PANNON EGYETEM

Matematika Tanszék

Mérnök informatikus BSc szak

SZAKDOLGOZAT

Lineáris és nemlineáris transzformációk szemléltetése számítógéppel

Készítette:

Hosszú Ivett

Témavezető:

Dr. Szalkai István

2013

Nyilatkozat

Alulírott Hosszú Ivett diplomázó hallgató, kijelentem, hogy a szakdolgozatot a Pannon Egyetem Matematika Tanszékén készítettem mérnök informatikus BSc (BSc in Computer Engineering) diploma megszerzése érdekében.

Kijelentem, hogy szakdolgozatomban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy a szakdolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Veszprém, 2013. május 6.

Hosszú Ivett

Alulírott dr. Szalkai István témavezető kijelentem, hogy a szakdolgozatot Hosszú Ivett a Pannon Egyetem Matematika Tanszékén készítette mérnök informatikus BSc (BSc in Computer Engineering) diploma megszerzése érdekében.

Kijelentem, hogy a szakdolgozat védésre bocsátását engedélyezem.

Veszprém, 2013. május 6.

dr. Szalkai István

Köszönetnyilvánítás

Elsősorban szeretnék köszönetet mondani témavezetőmnek, dr. Szalkai Istvánnak, hogy ötleteivel, tanácsaival segítette szakdolgozatom elkészítését, és hogy támogatott az utolsó percig.

Köszönettel tartozom továbbá családomnak, páromnak, barátaimnak, kollégáimnak, hogy munkámat ötleteikkel, véleményükkel és bíztatásukkal segítették.

TARTALMI ÖSSZEFOGLALÓ

Szakdolgozatom témája a lineáris és nemlineáris transzformációk szemléltetése számítógéppel. Dolgozatomban először röviden bemutatom a transzformációk világát, pár szót ejtek a témához kapcsolódó definíciókról, majd egy kis kitekintésként beszélek a transzformációk sokszínűségéről, felbukkanásukról a mindennapokban. Ezután térnék rá a programommal szemben állított elvárások felsorolására, a feladatom ismertetésére. Első lépésként készítettem egy felhasználói kézikönyvet, melyben részletesen ismertetem a program beépített funkcióit, és ezeknek helyes használatát. Ezt követően programozói szemszögből vizsgáltam meg a problémát, ismertettem a tervezési folyamatot, majd részletes leírást adtam a program megvalósításáról. A szoftver tesztelését is dokumentáltam, megvizsgáltam a futási idejét két különböző teljesítményű számítógépen, majd az eredményeket táblázatba gyűjtöttem.

A program megvalósításának nyelvéül a Javat választottam, fejlesztői környezetként pedig az Eclipse IDE bizonyult a legalkalmasabbnak. Első körben megismerkedtem a Java GUI lehetőségeivel, eszközeivel, majd a fejlesztői környezet kiegészítéseivel. A program megtervezése során UML modelleket írtam fel, melyek nagyban megkönnyítették a munkámat. A kész program képes a felhasználó által megnyitott kép lineáris és nemlineáris transzformációjának megjelenítésére, valamint egy Munkalap nézettel rendelkezik, ahová a felhasználó egy koordináta-rendszerbe alakzatokat rajzolhat jellemző adataik megadásával, illetve ezt megteheti adatfájl megnyitásának segítségével. A koordináta-rendszerben lévő ábrák lineáris és nemlineáris transzformációját is bemutatja a program.

Kulcsszavak: Lineáris és nemlineáris transzformáció, koordináta-rendszer, kép transzformáció, GUI

ABSTRACT

The topic of my thesis is the demonstration of linear and nonlinear transformations with computer. In the first place I will write about the world of transformations and the most important definitions, which are connected to the topic. Then I will list the user requirements and introduce my task.

I created a user manual for my application, wherein I gave a specification how to use the software. To reduce the number of errors, I tested the application for several inputs, and I measured the running time on two different machines.

Before I started my work, I had to choose a programming language and an integrated development environment. I selected Java and Eclipse IDE because they were the optimal choice for me. At first I got acquainted with the Java GUI and the Eclipse plug-ins. During the design period I created UML diagrams which made my work much easier. The developed application is able to transform a picture linearly and nonlinearly, and it has a Workspace view with a coordinate - system, where we can draw and transform shapes.

Keywords: Linear and nonlinear transformation, coordinate-system, image transformation, GUI

Tartalomjegyzék

1. Irodalmi áttekintés

Manapság a világban rengeteg olyan mérnöki megoldással találkozhatunk, amely valamilyen transzformációt használ. Jelen van a geológiában, a képfeldolgozásban és a grafikai megjelenítések minden területén. Ezért is fontos, hogy közelebbről is megvizsgáljuk, körbejárjuk ezt a témakört. Először is nézzük meg, mi is az a transzformáció.

1.1. Transzformációk

Az interneten rengeteg definíciót találni a transzformáció szóra, hiszen nem csak matematikailag van értelme.

Használatos a genetikában, fizikában, kémiában és még rengeteg más területen is. Ha nagy általánosságban nézzük, és a szó magyar nyelvű fordítását, a transzformáció mindig valamilyen átalakítást, átváltozást jelent.
Számomra a matematikai értelmezése volt fontos, ezért a következőkben ezzel a témakörrel fogok foglalkozni.

Definíció: transzformáció alatt egy halmaz önmagába való leképezését értjük.

A transzformációk többféle szempont szerint is csoportosíthatók, ezek közül néhány lehetséges módot mutatok be az alábbiakban.

1.1.1. Koordináta-transzformáció

A koordináta-transzformáció során a vizsgált alakzat egy új koordináta-rendszerre vonatkozó koordinátáit határozzuk meg a régiek ismeretében.

Ezt a transzformációt gyakran használják az élet számos területén, legfontosabb alkalmazási területei a geodézia és a távérzékelés.

Erről a transzformáció típusról a továbbiakban nem lesz szó, mivel a szakdolgozat témája a pont-transzformációkra korlátozódik.

1.1.2. Pont-transzformáció

A pont-transzformáció során a koordináta-rendszert nem változtatjuk, de az alakzat pontjaihoz új koordinátákat rendelünk.

A programom a pont-transzformációk bemutatására szolgál.

A továbbiakban a geometriai transzformációkról ejtenék pár szót, hiszen a szakdolgozatom lényegében erre a témakörre épül, illetve tisztáznék néhány fontos definíciót, amelyre a továbbiakban szükségünk lehet.

1.1.3. Geometriai transzformáció

Definíció: a geometriai transzformáció egy olyan leképezés, függvény, amely segítségével egy ponthalmaz minden pontjához hozzárendelünk egy-egy pontot, azaz értelmezési tartományuk is és értékkészletük is ponthalmaz.

A transzformációk ezen a csoporton belül is két részre oszthatók. Az egyik ilyen transzformáció típus a lineáris, a másik pedig a nemlineáris transzformációk csoportja. A továbbiakban e két csoport definícióját és jellemzőit ismertetném egy-egy példán keresztül.[2]

1.1.1.1. Lineáris transzformációk

A lineáris transzformációkat a következő alakban tudjuk felírni:

$$T(\vec{x}) = \mathbf{A}\vec{x},$$

vagyis a *T* transzformáció egy olyan $\mathbb{R}^n$ - ből $\mathbb{R}^m$ - be történő leképezés, ahol $\vec{x}$ egy *n* elemű oszlopvektor, **A** pedig egy *m x n*-es mátrix, melyet a *T* transzformáció mátrixának is nevezünk. [1]

A lineáris transzformáció tulajdonságai:

$$\mathrm{A}(\vec{x} + \vec{y}) = \mathrm{A}\vec{x} + \mathrm{A}\vec{y}$$

$$\mathrm{A}(\lambda * \vec{x}) = \lambda * \mathrm{A}(\vec{x})$$

Nézzünk néhány példát transzformációs mátrixszal együtt[6]:

- Identitás (helyben hagyás): az a geometriai transzformáció, amely a sík (vagy tér) minden pontjához önmagát rendeli hozzá.

