálisan fog megjelenni JDialog esetén, míg JInternalFrame használatakor az a szülő (vagy az azt tartalmazó) többrétegű panel MODAL_LAYER rétegében fog nem modálisan megjelenni.

A dialógusdobozt vagy mi példányosítjuk, konfiguráljuk és tesszük ki a képernyőre, vagy a JOptionPane következő statikus metódusait használjuk

- show[Internal]ConfirmDialog: egy jóváhagyó dialógusablakot jelenít meg. A szülőkonténer az első paramétere, második pedig a megjelenítendő üzenet. Az összes többi paraméter opcionális. A dialógusdoboz fejlécében megjelenítendő szöveg (alapértelmezett értéke: „Válasszon egy opciót") a harmadik paraméter, negyedik paraméter az opciók típusa (alapértelmezett értéke: YES_NO_CANCEL_OPTION), az üzenet típusa (alapértelmezett értéke: QUESTION_MESSAGE) az ötödik, végül az utolsó paramétere a megjelenítendő ikont adja meg. Visszatérési értéke egy egész szám lesz. Ez az érték a felhasználó által megnyomott gombot jelölő egész szám lesz.
- show[Internal]InputDialog: egy beviteli párbeszédablakot jelenít meg. Első paramétere a szülőkonténer, a megjelenítendő üzenet pedig a második. Az összes többi paraméter opcionális. Harmadik paramétere a megjelenítendő dialógusdoboz fejlécében megjelenítendő szöveg (alapértelmezett értéke: „Bevitel"), az üzenet típusa (alapértelmezett értéke: QUESTION_MESSAGE) a negyedik paraméter, ötödik paraméter a megjelenítendő ikon, a megadható opciók tömbje (alapértelmezett értéke null) a hatodik paramétere, utolsó paramétere pedig a kezdetben kiválasztott opciót adja meg. Visszatérési értéke a felhasználó által beírt String vagy a kiválasztott objektum, illetve null, ha nem történt bevitel.
- show[Internal]MessageDialog: üzenet-dialógusablakot jelenít meg egy OK gombbal. Első paramétere a szülőkonténer, a megjelenítendő üzenet pedig a második. Az összes többi paraméter opcionális. Harmadik paraméter a dialógusdoboz fejlécében megjelentendő szöveg (alapértelmezett értéke: „Üzenet"), az üzenet típusa

(alapértelmezett értéke: INFORMATION_MESSAGE) pedig a negyedik, végül az utolsó paramétere a megjelenítendő ikont adja meg. Visszatérési értéke nincs.

- showOptionDialog: általános kiválasztó dialógusablakot jelenít meg. Első paramétere a szülő konténer, második paraméter a megjelenítendő üzenet, harmadik paraméter a dialógusdoboz fejlécében megjelenítendő szöveg, negyedik paraméter az opciók típusa, ötödik paraméter az üzenet típusa, hatodik paramétere a megjelenítendő ikon, hetedik paraméter a kiválasztható opciót adja meg. A felhasználó által megnyomott gombot jelölő egész szám lesz a visszatérési értéke.

Ezen metódusok az általuk létrehozott JDialog vagy JInternalFrame dialógusablakokat egyből megjelenítik, és a felhasználó válaszát fogják visszaadni visszatérési értékként. A konstruktorban a következőket adhatjuk meg ha a dialógusdobozt mi magunk példányosítjuk: az első paraméter a megjelenítendő üzenet, második paraméter az üzenet típusa, az opció típusa a harmadik paraméter, negyedik paraméter a megjelenítendő ikon, a kiválasztható opciók tömbje az ötödik paraméter, utolsó paramétere pedig a kezdetben kiválasztott opciót adja meg. Még nem jelenik meg a dialógusdoboz a konstruktor végrehajtásakor, tehát lehetőségünk van annak további konfigurálására.A createDialog/createInternalFrame metódusok hozzák létre a megjelenítéshez szükséges JDialog/JInternalFrame objektumokat. Első paraméternek a szülőkonténert, második paraméternek pedig a show metódusának meghívásakor lehet a képernyőn is megjeleníteni. Ezután a getValue és getInputValue metódussal lehet lekérdezni a felhasználó válaszát.

Beviteli dialógusdobozok megjelentését az UIDefaults ezen jellemzői szabályozzák:

- OptionPane.background: háttérszínt megadó java.awt.Color
- OptionPane.border: keretet megadó Broder
- OptionPane.buttonAreaBorder: a gombokat tartalmazó rész keretét megadó Border
- OptionPane.errorIcon: hibát jelölő ikont megadó Icon
- OptionPane.font: betűtípust megadó java.awt.Font
- OptionPane.foreground: előtérszínt megadó java.awt.Color
- OptionPane.informationIcon: információt jelölő ikont megadó Icon
- OptionPane.messageAreaBorder: a üzenetet tartalmazó rész keretét megadó Border
- OptionPane.messageForeground: az üzenet előtérsínét megadó java.awt.Color

- OptionPane.minimumSize: minimális méretet megadó java.awt.Dimension
- OptionPane.questionIcon: kérdést jelölő ikont megadó Icon
- OptionPane.warningIcon: figyelmeztetést jelölő ikont megadó Icon

Színkiválasztás:

A JColorChooser osztály használható a színkiválasztásra. A javax.swing.colorChooser alcsomag tartalmazza a színkiválasztó dialógusdoboz létrehozásához szükséges egyéb típusokat. A showDialog statikus metódussal lehet megjeleníteni a dialógusablakot, melynek visszatérési értéke a kiválasztott színt reprezentáló java.awt.Color objektum vagy null lesz, ha a felhasználó nem választott ki színt és csukta be a dialógusablakot.

Az alapértelmezés szerinti színkiválasztó dialógusablak háromféle módon is lehetővé teszi a színkiválasztást:

- palettából való kiválasztás
- kiválasztás vagy megadás a HSB színrendszer alapján
- kikeverés az RGB színrendszer alapján

Színkiválasztó dialógusdobozok megjelenítését az UIDefaults e jellemzói szabályozzák:

- ColorChooser.background: háttérszínt megadó java.awt.Color
- ColorChooser.font: betűtípust megadó java.awt.Font
- ColorChooser.foreground: előtérszínt megadó java.awt.Color
- ColorChooser.selectedColorBorder: kiválasztott szín keretét megadó Border

Fájlkiválasztás:

A JFileChooser osztály használható fájlkiválasztásra. A javax.swing,filechooser alcsomag tartalmazza a fájlkiválasztó dialógusdoboz létrehozásához szükséges egyéb típusokat. A dialógusablakot a showDialog, fájlmegnyitáskor a showOpenDialog, mentéskor pedig a showSaveDialog metódusokkal lehet megjeleníteni, melyek visszatérési értéke:

- CANCEL_OPTION: a felhasználó megszakította a fájlkiválasztást
- APPROVE_OPTION: a felhasználó elvégezte a fájlkiválasztást.

Az is/setMultiSelectionEnabled metódusokkal szabályozható egyszerre több fájl kiválasztása. Ha a felhasználó kiválasztott egy fájlt, akkor a getSelectedFile, többszörös kiválasztást pedig a getSelectedFiles metódussal lehet lekérdezni.