  Mátrixa: $\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$

- *x* tengelyre tükrözés: a transzformáció során a sík minden *P* pontjához hozzárendelünk egy *P'* pontot úgy, hogy ha *P* rajta van az *x* egyenesen, akkor *P = P'*, egyébként pedig *P'* a sík azon pontja, amelyre teljesül, hogy *PP'* szakasz felezőmerőlegese az *x* tengely.

  Mátrixa: $\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$

- *y* tengelyre tükrözés: a transzformáció során a sík minden *P* pontjához hozzárendelünk egy *P'* pontot úgy, hogy ha *P* rajta van az *y* egyenesen, akkor *P = P'*, egyébként pedig *P'* a sík azon pontja, amelyre teljesül, hogy *PP'* szakasz felezőmerőlegese az *y* tengely.

  Mátrixa: $\begin{bmatrix}-1 & 0\\0 & 1\end{bmatrix}$

- origóra tükrözés: A transzformáció során a sík minden egyes *P* pontjához hozzárendelünk egy *P'* pontot úgy, hogy ha *P* egyenlő az origóval, akkor *P' = P*, egyébként a *P'* a sík azon pontja, amelyre teljesül, hogy a *PP'* szakasz felezőpontja az origó.

  Mátrixa: $\begin{bmatrix}-1 & 0\\0 & -1\end{bmatrix}$

- *y* tengellyel párhuzamos, λ mértékű nyújtás: a transzformáció során kapott kép az *y* tengellyel párhuzamosan, λ mértékben elnyújtva jelenik meg.

  Mátrixa: $\begin{bmatrix}1 & 0\\0 & \lambda\end{bmatrix}$

- *x* tengellyel párhuzamos, λ mértékű nyújtás: a transzformáció során kapott kép az *x* tengellyel párhuzamosan, λ mértékben elnyújtva jelenik meg.

  Mátrixa: $\begin{bmatrix}\lambda & 0\\0 & 1\end{bmatrix}$

- x tengelyre vetítés: a transzformáció minden pontot az *x* tengelyre képez le.

  Mátrixa: $\begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}$

- *x* tengelyű, λ paraméterű nyírás: olyan transzformáció, amelynél a pontok az *x* egyenessel párhuzamosan mozdulnak el, és az elmozdulásuk nagysága arányos az *x* tengelytől mért távolságukkal. Nyírás hatására a terület nem változik és téglalap képe paralelogramma.

  Mátrixa: $\begin{bmatrix} 1 & \lambda \\ 0 & 1 \end{bmatrix}$

Nem csak a lineáris transzformációk írhatók le mátrixok segítségével. Ilyen transzformációk például a projektív transzformációk, és az affin transzformációk. Róluk is érdemes ejteni néhány szót.

1.1.1.2. Affin transzformációk

Olyan transzformációkat nevezünk affin transzformációknak, melyek illeszkedés- és párhuzamosságtartók. Ezt a transzformációt affinitásnak is nevezzük. A definícióból kiderül, hogy minden lineáris transzformáció affinitás.

Minden háromszögre igaz az, hogy ha transzformáljuk valamilyen affin transzformáció szerint, a képe szintén egy háromszög lesz, illetve minden paralelogramma képe szintén paralelogramma.

Affin transzformáció például a nagyítás, kicsinyítés, tükrözés, elforgatás, nyírás, illetve a hasonlósági transzformációk mindegyike.
Általában az affin transzformáció az elforgatás, tükrözés, dilatáció és nyírás egy kombinációja [3,7].

Az $\mathbb{R}^n$ affin transzformációja egy olyan F: $\mathbb{R}^n \to \mathbb{R}^n$ leképezés, amely az alábbi formában írható le:

$$F(\vec{p}) = \mathbf{A}\vec{p} + \vec{q},$$

ahol minden $\vec{p} \in \mathbb{R}^n$, ahol $\mathbf{A}$ egy lineáris transzformációja az $\mathbb{R}^n$-nek.

1.1.1.3. Projektív transzformációk

Olyan transzformációkat nevezünk projektív transzformációnak, melyek egyenestartók, de nem feltétlenül párhuzamosságtartók. Ebből következik, hogy az affin transzformációk mind projektív transzformációk is.

Egy elterjedt példa a projektív transzformációra a perspektív transzformáció, mely a 3D-s koordinátákból állít elő 2D-s koordinátákat.
A projektív geometria témakörével foglalkozik például Dobos Sándor és Hraskó András feladatgyűjteményükben, melyet 11-12. évfolyam tanulóinak szántak. [4]

1.1.1.4. Nemlineáris transzformációk

A másik fő transzformáció típus, amit a későbbiekben többször is vizsgálni fogunk, a nemlineáris transzformációk csoportja. Itt nem beszélhetünk kötött szabályokról, hiszen szinte bármilyen függvény alapján transzformálhatjuk az *x* és az *y* koordinátákat is. Matematikailag a következő definíciót írhatjuk fel:

$$T(x,y) = \big(f(x,y), g(x,y)\big),$$

vagyis $x' = f(x,y)$, és $y' = g(x,y)$, ahol x és y a transzformálandó pont koordinátái, f és g pedig a transzformáció függvényei.

Fontos megemlítenem, hogy beszélhetünk térbeli és síkbeli transzformációkról is, de a programom kizárólag az utóbbi típus szemléltetésére szolgál.

1.2. Transzformációk a mindennapokban

Mielőtt belekezdenék az általam elkészített szoftver bemutatásába, szeretnék rávilágítani arra, miért is fontos téma a transzformációk vizsgálata. Sokszor nem is hinnénk, hogy amit látunk, tapasztalunk egy matematikai transzformáció eredménye. Egyszerű példa erre a fénykép készítése is, amikor a 3D-s világból képezünk a 2D-s világba.

Egy teljesen másik példa lehet a Rubik-kocka kirakása is, ahol folyamatos elforgatásokat végzünk. Az origami könyvekben is gyakran találkozunk tükrözésekkel, illetve elforgatásokkal. A festészetben szintén visszaköszön egy-egy ábra valamilyen geometriai transzformációja, legyen szó tükrözésről, elforgatásról, kicsinyítésről vagy nagyításról. De fellelhető az építészetben, szobrászatban, népművészetben, csillagászatban, sőt még a gazdasággal kapcsolatos jelentésekben is. Erre ad példát dr. Szalkai István Haladvány kiadványban megjelent cikke, melynek címe: Szemléltetés és becsapás [5]. Ez a

cikk a koordináta-transzformációk alkalmazását írja le érdekes, a hétköznapokból vett példákon keresztül.

A geometriai transzformációkat felhasználja maga a matematika is különböző tételek bizonyítására. Példaként említhetjük a Thalesz-tétel megfordításának bizonyítását is, ahol a középpontos tükrözést alkalmazzuk, illetve szintén ezt a transzformációt használva bizonyíthatjuk a háromszög középvonalára vonatkozó állítások helyességét. (Vagyis a háromszög középvonala párhuzamos a háromszög harmadik oldalával, és hossza a harmadik oldal hosszának felével egyenlő.

2. A program

2.1. Feladat megfogalmazása

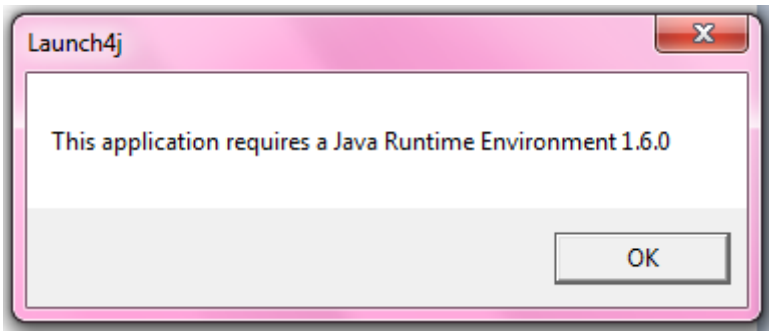
A szakdolgozatom legfontosabb feladata egy olyan oktató szoftver létrehozása volt, amely lehetővé teszi a felhasználó által kiválasztott, megrajzolt képen a kiválasztott síkbeli transzformáció reprezentációját.

2.2. Felhasználói kézikönyv

2.2.1. Program futtatása

Mivel a program platformfüggetlen, ezért bármelyik operációs rendszeren futtatható. Az egyetlen követelmény, hogy a számítógépen, amelyen a szoftvert futtatni szeretnénk, legyen Java RunTime Enviroment telepítve.

Ha az alábbi hibaüzenethez hasonló jelenik meg a képernyőn, szükség van a fentebb említett környezet telepítésére, amely megtalálható a mellékelt CD-n. A JRE telepítésével kapcsolatos információkat elérhetjük a http://java.com/en/download/help/download_options.xml weboldalon.



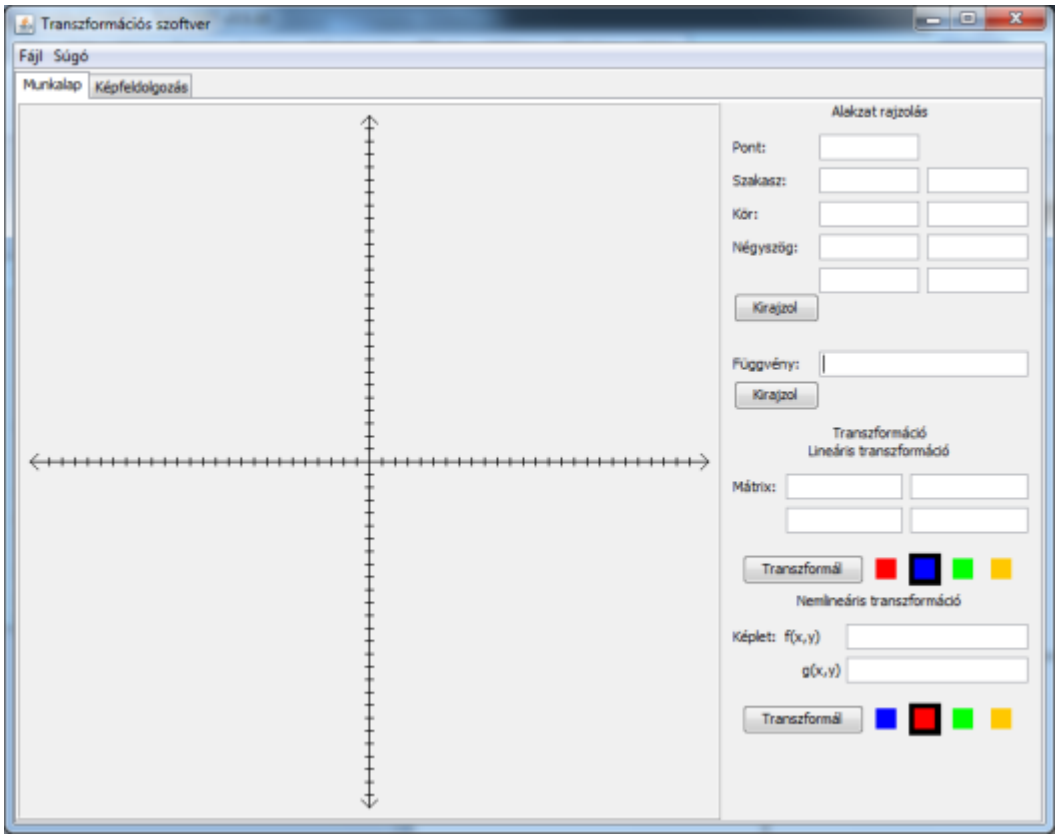
2.1. ábra: JRE hiányában fellépő hibaüzenet

Futtatás Windowson: A programot nem kell telepíteni, a CD-n található transzformaciosprogram.exe fájlra kell kétszer kattintani, és máris dolgozhatunk.

Linux rendszereken: Első körben nyitunk egy terminál ablakot, majd abba a könyvtárba navigálunk, amely tartalmazza a transzformaciosprogram.jar fájlt. Ezután kiadjuk a java –jar transzformaciosprogram.jar parancsot.

2.2.2. A program használata

A program indításakor az alábbi ablak jelenik meg:



2.2. ábra: Főképernyő

Láthatjuk, hogy a program két fő részből áll, egy Munkalap, illetve egy Képfeldogozás nézetből.

A Munkalap nézet lesz segítségünkre, ha koordináta-rendszerben szeretnénk dolgozni, a Képfeldogozás nézet pedig a képek transzformációja során lesz fontos számunkra.

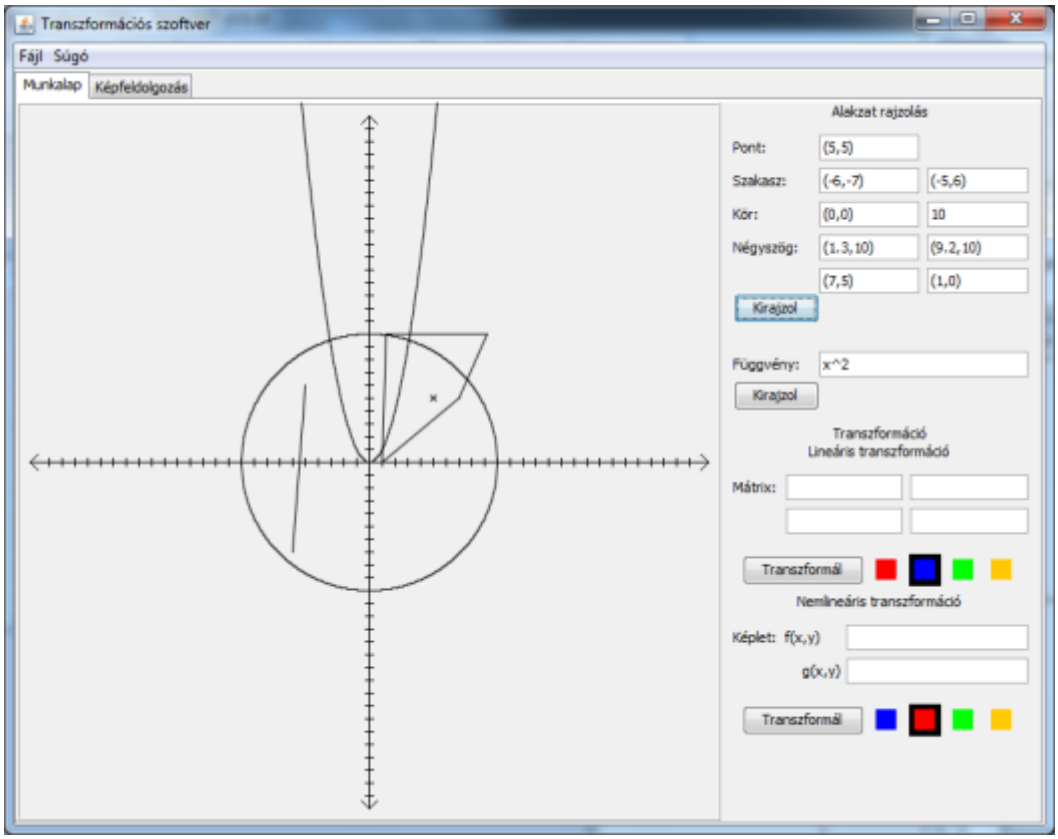
Elsőként a Munkalap nézetet mutatom be.