Fájlkiválasztó dialógusdobozok megjelenítését az UIDefaults e jellemzői szabályozzák:

- FileChooser.acceptAllFileText: fájlnevek szűrését megadó String
- FileChooser.cancelButtonText: Mégsem gomb felirítát megadó String
- FileChooser.cancelButtonToolTipText: Mégsem gomb segédszövegét megadó String
- FileChooser.detailsViewIcon: részletes nézetre váltó gomb ikonját megadó Icon
- FileChooser.helpButtonText: Súgó gomb feliratát megadó String
- FileChooser.helpButtonToolTipText: Súgó gomb segédszövegét megadó String
- FileChooser.homeFolderIcon: kezdeti könyvtárra léptető gomb ikonját megadó Icon
- FileChooser.listViewIcon: lista nézetre váltó gomb ikonját megadó Icon
- FileChooser.newFolderIcon: új könyvtárat létrehozó gomb ikonját megadó Icon
- FileChooser.openButtonText: Megnyitás gomb felirítát megadó String
- FileChooser.openButtonToolTipText: Megnyitás gomb segédszövegét megadó String
- FileChooser.saveButtonText: Elmentés gomb feliratát megadó String
- FileChooser.saveButtonToolTipText: Elmentés gomb segédszövegét megadó String
- FileChooser.upFolderIcon: szülőkönyvtárba léptető gomb ikonját megadó Icon
- FileChooser.updateButtonText: Frissítés gomb feliratát megadó String
- FileChooser.updateButtonToolTioText: Frissítés gomb segédszövegét megadó String

A javax.swing.filechooser.FileView absztrakt osztály a fájlok grafikus reprezentálásához nyújt segítséget, melyet megvalósító objektumot a get/setFileView metódussal lehet lekérdezni/beállítani. Ezen osztály a megjelenítéséhez a következő információkat nyújtja:

- A getDescription metódus ad szöveges leírást adott fájlhoz
- A getIcon metódus adja meg az adott fájlt reprezentáló ikont
- A getName metódus kérdezi le adott fájl nevét
- Adott fájl típusáról a getTypeDescription metódus ad szöveges leírást
- Az isTraversable metódus megadja, hogy bele lehet-e lépni adott könyvtárba

Fájlok megjelenítését az UIDefaults következő jellemzői szabályozzák:

- FileView.computerIcon: számítógépet reprezentáló ikont megadó Icon
- FileView.directoryIcon: könyvtárat reprezentáló ikont megadó Icon
- FileView.fileIcon: fájlt reprezentáló ikont megadó Icon
- FileView.floppyDriveIcon: hajlékonylemezes meghajtót reprezentáló Icon
- FileView.hardDriveIcon: merevlemezes meghajtót reprezentáló ikont megadó Icon.

6.5.4. Konténerek

Panel:

A JPanel a legegyszerűbb teljes értékű Swing konténer. Funkcionalitását teljes egészében a JComponent-től örökli. A java.awt.FlowLayout az alapértelmezett elrendezési stratégiája, amit akár már a konstruktorban is meg lehet változtatni.

Panelek megjelenítését az UIDefaults következő jellemzői szabályozzák:

- Panel.background: háttérszínt megadó java.awt.Color
- Panel.font: betűtípust megadó java.awt.Font
- Panel.foreground: előtérszínt megadó java.awt.Color

JInternalFrame:

Belső ablakok használatát a JInternalFrameTeszt példaprogramunkkal szemléltetjük. A felhasználói felület felső része tartalmazza a belső ablakokat, alul pedig az új ablakot létrehozó gomb melletti nyomógombokkal a megjelenítés stílusát lehet megváltoztatni. A belső ablakokat az Ablak belső osztályunk valósítja meg, melynek minden példányát sorszámmal jelöljük. Konstruktorának paramétere jelzi, hogy melyik rétegben fog elhelyezkedni az ablak, azt a bal alsó sarokban található szövegmezőben lehet megadni az ablak létrehozásakor. Egy belső ablak tetején az ablakkal végzett műveletek naplója látható, alatta pedig az ablak tulajdonságait és műveleteit kiválasztható gombokon kívül még két gomb található, melyekkel az balakot annak rétegén belül előtérbe, illetve háttérbe lehet mozgatni. A belső ablakok közül legalább egy mindig látható, azaz nem lehet mind egszerre ikonizált állapotban. Az utolsó ablakot csak alkalmazás esetén lehet bezárni, ekkor a program futása is véget ér. A belső ablakok megjelenítése nyomkövethető.

6.5.5. Swing menük

A MenuElement interfészt, amely a megjelenítésért felelős komponenst lekérdező getComponent, az esetleges almenü elemeit megadó getSubElement, a komponens kiválasztását jelző menuSelectionChanged, valamint a menü eseménykezelését előíró processKey/MouseEvent metódusokat specifikálja, minden menünek használható komponens megvalósítja. Tetszőleges komponensből menü(pont)ot lehet csinálni ezen interfészt megvalósítva.

A MenuSelectionManager vezérli a menürendszer kiválasztási állapotát. Ezen osztály aktuális példányát a statikus defaultManager metódussal lehet lekérdezni. A események továbbítása a menükhöz a feladata, valamint a kiválasztott menüpont nyilvántartása, melynek menürendszeren belüli útvonalát a get/setSelectedPath metódusokkal lehet kezelni. Az aktuálisan kiválasztott menüpont változása mindig ChangeEvent eseményt generál.

Menüpontok:

A menüpontokat a JMenuItem osztály reprezentálja. Ez természetesen megvalósítja a MenuElement interfészt. Tulajdonképpen egy menüpont nem más, mint egy listában megjelenő nyomógomb, ezért ezen osztály az AbstractButton leszármazottja. Ez az ősosztály teszi lehetővé például adott menüpontok tiltását, billentyűparancsok használatát, menüpontokban ikonok megjelenítését, és ActionEvent generálását a menüpont kiválasztásakor. A következő események kezelésére szolgál a bevezetett új metódusok többsége:

- Egy MenuKeyEvent esemény jelzi a menüponton történt billentyűzeteseményeket, ez az esemény az ős KeyEvent-et csak a menü elérési útvonalát megadó getPath,és a menü kiválasztási vezérlőjét elérő getMenuSelectionManager metódusokkal egészíti ki. Az esemény MenuKeyListaner figyelői – melyek billentyűlenyomás (menuKeyPressed), felengedés (menuKeyReleased), és megnyomás (menuKeyTyped) figyelését végzik – listáját az add/removeMenuKeyListener metódusokkal kezelhetjuk.
- Egy MenuDragMouseEvent esemény jelzi a menüponton történt egérhúzási eseményeket, ez az esemény az ős MouseEvent-et csak a menü elérési útvonalát megadó getPath, és a menü kiválasztási vezérlőjét elérő getMenuSelectionManager metódusokkal egészíti ki. Ezen esemény MenuDragMouseListener figyelőinek listáját

az addMenuDragMouseListener és removeMenuDragMouseListener metódusokkal kezelhetjük (a MenuDragMouseListenerfigyelőinek listája az egérhúzása megkezdése (menuDragMouseDrag), befejezése (menuDragMouseReleased), valamit az egérhúzás közben menü fölé történő belépés (menuDragMouseEntered) és kilépés (menuDragMouseExited) figyelését végzik).

A menüpontok megjelenítését az UIDefaults következő jellemzői szabályozzák:

- MenuItem.acceleratorFont: Gyorsbillentyű ábrázolásának betűtípusa.
- MenuItem.acceleratorForeground: Gyorsbillentyű ábrázolásánakelőtérszíne
- MenuItem.acceleratorSelectionForeground: Gyorsbillentyű kiválasztott ábrázolásának előtérszínét megadó java.awt.Color.
- MenuItem.arrowIcon: Almenüt jelölő ikont megadó Icon.
- MenuItem.background: Háttérszínt megadó java.awt.Color.
- MenuItem.border: Keretet megadó javax.swing.border.Border.
- MenuItem.borderPainted: Keret megrajzolását szabályozó Boolean.
- MenuItem.checkIcon? Kiválasztást jelölő ikont megadó Icon.
- MenuItem.disabledForeground: Tiltott menüpont java.awt.Color előtérszíne.
- MenuItem.font: Betűtípust megadó java.awt.Font.
- MenuItem.foreground: Előtérszínt megadó java.awt.Color.
- MenuItem.mergin: Keretméretet megadó java.awt.Insets.
- MenuItem.selectionBackground: Kiválasztás háttérszínét megadó java.awt.Color.
- MenuItem.selectionForeground: Kiválasztás előtérszínét megadó java.awt.Color.

Megjelölhető menüpontok:

Egy JCheckBox menüpontra történő ágyazásának tekinthetők tulajdonképpen a megjelölhető menüpontok, mint ahogy az azt (JCheckbox) reprezentáló osztály neve,a JCheckBoxMenuItem is mutatja. A MenuItem leszármazottja ezen osztály, menüpont lévén, azt csak a menüpont megjelölését lekérdező/megadó get/setState metódussal bővíti ki. Megjelölhető menüpontok megjelenítését az UIDefaults ezen jellemzői szabályozzák:

- CheckBoxMenuItem.acceleratorFont: Gyorsbillentyű ábrázolásának betűtípusa.
- CheckBoxMenuItem.acceleratorForeground: Gyorsbillentyű ábrázolásának előtérszínét megadó java.awt.Color.