2.2.3. Munkalap nézet

Ez a nézet két fő részből áll. Bal oldalon található a koordináta-rendszer, melyben majd az alakzatokat ábrázoljuk, jobb oldalon pedig az ehhez tartozó eszköztár. Az eszköztár segítségével vihetünk fel alakzatokat a koordináta-rendszerbe, illetve transzformálhatjuk azokat.

Elsőként nézzük, milyen lehetőségeink vannak, ha rajzolni szeretnénk.

Az eszköztárból felvihetünk pontokat, szakaszokat, négyszögeket, illetve köröket is a koordináta- rendszerbe.



2.3.ábra: Alakzatok felrajzolása eszköztárból

Pont felrajzolása

A pontok felrajzolásához meg kell adni a pont koordinátáit a következő alakban: (x, y), ahol x és y valamilyen valós szám. A koordináták megadása során figyeljünk arra, hogy tört számok esetén tizedesvessző helyett pontot használjunk, különben a program nem tudja értelmezni a megadott pontot. Az x és y

koordináták közé elválasztó karakterként ’,’-t tegyünk. Ezt fontos hangsúlyozni a félreértések elkerülése végett.
Ha mégis rosszul adtunk meg adatot, a program jelzi a hibás adatbevitelt.

Szakasz felrajzolása

A szakasz megadása két végpontjának megjelölésével történik. A végpontokat a már az előzőekben megbeszélt módon adhatjuk meg.

Négyszög felrajzolása

A négyszöget szintén négy pontja segítségével tudjuk ábrázolni. A csúcsokat szintén a fentiekben leírtak alapján adhatjuk meg. Figyeljünk arra, hogy a csúcsok megjelölésének iránya az óramutató járásával megegyező (matematikailag negatív) irányban történik, vagyis a helyes kirajzolás érdekében érdemes a pontokat sorrendhelyesen megadni a bal felső saroktól kezdődően.

Kör felrajzolása

Az eszköztárból megadhatunk egy kört is a középpontja és a sugara segítségével. A középpont megadása az előzőekben leírt módon történik, a sugarat pedig egy valós számként írhatjuk fel. (Itt is figyeljünk, hogy tizedesvessző helyett tizedespontot használjunk.)

Az alakzatok felrajzolását tehát könnyedén megtehetjük a felhasználói felületen.

Függvény megjelenítése

Az eszköztárból lehetőségünk van függvények rajzolására is, mely sok esetben nagyon hasznos lehet. A program az alábbi műveletek, függvények szemléltetésére képes:

- abszolút érték: abs(x)
- szinusz: sin(x)
- koszinusz: cos(x)
- természetes alapú logaritmus: log(x)
- e^x érték: exp(x)

- tangens: tan(*x*)
- arkusz szinusz: asin(*x*)
- arkusz koszinusz: acos(*x*)
- szorzás, osztás, összeadás, kivonás, hatvány: *,/,+,-
- négyzetgyök: sqrt(*x*)
- köbgyök: cbrt(*x*)

Nagyon kell figyelnünk arra, hogy a függvényt zárójelhelyesen adjuk meg.

Munkalap megnyitása fájlból

Az eszköztár segítségével történő adatfelvitelen kívül más megoldás is van az alakzatok felrajzolására. Lehetőségünk van az adatok fájlból történő beolvasására is.

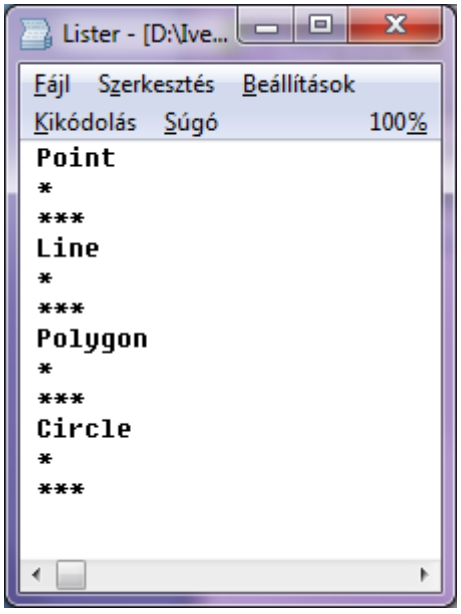
Ha fájlból szeretnénk Munkalapot megnyitni, a Fájlmenü/Megnyitás/Munkalap megnyitása menüpont segítségével tehetjük meg.

Olvashatunk be már meglévő fájlokat, melyeket az Adatfájlok könyvtárban találunk, illetve létrehozhatunk mi is saját adatfájlt, amely igazán megkönnyítheti a munkánkat.

Adatfájlok szerkezete

A fájl létrehozásakor figyelni kell az adatfájl rögzített vázára, ezért erősen javasolt a sablonok használata. Az Adatfájlok könyvtárban találhatunk ilyen sablon fájlokat, melyek igazán megkönnyítik a munkánkat. Ilyenkor csak annyi a dolgunk, hogy megfelelően töltjük ki a rendelkezésre álló fájlunkat.

A pontok megadása során figyelni kell, hogy minden felvinni kívánt pontot új sorba írjunk. A pontokat egy új sorba írt ’*’ karakter választ el egymástól. A pontok felvitelének befejeztét szintén új sorba írt „***” karaktersorozat jelzi. A pontokat ebben az esetben is a koordinátáik segítségével adhatjuk meg (x,y) alakban, ahol x és y valamilyen valós szám. Itt is fontos, hogy az elválasztó karakter ’,’ legyen, illetve a tizedesvessző helyett tizedespontot használjunk.