- CheckBoxMenuItem.arrowIcon: Almenüt jelölő ikont megadó Icon.
- CheckBoxMenuItem.background: Háttérszínt megadó java.awt.Color.
- CheckBoxMenuItem.border: Keretet megadó javax.swing.border.Border.
- CheckBoxMenuItem.borderPainted: Keret megrajzolását szabályozó Boolean.
- CheckBoxMenuItem.checkIcon: Kiválasztást jelölő ikont megadó Icon.
- CheckBoxMenuItem.disabledForeground: Tiltott menüpont java.awt.Color előtérszín.
- CheckBoxMenuItem.font: Betűtípust megadó java.awt.Font.
- CheckBoxMenuItem.foreground: Előtérsznt megadó java.awt.Color.
- CheckBoxMenuItem.margin: Keretméretet megadó java.awt.Insets.
- CheckBoxMenuItem.selectionBackground: Kiválasztás java.awt.Color háttérszíne.
- CheckBoxMenuItem.selectionForeground: Kiválasztás java.awt.Color előtérszíne.

Csoportokba szervezhető menüpontok:

Tulajdonképp egy JRadioButton menübe történő ágyazásának tekinthetők a csoportokba szervezhető menüpontok, mint ahogy azt a reprezentáló JRadioButtonMenuItem osztály neve is mutatja. Ezen osztály – menüpont lévén – a MenuItem leszármazottja.

Elválasztó vonalak:

A JSeparator osztály használható a menüpontok közti elválasztó vonalak reprezentálására. Az irányítottságának megfelelő vonalként jelenik meg ezen komponens. Irányítottságát a get/setOrientation metódusával lehet megtudni/megadni vagy a konstruktorban beállítani a SwingConstants konstansait felhasználva. Megjegyzendő, hogy ez a komponens teljes értékű Swing felületelemként is használható, nemcsak menükön belül. Ezen elválasztó vonalak megjelenítését az UIDefalóults következő jellemzői szabályozzák:

- Separatpr.highlight: Kiemelés színét megadó java.awt.Color.
- Separator.shadow. Érnyék színét megadó java.awt.Color.

Menütípusok:

A menüpontok listába foglalva menüt alkotnak. Tehát minden menü egy olyan konténer, amely JMenuItem objektumokat tartalmaz. A menüpontok felvételénél arra kell figyelni, hogy mindig csak egy menühöz tartozhat adott menüpont, ha tehát több menübe is felveszünk egy menüpontot, az csak a legutoljára felvett helyen fog megjelenni.

Felbukkanó menük:

Mindig egy különálló kis ablakban jeleníti meg menüpontjait egy felbukkanó menü, amit a komponenshierarchián kívül álló Popup osztály reprezentál. Ennek egyetlen feladata a felbukkanó ablak megjelenítése (show) és elrejtése (hide), ezért megegyezés szerint más látható tulajdonsága (például a mérete, vagy pozíciója) nem változtatható meg. Ez a PopupFactory osztály által végzett gyorsító tárazás hatékony megvalósításával magyarázható. A szülőkomponensben lehet egy osztott példányát kérni, felbukkanó Popup objektumot pedig a getPopup metódusa segítségével lehet.

A menüt magát a JPopupMenu osztály reprezentálja, amely – mivel menüelemekkel dolgozik – megvalósítja a MenuElement interfészt. A menüelemek hozzá vétele az add konténermetódussal vagy az insert metódussal történik, ennek paramétere nemcsak JMenuItem, hanem Action művelet vagy akár szöveg is lehet. Ilyenkor a szükséges menüpont objektum automatikusan létrejön. Az addSeparator metódussal lehet a menübe felvenni az elválasztó vonalat.

Normál menük:

A JMenu osztály reprezentálja az általános menüket, ez az osztály a JMenuItem leszármazottja, ezáltal lehetővé téve ilyen menük menüpontként történő használatát, tehát almenük megvalósítását. Így tulajdonképpen egy normál menü egy menügombnak is felfogható, ugyanis egy automatikusan létrejövő JPopupMenu felbukkanó menü fogja tartalmazni a menü elemeit (azt a getPopupMenu metódussal lehet lekérdezni), ami csak akkor jelenik meg, ha a felhasználó megnyomja a menüt reprezentáló gombot, vagy elég csak az egérrel adott menügomb felé mozdulni ha már megjelent a menü. Ha egy menüsor tartalmazza a normál menüt (ezek az úgynevezett felsőszintű menük, ezen tulajdonságát pedig az is TopLevelMenu metódussal lehet lekérdezni), akkor a felbukkanó menü rendszerint az adott menüpont alatt, míg almenük esetén az adott menüpont mellett fog megjelenni.

Menüsorok:

A JMenuBar osztály reprezentálja a menüsorokat, ez az osztály – mivel menüelemekkel dolgozik – megvalósítja a MenuElement interfészt. Tehát egy menüsor egy olyan konténer, melynek JMenu menük lehetnek az elemei. A getMenuCount metódussal lehet lekérdezni a menüpontok számát. Kinevezhetjük a menüpontok egyikét súgó menüpontnak, melynek

megjelenítése rendszerint eltér a többi menüpont megjelenítésétől. A súgó menüpontot a get/setHelpMenu metódusokkal lehet elérni.

Egyszerre mindig csak egy elemet lehet kiválasztani a menüsor menüpontjai közül. A SingleSelectionModel kiválasztási modell feladata, hogy ezt szabályozza. A get/setSelectionModel metódussal lehet ezt lekérdezni/beállítani. Az adott menüpontot a setSelected metódussal lehet kiválasztani.

7. A program bemutatása

A szakdolgozatban bemutatott példaprogramnak 3 menüje van, File, Színszűrők és Képek helyrehozása a nevük. A következőkben ezen menükhöz kapcsolódó menüpontok bemutatása következik.

A File nevű menünek 3 menüpontja van, Kilépés, Menés és Megnyitás.

Kilépés:

Ha rákattintunk a Kilépés menüpontra először egy felugró ablak jelenik meg, melynek szövege: Kívánja menteni a képet?, 2 lehetőség van: vagy mentjük a képet vagy nem. Ha menteni akarjuk a képet, akkor meghívódik a Mentés osztály és elmenthetjük a képet, utána kilép a programból, ha nem kívánjuk menteni, akkor egyszerűen csak kilép.

Megnyitás:

Ahhoz, hogy szerkeszteni tudjuk egy képet először meg kell nyitni, erre a Megnyitás menüpont alkalmas. Ekkor felugrik egy ablak, melyben ki lehet választani, mely képet szeretnénk szerkeszteni. Ha kiválasztottuk és ezt a megfelelő gomb (Open) segítségével érvényesítettük, megjelenik a kép, melyet ezután manipulálhatunk. Ellenkező esetben, ha mégsem szeretnék kiválasztani képet, ezt a Cancel gomb segítségével tehetjük meg, vagy kiixelhetjük a felugró ablakot.

Mentés:

A Mentés menüpontra kattintva szintén egy fájlkiválasztó ugrik fel, ahol kiválaszthatjuk, hogy hol legyen letárolva a kép. A File Name nevű szöveges mezőben az elmentendő kép nevét kell megadni a kiterjesztésével együtt, ami jpg, png és gif lehet. Ha nem adunk meg kiterjesztést a fájlnak, akkor a jpg állítódik be.

A Színszűrők menünek 7 menüpontja van, ezek a Fekete-szürke-Fehér, Fekete-fehér, Színtől-fehér, Sárgult, Világosít, Sötétít és a Kontrasztosít.

Fekete-szürke-Fehér:

Ezen menüpont a képet fekete-fehérré alakítja át.

A javaban egy képpont színét az rgb érték határozza meg, ez 4 csatornából épül fel: alpha, red, green és blue. Ezekből biteltolásokkal logikai és-vagy műveletekkel kiszámíthatjuk a java által használt rgb értéket. Ezen művelet: tegyük fel, hogy az rgbAlpha az alpha, az rgbRed a

red, az rgbGreen a green és az rgbBlue a blue színcsatornák adottak. Ekkor a ((rgbAlpha & 0xFF)<<24)| ((rgbRed & 0xFF)<<16)| ((rgbGreen & 0xFF)<<8)| ((rgbBlue & 0xFF)<<0) logikai műveletsorozattal megkapjuk a képpont színének értékét. Visszafele is működik, ha a képpont értékéből szeretnénk megkapni a csatornák értékeit, ekkor az alphát a double rgbAlpha = (rgb >> 24) & 0xff; logikai művelettel kapjuk meg. A többi: double rgbRed = (rgb >> 16) & 0xff; double rgbGreen = (rgb >> 8) & 0xff; double rgbBlue = (rgb >> 0) & 0xff.