2.4.ábra: Sablon fájl

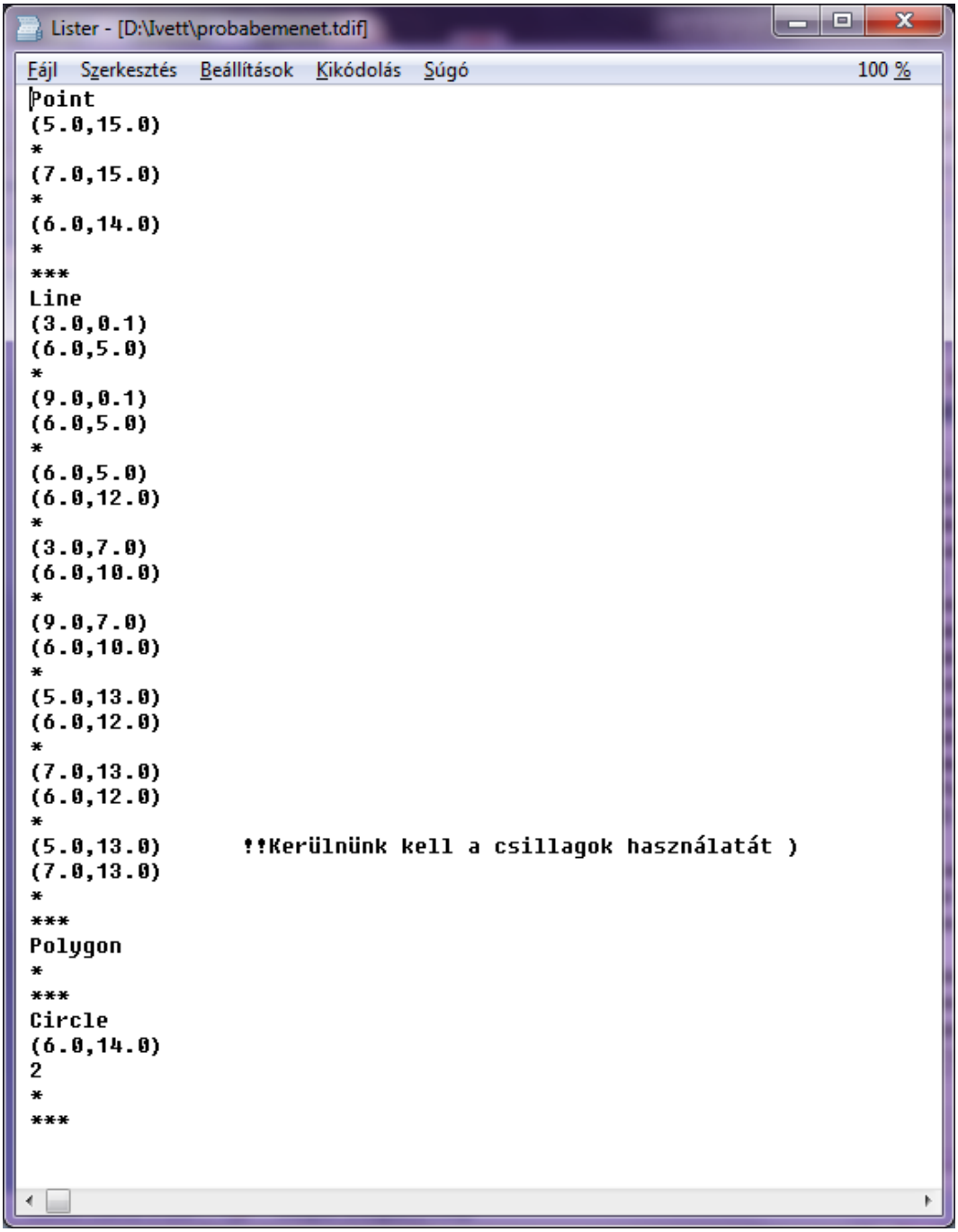
Ha nem sablonfájlban dolgozunk, mindenképpen figyelnünk kell a pontos szerkezetre, különben a programunk nem fogja tudni értelmezni a fájlunkat.
Sablon létrehozása egyébként egyszerűen kivitelezhető egy üres Munkalap elmentésével.

Ha szakaszt szeretnénk felvinni, a szakasz két végpontját kell megadnunk. Először megadjuk az egyik pontot (x_1,y_1) alakban, majd új sorban megadjuk a másik pont koordinátáit (x_2,y_2) alakban, majd új sorba írunk egy ’*’ karaktert, ezzel jelezzük az adott szakasz megadásának végét. Ha nem kívánunk több szakaszt felvinni, akkor egy új sorba írt „***” karaktersorozattal zárjuk a sort.

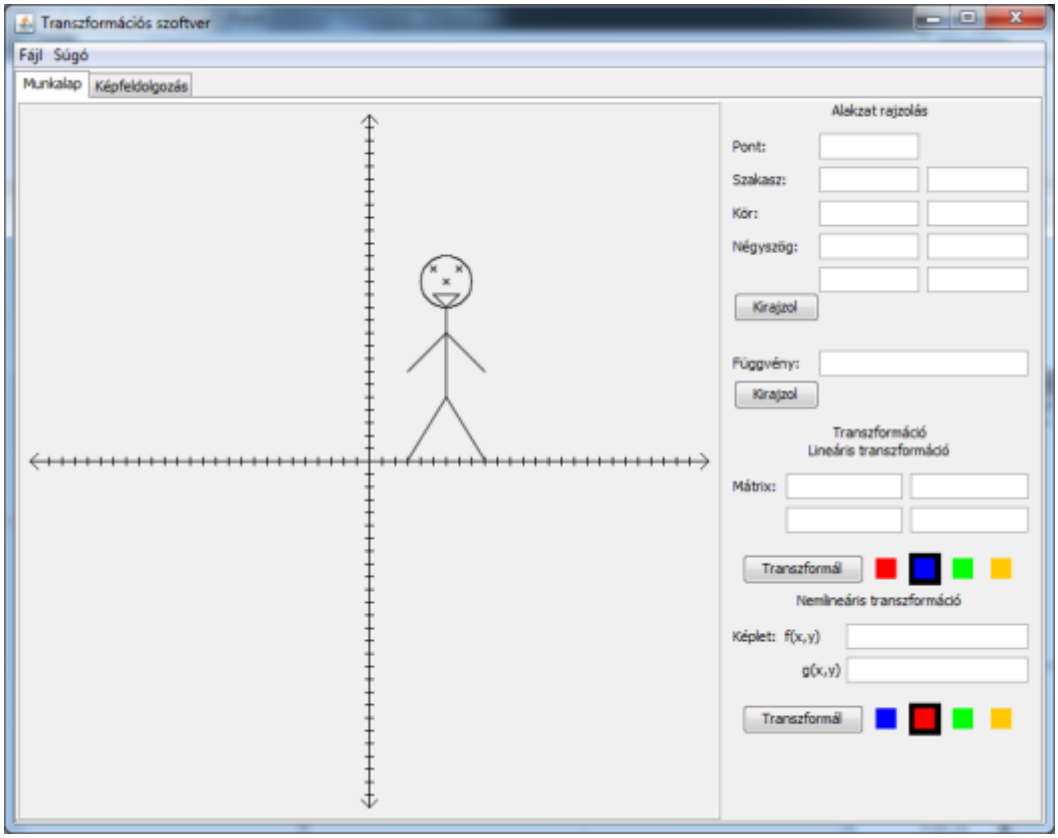
A poligon felvitele igazából csak a felhasználónak nyújt segítséget, hiszen szakaszok felvitelével is megadhatunk egy sokszöget, így viszont csak a sokszög csúcsait kell felsorolnunk egymás alá új sorba sorrendhelyesen. A csúcsok leírása az előzőekben leírt formában történik, vagyis (x,y) formában. Az utolsó pont felvitele után ’*’ karakterrel jeleznünk kell a poligon felvitelének végét. Ez mindenképpen fontos, hiszen poligon esetében előre nem tudhatjuk, hogy hány pontja lesz, ezt a felhasználó maga döntheti el. Ha nem kívánunk több poligont felrajzolni, új sorba írt „***” sztring zárja a sort.