Egy képpont akkor fehér, ha az rgb értékei: a red: 255, a green: 255 és a blue: 255. Akkor fekete, ha mind a red, mind a green és mind a blue értéke: 0. Akkor szürkeárnyalatú a képpont, ha a red, a green és a blue értékei megegyeznek. Mivel színes képeknél ezek az értékek különbözőek, ezért ha veszem a képpont színének értékét és arra hajtom végre azt a műveletsorozatot, amelyet a csatornák színeiből az rgb értékének kiszámítására használtam, akkor eredményül egy szürkeárnyalatos képet kapok.

Fekete-fehér:

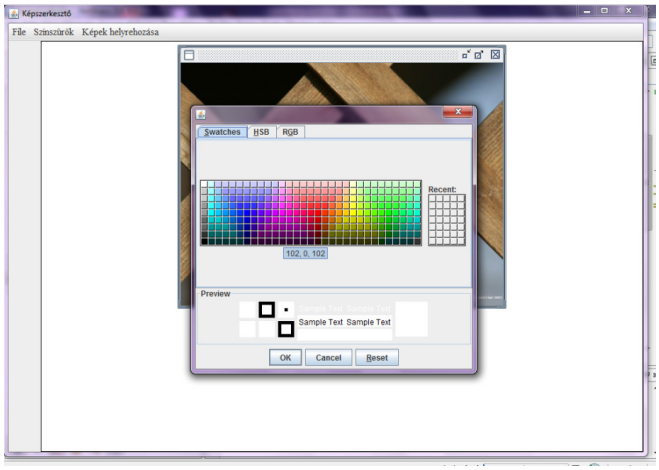
Ezen menüpont olyan funkciót lát el, hogy a szerkesztendő képet csak fekete és fehér színekkel ábrázolja. Veszem a képen előforduló legsötétebb és legvilágosabb színt, a sötétből kivonom a világosat, majd osztom 2-vel, így megkapom a 2 között elhelyezkedő átlagos sötétségű színt. Ez alapján, ha ennél az értéknél kisebb a képpont értéke, akkor fekete, egyébként fehér színű lesz. Azért az átlagértéknél kisebb értékű képpont lesz a fekete, mert a fekete a legkisebb rgb értékű szín, a fehér meg a legnagyobb.

Színtől-fehér:

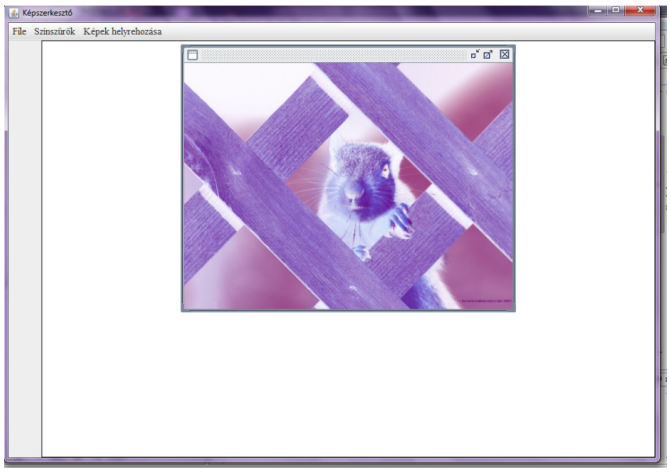
Ez a funkció azt csinálja, hogy képet a kiválasztott színnel, fehérrel, illetve a 2 szín közötti árnyalatokkal ábrázolja úgy, hogy a legsötétebb pixelek fehérek lesznek, a legvilágosabbak pedig a kiválasztott szín. A színt egy színkiválasztóval lehet megadni, ez a JColorChooser, amelyről már volt szó az előbbiekben. Ki lehet választani a színt palettából, HSB színrendszer alapján, valamint az RGB színrendszer alapján ki lehet keverni. Példa a színtől-fehér menüpont funkciójára: Az első kép (7.1-es ábra) az eredeti kép, ez a http://www.hatterkepek.hu/download.php?img=305638 oldalról származik.

7.1-es ábra

A második képen (7.2-es ábra) a színkiválasztóval kiválasztom a színt, amelynek ez esetben a red csatornája: 102, a green: 0 és a blue pedig 102. Az 7.3-as ábrán a funkció eredményét lehet látni.



7.2-es ábra



7.3-as ábra

A Színtől-fehér menüpont kiválasztásakor a SzíntőlFehérEsemény osztály hívódik meg. Ezen osztály rgbSzíntőlSzámol metódusa számolja ki a szerkesztett kép pixeleinek színeit. Elsőként az eredeti képpontok rgb értékéből kiszámolja az rgb csatornák értékeit, majd ugyanezt megcsinálja a kiválasztott színnel. Ezután kivonja ezeket az értékeket (255 – kiválasztott szín egy csatornája) egyenként 255-ből, ekkor megkapjuk, hogy a legvilágosabb érték és az eredeti között hány érték szerepel (és éppen ezért 255-ből vonjuk ki, mert ha a red és a green és a blue is 255, akkor a képpont fehér), ezek lesznek azok a csatornák, amelyekkel az eredmény kép ábrázolva lesz. Majd elosztjuk 255-el, ezzel megkapjuk az arányt, tehát, hogy a választott szín csatornájának értéke és 255 közötti értékekkel lehessen ábrázolni 255 féle csatornát. Mivel pixel értéke csak egész szám lehet, ezért kerekíteni kell, de így is jobban közelít a kiszámítandó értékhez, mintha csak a 255-csatornaérték csatornaértékkel lehetne ábrázolni a képet. Az osztás eredményével megszorozzuk az eredeti kép pixeleinek csatornáit, így arányossá téve őket, majd ezeket kivonjuk 255-ből, hogy a csatorna értéke 255 és a kiválasztott szín csatornájának értéke közé essen és ez a végeredmény. Pl.: az előbbi példa alapján a kiválasztott szín red csatornája: 102 → (255-102)/255 = 0.6, green: 0 → (255-0)/255 = 1, blue: 102 → (255-102)/255 = 0.6. Tehát az arányok red, green, blue csatornákon rendre: 0.6, 1, 0.6. Az eredeti képből vegyük a 476. sor 2. oszlop pixelét, ennek rgb értéke: -6259890, ebből számolva a red csatorna értéke: 160, a green: 123 és a blue: 78. Ezután az eredeti kép csatornái * arány műveletek következnek, ez alapján:

red = 160 * 0.6 = 96, green = 123 * 1 = 123, blue = 78 * 0.6 = 46.8, majd 255-ből kivonjuk ezeket, tehát: red = 255 – 96 = 159, green = 255 – 123 = 132, blue = 255 – 46.8 = 208.2. Végül kiszámoljuk, hogy a csatornák megváltozott értékével milyen rgb értékű lesz a pixel, ez esetben a 476. sor 2. oszlopának képpontja: -6322992.

Sárgult:

Ezen funkció a képet sárgulttá varázsolja.

Világosít:

Ha a Világosít menüpontra kattint valaki, akkor a Halványít osztály hívódik meg. Ezen osztály számol metódusa a pixel rgb értékét kapja paraméterként, majd ebből kiszámolja az rgb csatornák értékeit, majd ezen számokat 10-el növeli, egészen míg kisebb vagy egyenlők, mint 245. Ha egyszer kattint a felhasználó a Világosít menüpontra, akkor 10-el növeli csatornái értékét.

Sötétít:

A Világosít menüpont ellentéte, ugyanazt csinálja, csak 10-el csökkenti a csatornák értékét, így a kép sötétebb lesz.

Kontrasztosít:

Ezen funkció kontrasztosítja a szerkesztendő képet. Összesen 256 féle értéke lehet egy színcsatornának, 256/2 = 128 a fele ennek, ha egy csatorna értéke nagyobb vagy egyenlő mint ez a szám (128) és kisebb vagy egyenlő, mint 245, akkor hozzáadok 10-et a csatorna értékéhez. Azért kell, hogy a feltételben 245-nél kisebb legyen vagy egyenlő vele, mert, ha hozzáadok 10-et, akkor így nem fog olyan értéket visszaadni, amilyet nem vehet fel a színcsatorna. Ha a csatorna értéke kisebb, mint 128 és nagyobb vagy egyenlő, mint 10, akkor, a csatorna értékéből kivon 10-et. Végül a színcsatornák értékeiből kiszámolja a képpont rgb értékét.

A Képek helyrehozása menünek 4 menüpontja van, ezek: Zajok eltűntetése, Karcolat, Élek, Színes élek, Élesít (erősen), Élesít (enyhén).

Zajok eltűntetése:

A zaj több eltérő frekvenciájú és intenzitású jel zavaró összessége. Érdektelen, információ tartalom nélküli adat/jel. A zajok típusai:

- Kép független, additív zaj (fehér zaj):
  Tipikus csatorna átviteli zaj. $g(x,y)=f(x,y)+v(x,y)$.
- Korrelálatlan multiplikatív zaj:
  TV antenna raszter csíkjai. $g(x,y)=f(x,y) \bullet v(x,y)$.
- Kvantálási zaj:
  Az eredeti jelérték folytonos, a kvantált jelérték diszkrét, a különbség véletlen zajként jelenik meg. $v(x,y)=f_{kvantált}(x,y) - f_{eredeti}(x,y)$.
- Só – és - bors zaj:
  Pontszerű, korrelálatlan, véletlen zaj. Legtöbbször szélsőértékű (fekete és fehér). Tipikus egyes fajta űrfelvételekre [11], [12].

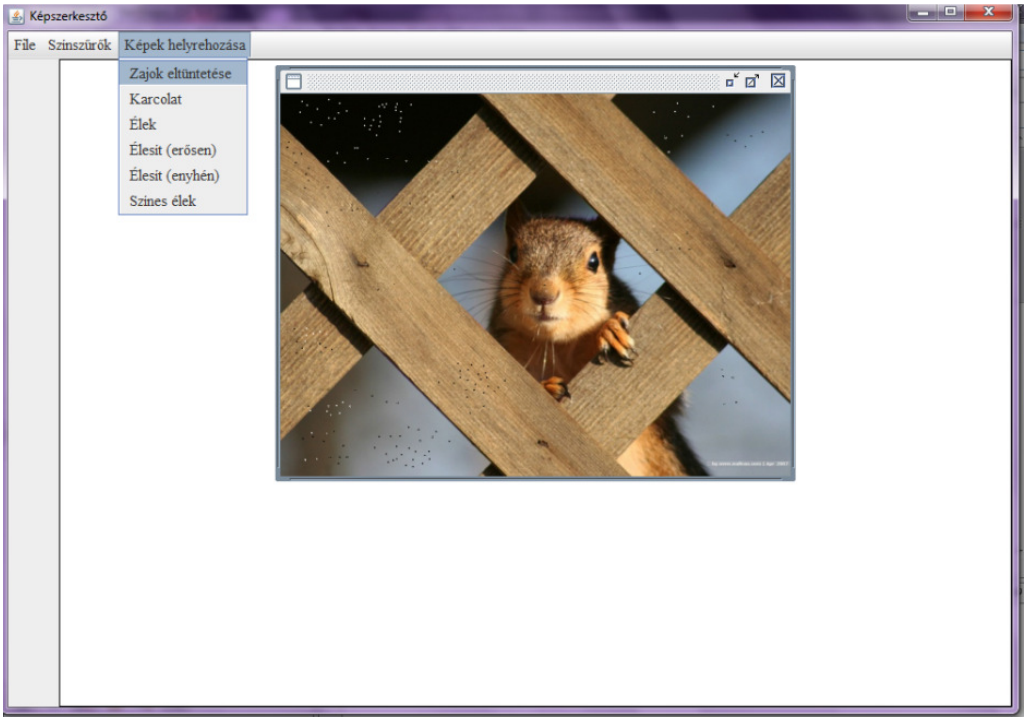
A Zajok eltüntetése menüpont szolgál a példaprogramban a zajok eltűntetésére. Ha a felhasználó rákattint ezen funkcióra, akkor a ZajEltűntetés nevű osztály hívódik meg, mely 3 X 3-as medián szűréssel hajtja végre feladatát. A medián szűrés alapötlete, hogy a kiugró

intenzitásértékek (zajok) a sorba rendezéskor a sorozat széleire szorulnak, a középső elem az adott képtartomány átlagos intenzitását mutatja. A medián szűrés sorba rendezi a vizsgált képtartomány pontjainak (ez esetben 9 pixel) intenzitását növekvő sorrendbe, majd az így kapott sorozat középső elemét (mediánját) kiválasztjuk, amelyet a centrális képponthoz rendelünk új intenzitásértékként. Általában az x_5 intenzitást is felhasználjuk a sorba rendezésben, hogy valóban létezzen a sorozatnak „fizikailag" középső eleme. Amennyiben a centrális képpontot ajánlatosabb kihagyni ebből, úgy a sorozat 8 elemet tartalmaz, amelyből értelemszerűen a 4. vagy 5. elemet tekinthetjük középsőként. A medián szűrés népszerű módszer a zajszűrésben, melynek egyik oka, hogy az eljárás során nem jelennek meg új intenzitásértékek (szemben például az átlagoló szűréssel). A medián szűréshez használhatunk nagyobb maszkokat is, például 5 X 5-ös vagy 7 X 7-es maszkokat, illetve tekinthetünk több iterációs lépést is [13].

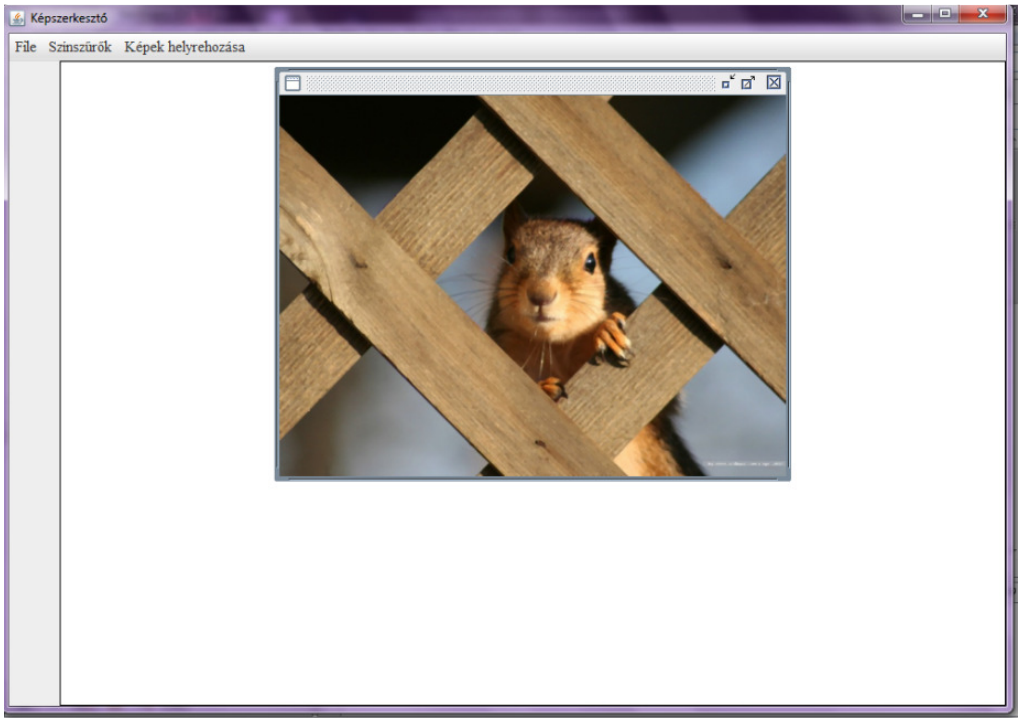
A ZajEltűntetés osztályban a számol metódus elkezdi bejárni a szerkesztendő képet úgy, mintha egy 3 X 3-as mátrix lenne egy elem, tehát a kép legelső bejárt eleme az 1. sor 1., 2., 3., 2. sor 1., 2., 3. és a 3. sor 1.,2.,3. pixelei lesznek. A második bejárt elem az 1. sor 2., 3., 4., 2. sor 2., 3., 4. és a 3. sor 2., 3., 4. pixelei, tehát a 1. bejárt elemet eltoljuk 1 pixellel jobbra. Egészen addig folytatja ezt míg igaz az a feltétel, hogy az adott elem legkülső oszlopa kisebb, mint a sor vége – 1, ez azért van, hogy ne menjen a képen kívülre a bejárás. Ha elérte a sor végét, utána a sor elején folytatja a bejárást, csak 1-el lejjebb tolva a „mátrixot". Tehát a 2. sor 1., 2., 3., 3. sor 1., 2., 3. és a 4. sor 1., 2., 3. pixelei lesznek a következő elemei, ezután ugyanúgy folytatjuk a bejárást, mint azt az előbbiekben az 1. sor bejárásánál leírtam. A soronkénti bejárást kisebb, mint a sorok száma – 1-ig folytatja. A kép „mátrix általi" bejárásán kívül még az elemeket is külön be kell járni. Az 1.sor 1. elemét, a 2.-at stb, amelyek ez esetben 9-9 elemet tartalmaznak. Ezt a 9 elemet elemenként sorba rendezzük és kiválasztjuk a sorba rendezés következtében előálló középső elemet, a mediánt és ezt adjuk értékül az adott elem kilences középső elemének. Például az 1. sor 1. eleme a következő pixelekből áll: 1. sor 1., 2.,3., 2. sor 1., 2., 3. és a 3. sor 1., 2., 3., tehát ennek az elemnek a középső értéke a 2. sor 2. eleme.