Az adatfájl segítségével körök is felvihetők, ilyenkor a kör középpontját kell megadni (x,y) formában, majd új sorban a kör sugarát, mely egy valós, nem

negatív szám. Ezután egy ’*’ karakter jelzi új sorban, hogy végeztünk egy kör felvitelével. Ha nem kívánunk több kört ábrázolni, ezt az új sorba írt „***” sztringgel jelöljük.



2.5.ábra: Példa egy adatfájlra



2.6.ábra: A 2.5. ábrán lévő adatfájl beolvasásának eredménye

A munkánk megkönnyítése érdekében lehetőségünk van az adatfájlokba megjegyzéseket írni, melyeket a pontok megadása után tehetünk meg. Figyeljünk oda, hogy semmiképpen sem az alakzatok angol neve mellé tegyük megjegyzéseinket, mert így a program nem az elvárásainknak megfelelően rajzolja ki az ábrákat. Fontos ezen kívül, hogy a megjegyzéseinkben ne használjunk ’*’ karaktert, mert ez a karakter a pontok egymástól való elkülönítésére szolgál.

Az adatfájl formátumát tekintve mindenképpen egy szöveges fájl, amelyet könnyedén szerkeszthetünk szinte bármilyen szövegszerkesztővel, a kiterjesztés viszont minden esetben .tdif kell hogy legyen, mivel a Munkalapok megnyitása menüpontban a program csak az ilyen kiterjesztésű fájlokat listázza ki, és ezekből enged választani.

Az adatfájlból való beolvasás nagy segítséget nyújthat, ha egyszerre több alakzat felrajzolását szeretnénk elérni, de sokkal kevesebb figyelmetlenséget enged meg. Ha az adatfájl formája nem felel meg a fentebb leírtaknak, a program

nem tudja értelmezni azt, így vagy nem rajzol ki semmit, vagy nem az általunk kívánt alakzatot jeleníti meg, így érdemes odafigyelni a szabályok betartására.

A Munkalap nézetre tehát könnyedén felvihetünk csaknem bármilyen 2-dimenziós ábrát.

A program legfőbb célja azonban nem az alakzatok megrajzolása, hanem azok transzformálása.

Munkalap lineáris transzformációja

A Munkalap transzformálásának két módja van. Ha lineáris transzformációt szeretnénk végrehajtani az alakzatainkon, meg kell adnunk a transzformáció mátrixát.

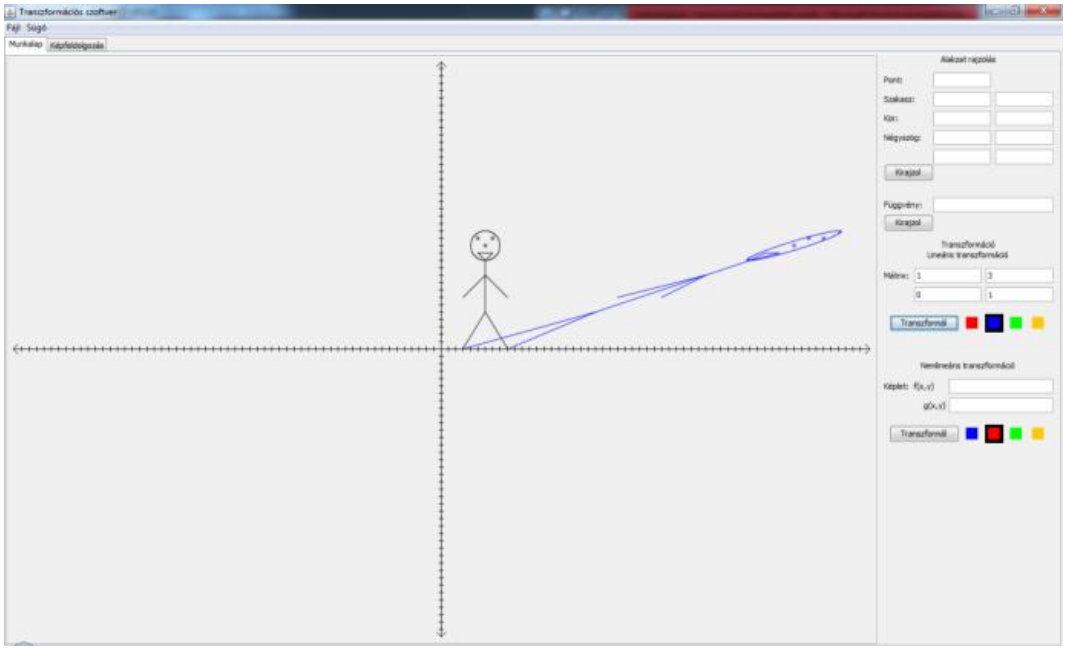
A mátrixot

$$\begin{bmatrix} x11 & x12 \\ x21 & x22 \end{bmatrix}$$

alakban adhatjuk meg, ahol *x11*, *x12*, *x21*, *x22* valós számok. Itt is oda kell figyelnünk, hogy ne használjunk zárójeleket, illetve tört szám esetén tizedesvessző helyett pontot használjunk.

A transzformált kép mindig az eredeti képtől eltérő színnel jelenik meg a koordináta-rendszerben. Ha több transzformációt is szeretnénk végrehajtani, az átláthatóság érdekében átválthatunk másik színre is a „Transzformál” gomb melletti színes négyzetek segítségével.

Az alábbi ábrán egy példát mutatunk a lineáris transzformációra. Esetünkben az x tengelyű 3 paraméterű nyírás eredménye látható.



2.7.ábra: Példa lineáris transzformációra

Munkalap nemlineáris transzformációja

A nemlineáris transzformáció másként történik. Ekkor két kétváltozós függvényt adhatunk meg, melyek az új *x'* és *y'* koordináták helyét adják meg. Figyeljünk, hogy csak az alábbi listában megadott függvényeket használjuk, és minden nyitó zárójelnek legyen meg a záró párja. Negatív számok, kifejezések esetén mindenképpen tegyük zárójelbe azokat, akkor is, ha a függvény elején szerepelnek. Ez azért szükséges, hogy jól elkülöníthető legyen a negatív jel, és a kivonás művelet.

A felhasználható matematikai kifejezések listája:

abs(x): x abszolút értéke

acos(x): x arkusz koszinusz értéke

asin(x): x arkusz szinusz értéke

atan(x): x arkusz tangens értéke

cbrt(x): x köbgyöke

cos(x): x koszinusza

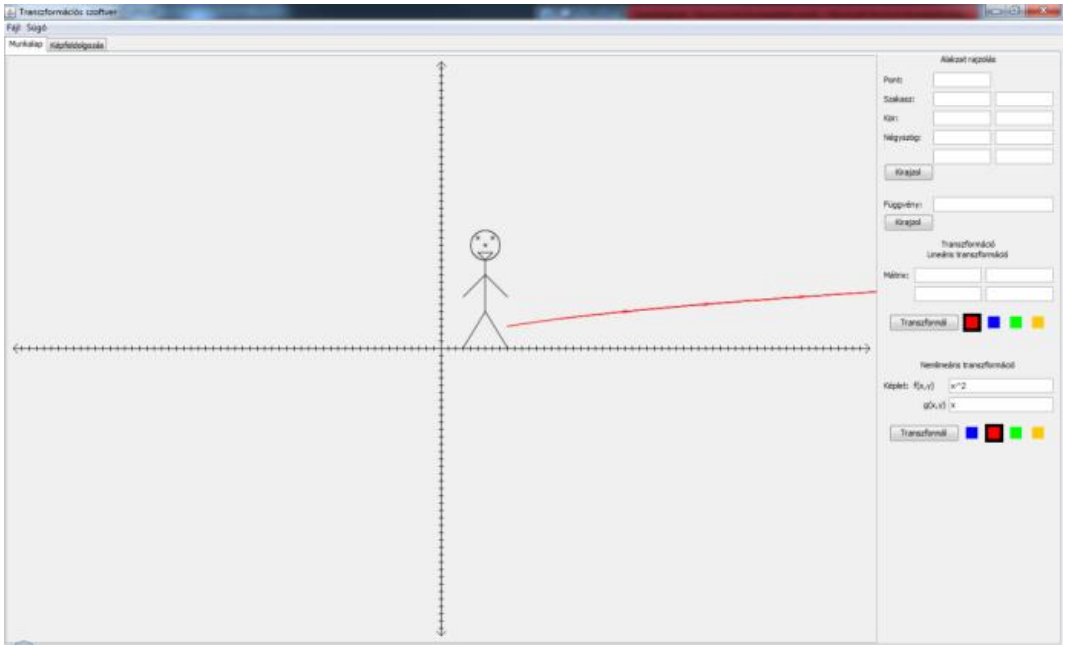
exp(x): e^x érték

log(x): x természetes logaritmusa

sqrt(x): x négyzetgyöke

Felhasználhatók még az alábbi matematikai operátorok: +, -, /, *, ^, valamint nyitó ’(’, illetve záró ’)’ zárójelek.

A nemlineáris transzformáció esetében a transzformált kép színe szintén eltér az eredetitől. Ha több transzformációt is szeretnénk végrehajtani, az átláthatóság érdekében átválthatunk másik színre is a „Transzformál” gomb melletti színes négyzetek segítségével.



2.8.ábra: Példa nemlineáris transzformációra ($x' = x^2, y' = x$)

Új Munkalap

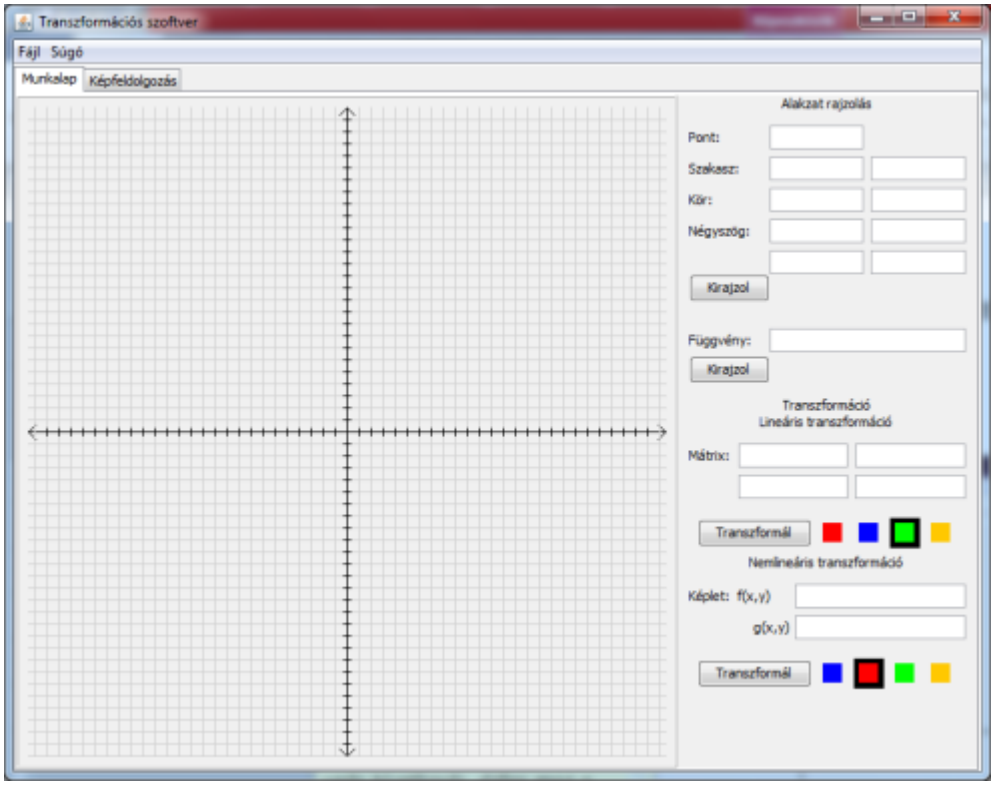
A Munkalap nézet esetében lehetőségünk van egy teljesen új Munkalapot kezdeni, mely a Fájl/Új Munkalap menüpont segítségével lehetséges. Ilyenkor a program felkínálja a régi Munkalap mentésének lehetőségét.

Munkalap mentése

Ha szeretnénk később is dolgozni az újonnan felvitt alakzatokkal, menthetjük a Munkalapunkat. A Munkalapot a program egy szöveges fájlba menti, melynek kiterjesztése .tdif, ezen kívül készít egy képet a koordináta-rendszerről, amelyet az adatfájllal azonos néven ment el ugyanabba a mappába, valamint készít egy logfájlt, amiben feljegyzi, hogy melyik adatfájlhoz melyik kép tartozik.

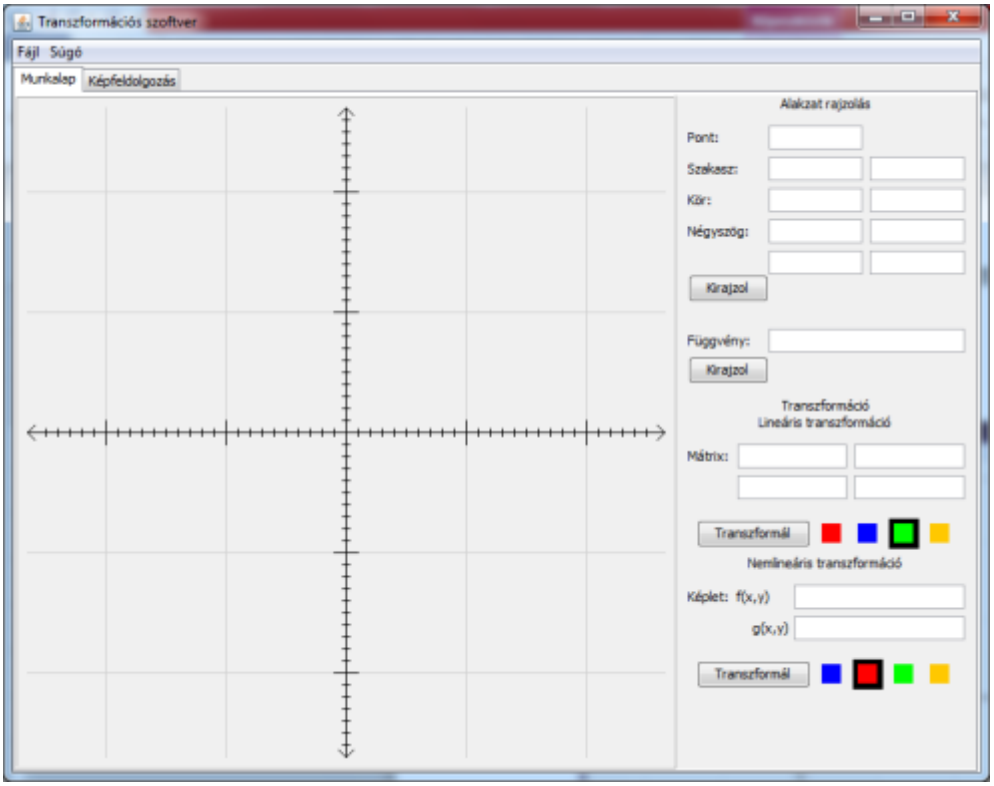
Munkalap tulajdonságai

A koordináta-rendszer néhány tulajdonságát változtatni tudjuk. Lehetőségünk van például rácsvonalak berajzolására, amely könnyebbé teszi az eligazodást. Ezt a funkciót a úgy érhetjük el, hogy jobb egérgombbal a koordináta-rendszerre klikkelünk, majd a felugró menüből kiválasztjuk a Rácsozás be menüpontot. Ugyanitt van lehetőségünk a funkció kikapcsolására is.