Az 1. kép (7.4-es ábra) zajokkal terhelt. A következő kép (7.5-ös ábra) a medián szűrő hatására keletkezett az előbbi képből. A zajok eltűntek a képről viszont kissé homályosabb, elmosódottabb lett, ez hátránya a medián szűrőnek, de például az átlagoló szűrő is rendelkezik

ezzel a hátulütővel. Minél többször hajtom végre egy zajos képen a Zajok eltűntetése menüpont funkcióját, annál kevesebb zaj lesz a képen, esetleg teljesen el is tűnnek, de annál elmosódottabb is lesz.



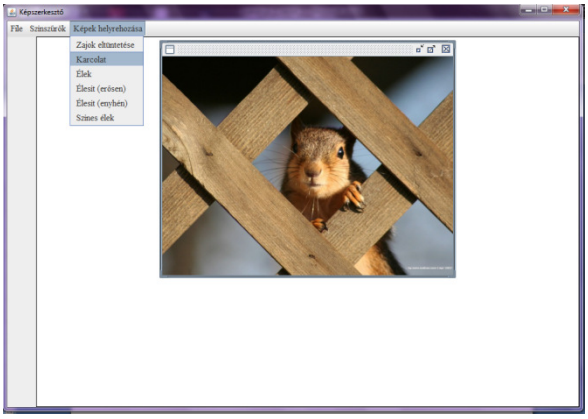
7.4-es ábra



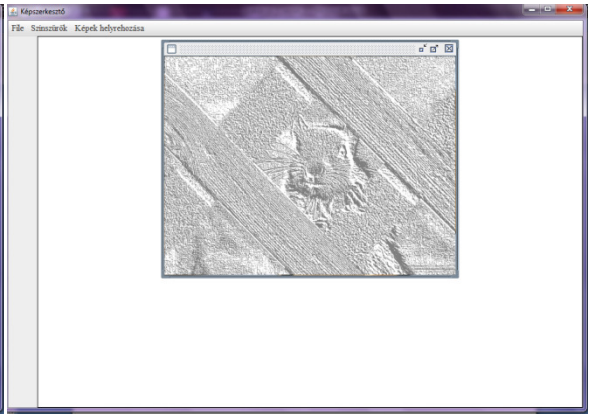
7.5-ös ábra

Karcolat:

Ha erre a menüpontra kattint a felhasználó, akkor a képnek (7.6-os ábra) egy karcolathoz hasonló képét kapja vissza (7.7-es ábra). Az előbbiekben taglalt módon járjuk be a képet. Konvolúciós eljárást [14] hajtunk végre és a függőleges és vízszintes irányú Sobel – operátort használva kapunk 2 számot, a szamX-et és a szamY-t. A theta-t úgy kapjuk meg, hogy a szamY-t, tehát a vízszintes irányra vonatkozó Sobel – operártot elosztjuk a szamX-el, a függőleges irányra vonatkozó Sobel – operátorral. Végül, ha theta nagyobb, mint 0, akkor az adott pixel színe fehér, ha egyenlő vele, akkor szürke, ellenkező esetben pedig fekete lesz.



7.6-os ábra



7.7-es ábra

Sobel operátor, mely vízszintes irányú: $\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$

Sobel oprátor, mely függőleges irányú: $\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$

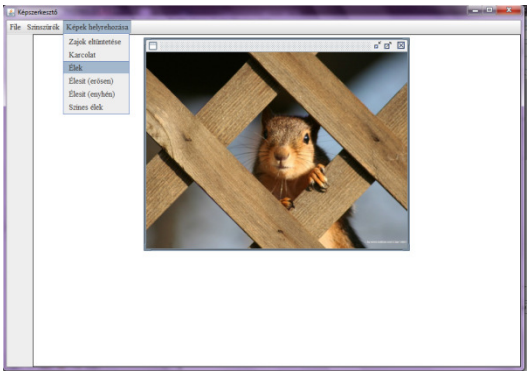
Élek:

A példaprogramban található egy Élek nevű menüpont, ennek funkciója az élek kiemelése, eredményként fekete alapon a kép élei jelennek meg fehérrel ábrázolva (7.9-es ábra), melyet az 7.8-as ábrán látható képből kapunk. Konvolúciós eljárással kapjuk meg a már szerkesztett új képet. Az éldetekt metódus hajtja végre a konvolúciót. Amely olyan, mint az előbbiekben bemutatott medián –szűrés csak nem sorba kell rendezni az elemeket, hanem a konvolúciós mátrix adott elemével össze kell szorozni az adott elemet. Pontosabban bejárjuk a képet úgy, mintha egy 3 X 3-as mátrix lenne egy eleme, mint a medián – szűrésnél. Vegyük az 1. ilyen elemet, ez 9 pixelt tartalmaz (1. sor 1., 2., 3. pixel, 2. sor 1., 2., 3. pixel és a 3. sor 1., 2., 3.

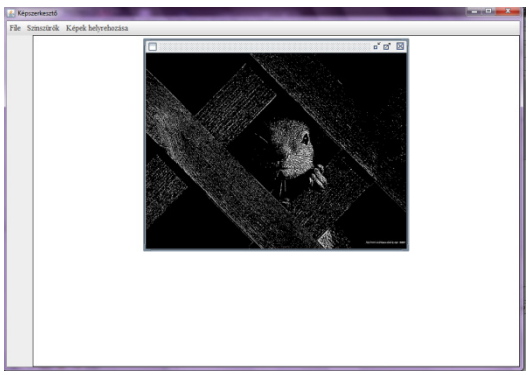
pixel). A konvolúciós mátrix is egy 3 X 3-as mátrix ez esetben és ezért kell „3 X 3-as mártixxal” bejárni a képet. Tehát vesszük az 1. sor 1. pixelének rgb – értékét és összeszorozzuk a konvolúciós mátrix 1. sorának 1. elemével, a kép 1. sorának 2. elemét a mátrix 1. sor 2. elemével és így tovább, majd ezeket összeadjuk. Ha képen például a 234. sor 54., 55.,56, a 235, sor 54., 55.,56. és a 236. sor 54., 55., 56. pixeleknél járunk, akkor a 234. sor 54. elemét a mátrix 1. sor 1. elemével szorozzuk össze és a 236. sor 55. elemét pedig a konvolúciós mátrix 3. sor 2. elemével stb és a 9 szorzatot összeadjuk. Ha összeadtuk a 9 szorzatot, utána el kell osztani 9-el, mert elemű a mátrix. Végül ezt az értéket az elem középső elemének adjuk értéküket, de ennek is, ha nagyobb vagy egynlő, mint 0, akkor a fekete, egyébként a fehér értéket adjuk. Többféle konvolúciós mátrix van, például Sobel gradiens, Roberts gradiens, Laplace – operátor. A példaprogramban a Laplace – operátort használtam.

Ez a következő mátrix:

-1	-1	-1
-1	9	-1
-1	-1	-1



7.8-as ábra

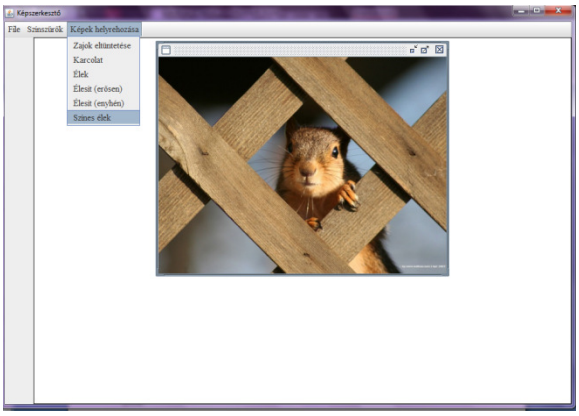


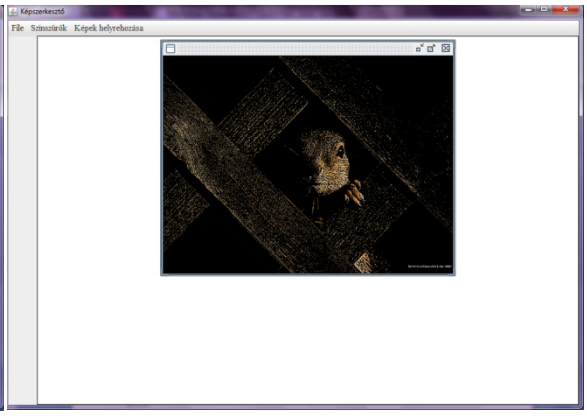
7.9-es ábra

A 2. kép úgy jött létre, hogy az Élek menüpont funkciója lett alkalmazva az 1. képre.

Színes élek:

Lényegében ugyanazt a funkciót hajtja végre, mint az Élek menüpont, csak itt az élek a fekete alapon nem fehérrel, hanem az él eredeti színe – 1-el jelennek meg. A 2 algoritmusa között a különbség az, hogy míg az Élek menüpontban az élek fehér értéket kapnak, itt ugyanúgy fehér értéket kapnak, de az eredeti kép azonos pontjához hozzáadódnak, így az eredeti kép éle – 1 lesz a színe a kirajzolt éleknek. A következő példában az 1. (7.10-es ábra) képen az eredeti kép látható, míg a 2.-on (7.11-es ábra) a Színes élek menüpont funkciójának hatása.

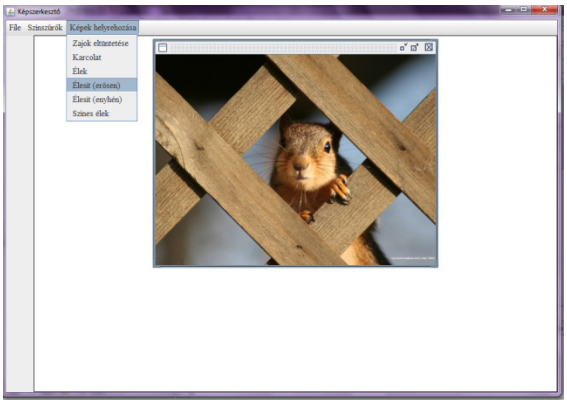


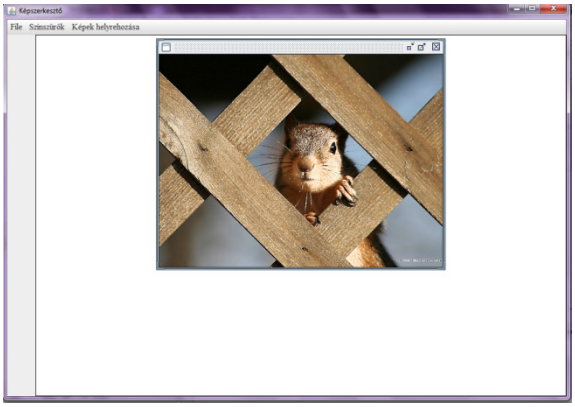


7.10-es ábra **7.11-es ábra**

Élesít (erősen):

Az Élesít menüpont a kép élesítésére szolgál. Mint az élek kiemelésénél, itt is konvolúcióval vannak kiszámolva a pixelek értékei. A konvolúciós mátrix ugyanaz, mint az Élek és a Színes élek menüpontokban. Miután létrejöttek a pixelek új értékei, utána, ha ez a szám kisebb, mint a legkisebb rgb - érték (a fekete), akkor fekete lesz az a képpont, ha nagyobb, mint a legnagyobb rgb – érték (a fehér), akkor a fehér értékét kapja, egyébként pedig ugyanaz marad a képpont színe (7.13-as ábra), mint az eredeti képen (7.12-es ábra). Például:





7.12-es ábra **7.13-as ábra**

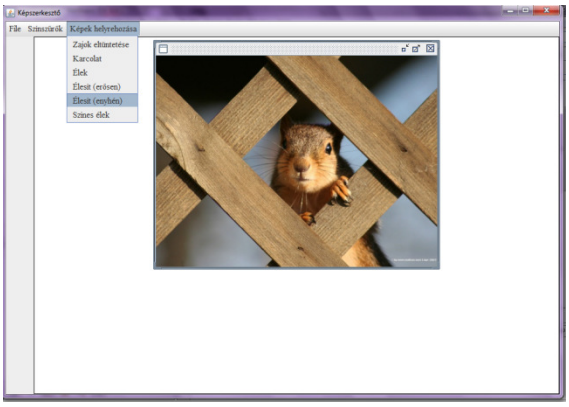
Élesít (enyhén):

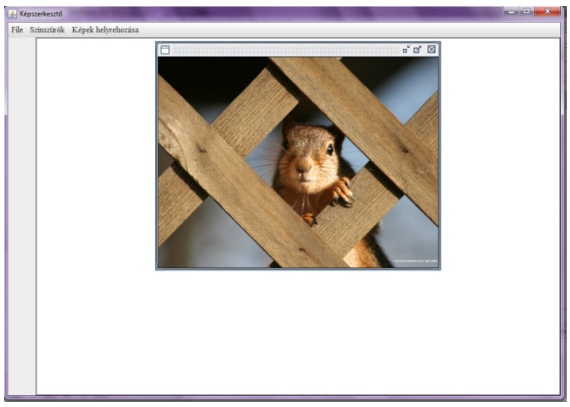
Ugyanaz az algoritmusa, mint az előző funkciónak („Élesít (erősen)”), csak a konvolúció által használt mátrixban, különböznek egymástól.

Ez a mátrix: $\begin{pmatrix} -1 & -1 & -1 \\ -1 & 17 & -1 \\ -1 & -1 & -1 \end{pmatrix}$

Ha elvégeztük a konvolúciót, még el kell osztani egyenként a pixelek értékeit 9–el. Miután ez megvan, ha az érték nagyobb, mint fekete, akkor a fekete, ha kisebb, mint fehér, akkor a

fehér, egyébként az eredeti képpont értékét kapja meg, mint ezt az előbbi funkcióban, a másik élesítésnél is leírtam. Ez látható az 7.14-es és 7.15-ös képen:





7.14-es ábra **7.15-ös ábra**

Mivel csak kicsit élesíti ez a funkció a képet, így nem olyan szembetűnő, mint az előzőnél (erős élesítés), de valamikor, ha nem szükséges, vagy nem akarjuk annyira élessé tenni, akkor ez a megfelelő menüpont.

A program külső kinézete:

Végül pár szó a példaprogram kinézetéről. Mint az előző képeken is látszik, tartalmaz egy menüsort, mely ismertetése a Swing menük alcím alatt található a korábbiakban. Ez alapján készítettem el ezt a menüsort. Elsőként létrehoztam egy JMenuBar típusú objektumot, menüsor néven. Ezek után a JMenu típusú objektumok (file, színSzűrő, képhelyrehozás neveken) elkészítése jött, például: public JMenu file = new JMenu(”File”);. Végül a menüpontokat hoztam létre, melyek JMenuItem típusúak. A felület metódusban rakom össze igazából a program külsejét. Itt helyezem el a menükbe a menüpontokat, például a file menübe a mentés, megnyitás és kilépés menüpontokat (file.add(kilép); például), valamint a menük és menüpontok betűméretét és nagyságát is beállítom (például: kilep.setFont(new Font(Font.SERIF,Font.PLAIN,15));). Miután ez kész lett, a menüket bepakoltam a menüsorba (például: menüsor.add(file);), majd ezt elhelyeztem a képernyőn a setLayout metódus segítségével (menüsor.setLayout(new FlowLayout(FlowLayout.LEFT,0,0));), így a menüsor a képernyő bal felső sarkába került. A FlowLayout, az elemeket egymás után rakja, de meg lehet adni, hogy, mint a programban balra (FlowLayout.LEFT), jobbra (FlowLayout.RIGHT) vagy középre (FlowLayout.CENTER) pozícionált legyen. Az első 0 azt jelenti, hogy a menü előtt és után vízszintesen 0 pixel legyen, míg a második 0 azt, hogy a

menü alatt és fölött 0 pixel legyen. Layout maneger segítségével tudunk elhelyezni egy konténeren belül komponenseket. Az előbb ismertettem a FlowLayout-ot, de van még BorderLayout, GridLayout, CardLayout, GridBagLayout, stb. A BorderLayout segítségével maximum 5 komponens helyezhető el az égtájaknak megfelelően (NORTH, WEST, EAST, SOUTH, CENTER) a konténer széléhez igazítva. Készítettem a menüsoron kívül még 2 komponenst, ezek JPanel típusú objektumok. Az egyik neve szerkesztő, a másiké pedig bal. A szerkesztőben fog megjelenni a megnyitott kép, a bal pedig esztétikai szempontokat szolgál (egy 50x500-as kis JPanel). Létrehoztam egy konténert (Container cp =getContentPane();), hogy ezeket el tudjam benne helyezni és a képernyő kinézetét rendezni tudjam. A cp-ben a BorderLayout segítségével helyeztem el a komponenseket (például cp.add(szerkesztő, BorderLayout.CENTER);), így a menüsor a képernyő tetejére került, a bal bal oldalra, míg a szerkesztő középre.

A FelhasználóiFelület osztályban található még az actionPerformed metódus, bár ez nem a program kinézetét adja. Ez a metódus adja meg, hogy mi történjen, ha rákattintok egy gombra. Létrehoztam egy Stringet, az ac-t. Ennek az értéke meg fog egyezni annak a gombnak a nevével, amire rákattint valaki (String ac = e.getActionCommand), az e egy ActionEvent típusú objektum. Ezek után jöhetnek az összehasonlítások, hogy melyik gomb nevével egyezik meg az ac. Ha az ac a „Megnyitás”-al egyenlő, akkor példányosítom a MegnyitGombEseményt és meghívom a megfelelő metódusait, valamint a folytat boolean típusú objektumnak true értéket adok, különben, ha a „Kilépés”-el egyezik, példányosítom a KilépGombEsemény-t és meghívom a megfelelő metódusait. Ha nem a „Megnyitás”-sal és nem a „Kilépéssel”-el egyezik, de a folytat objektum értéke true, tehát, korábban már meg lett nyitva egy kép, akkor összehasonlítom a többi gomb nevével, ha egyezés van példányosítom az adott gombhoz tartozó osztályt és végrehajtom a megfelelő metódusokat. Ellenkező esetben, ha a folytat értéke false és nem egyezik sem a „Megnyit”, sem a „Kilép”-el az ac értéke, akkor nem csinál semmit, mivel ez a lehetőség kizárt, mert nem tud valaki olyan gombra kattintani a képernyőn, ami nincs ott.

A kép módosítása úgy lett megoldva, hogy a szerkesztő JPanelen levő JInternalFrame letörlődik (a JFrame-ben van elhelyezve a megnyitott kép (szerkesztő.remove(frame);)). Újrapéldányosítom (frame = new JInternalFrame(null,true,true,ture,ture);), majd a MegnyitGombEsemény típusú megnyitEs objektum kirajzol metódusa segítségével

létrehozom a módosított képet, amely a JInternalFrame típusú, újrapéldányosított frame-ben fog megjelenni (megnyitEs.kiRajzol(image,frame);), majd ezt hozzáadom a szerkesztőhöz (szerkesztő.add(frame);).

8. Összefoglalás

A Java SE kiválóan alkalmas a hagyományos desktop alkalmazások, illetve appletek készítésére, így fotók, ábrák módosítására is.

Mint azt a célkitűzésekben megfogalmaztam, végeredménynek egy olyan szoftvert képzeltem el, mely tartalmaz különböző képeket alakító funkciókat, valamint az alapvető fájlkezelési műveleteket, melyeket a programomban a „File” menüben találhatunk meg.

Mint azt a bevezetőben írtam, a régi megkopott fényképek helyrehozására sok szoftvernek rengeteg funkciója van, így a szakdolgozat példaprogramjának is. Ezen képek „újjá tételének” egyik legfontosabb lépése, a zajok eltűntetése az ábráról, melyet a programomban a „Zajok eltűntetése” menüpont használatával tehetünk meg. A zajok eltűntetésének viszont hátránya, hogy az élek elmosódottak lesznek. A másik legfontosabb lépése a képek „újjá tételének”, hogy a kép élei látszódjanak, a szakdolgozatomban az „Élesít (enyhén)” illetve „Élesít (erősen)” címek alatt található annak a leírása hogyan valósítottam meg a programomban az élesítést.

Sokszor nem a régi fotókat szeretnénk „újjá tenni”, hanem új fotókból „régi hatásút” készíteni. Erre való például a kép fekete-fehérré tétele, mely a „Fekete-szürke-Fehér” cím alatt található a dolgozatomban. Kiemelhetjük a képek éleit fehérrel fekete alapon, illetve fekete alapon az éleket a kép éleinek megfelelő színekkel. Ha azt szeretnék, hogy a kép csak 2 színből álljon, és a fehér legyen a legsötétebb, és egy általunk kiválasztott szín a legvilágosabb, az is megoldható a programommal, a „Színszűrők” menü „Színtől-fehér” menüpontját használva.

A multimédiához szorosan hozzákapcsolódik a kép. Általában minden multimédiás alkalmazásban megtalálhatóak ezek, így dolgozatomban ezzel foglalkozom, mivel elég fontos szerepet töltenek be a mindennapi életben a fotók, ábrák. Szakdolgozatomban az egyszerűbb láthatóság és jobb érthetőség kedvéért több ábrát is használtam, hogy ezzel szemléltessem a program tevékenységeit, így átláthatóbbá téve azt. A Java programozási nyelvben valamely funkciók már megtalálhatóak, de általam készített új tevékenység például a „Színtől-fehér” menüpont hatása. A Java tartalmazza például az él detektálást [16] vagy az élesítést, [17], ezeket a tevékenységeket újraimplementáltam saját eszközökkel, meglévő matematikai algoritmusok alapján, felhasználva számos matematikus munkásságát.

9. Irodalomjegyzék

[1] Sain Márton: A fény birodalma, Gondolat zsebkönyvek, Budapest, 1980, 7.o, 98. o.- 100. o.

[2] Damjanovich Sándor, Fidy Judit, Szöllősi János: Orvosi biofizika, Medicina Könyvkiadó Rt., Budapest, 2006, 309. o. – 311. o., 317. o., 321.o.

[3] Budó-Mátrai: Kísérleti fizika III., Tankönyvkiadó, Budapest, 1977, 251. o.

[4] George Gamow, John M. Cleveland: Fizika, Gondolat, Budapest, 1977, 287.o. – 289. o.

[5] http://www.mfk.unideb.hu/userdir/dmk/docs/20071/07_1_11.pdf

[6] Schwarz Tibor: Bevezetés a számítógépi grafikába tárgy jegyzete, 93. o. – 95. o.

[7] http://www.epab.bme.hu/oktatas/Jegyzetek/visualization/15_CMY.pdf

[8] http://www.epab.bme.hu/oktatas/Jegyzetek/visualization/14_CMY.pdf

[9] Java 2 Útikalauz programozóknak: 5.0 I., szerk.: Lakatos Attila, Csizmadia Balázs, Csonka Ferenc, 359. o, 360. o, 361. o

[10] Java 2 Útikalauz programozóknak: 5.0 II., szerk.: Lakatos Attila, Csizmadia Balázs, Csonka Ferenc, 240.o – 245.o, 283.o., 298.o, 371.o. – 375.o.

[11] https://wiki.sch.bme.hu/bin/view/Valaszthato/KerteszKerdesek#7 Heurisztikus_sz_r_tervez_s_2_Z 7. tétel

[12] http://www.gamf.hu/portal2/sites/default/files/lecture_03-Szures1.pdf 4.o

[13] http://www.inf.unideb.hu/~hajdua/km_main.pdf 26.o

[14] http://tavmjong.free.fr/INKSCAPE/MANUAL/html/Filters-Pixel.html

[15] http://progmat.uw.hu/oktseg/javaspeci/12_awt_c.pdf

[16] http://www.tomgibara.com/computer-vision/canny-edge-detector

[17] http://www.pages.drexel.edu/~weg22/edge.html

10. Köszönetnyilvánítás

Köszönetet szeretnék mondani témavezetőmnek, Dr. Tornai Róbertnek, aki lehetővé tette, hogy nála írjam a szakdolgozatomat, valamint családomnak, akik támogattak ebben.