2.9.ábra: Munkalap rácsozása

Ezen kívül változtathatjuk a koordináta-rendszerünk beosztását is az előzőekben leírt módon. Szintén a koordináta-rendszerre kell kattintanunk az egér jobb gombjának megnyomásával, majd válasszuk ki a menüből a nagyít/ kicsinyít menüpontot.



2.10. ábra: Munkalap nagyítása

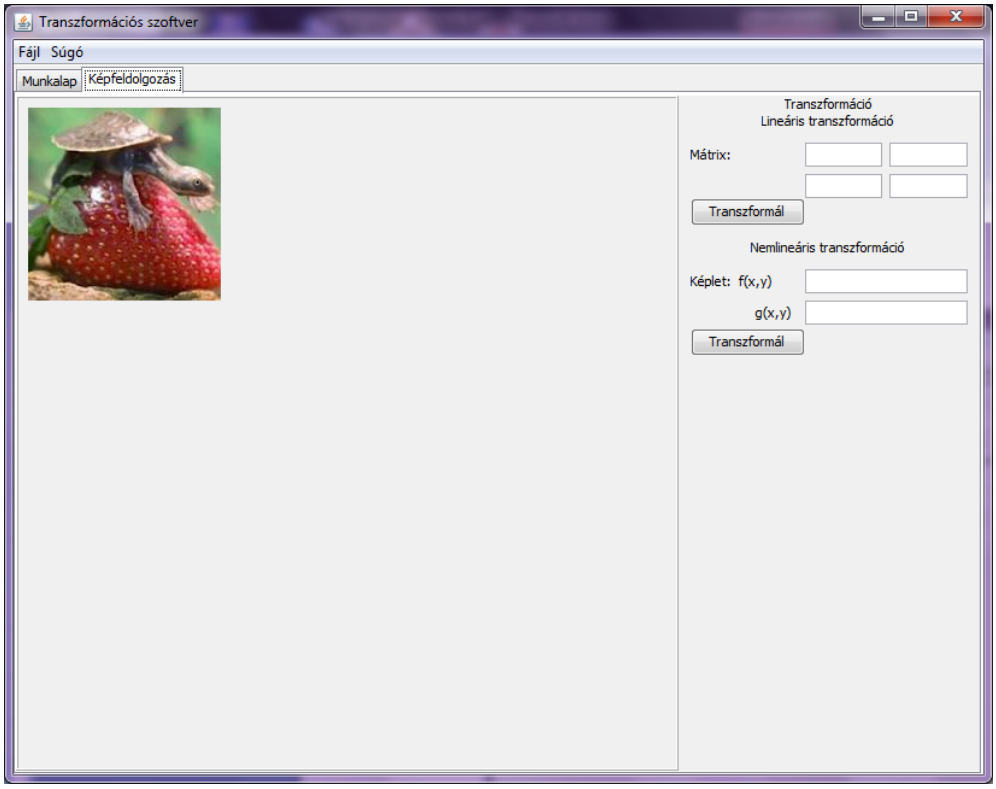
2.2.4. Képfeldolgozó nézet

A másik fő nézet a Képfeldolgozó nézet, mely a képek transzformációját hivatott bemutatni.

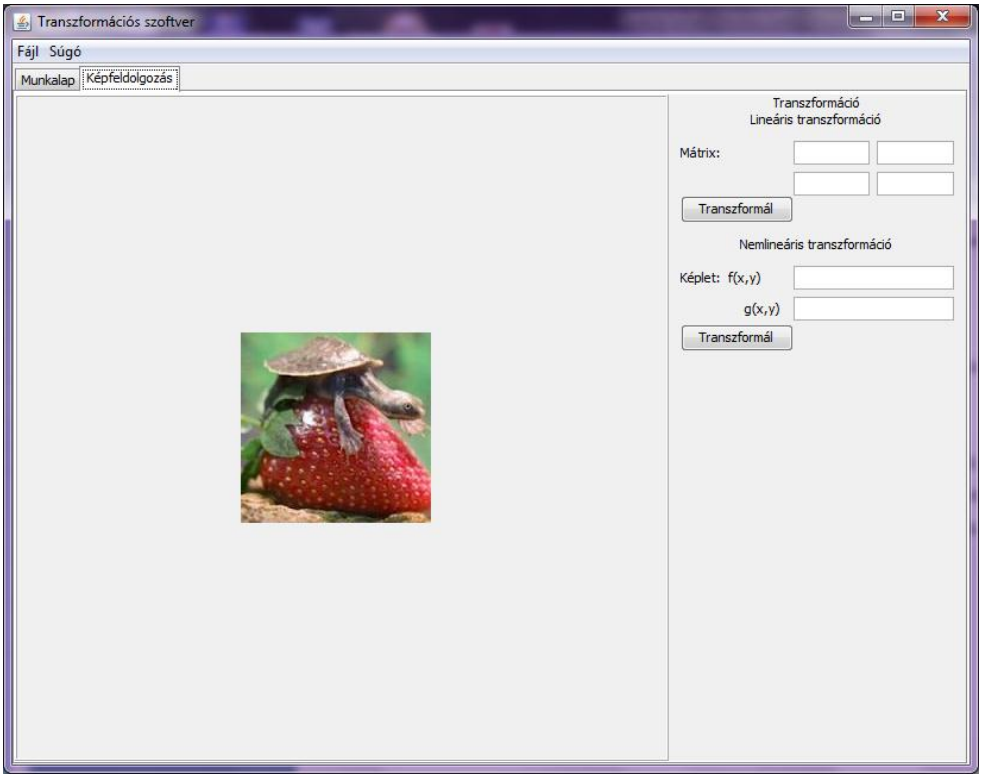
Ahhoz, hogy új képet tudjunk megnyitni annak transzformálása céljából, a Fájl/Megnyitás/Kép megnyitása menüpontra kell kattintanunk. Ekkor megjelenik egy fájlválasztó ablak, ahol kiválaszthatunk bármilyen png, jpg, bmp és gif kiterjesztésű fájlt.

A kép transzformációja során sok pont egyenkénti transzformációjára van szükség, ezért ez a művelet időigényes. A kiválasztott képet éppen ezért kicsinyítenünk kell, amelyet a program helyettünk elvégez.

A kép pozícióját változtathatjuk, ha jobb egérgomb segítségével ráklikkelünk a képre, majd az így felbukkanó menüben kiválasztjuk a kívánt elhelyezkedést. Fontos megjegyezni, hogy a Képfeldolgozó nézetben az origó a bal felső sarokban van.



2.11. ábra: Kép megnyitása

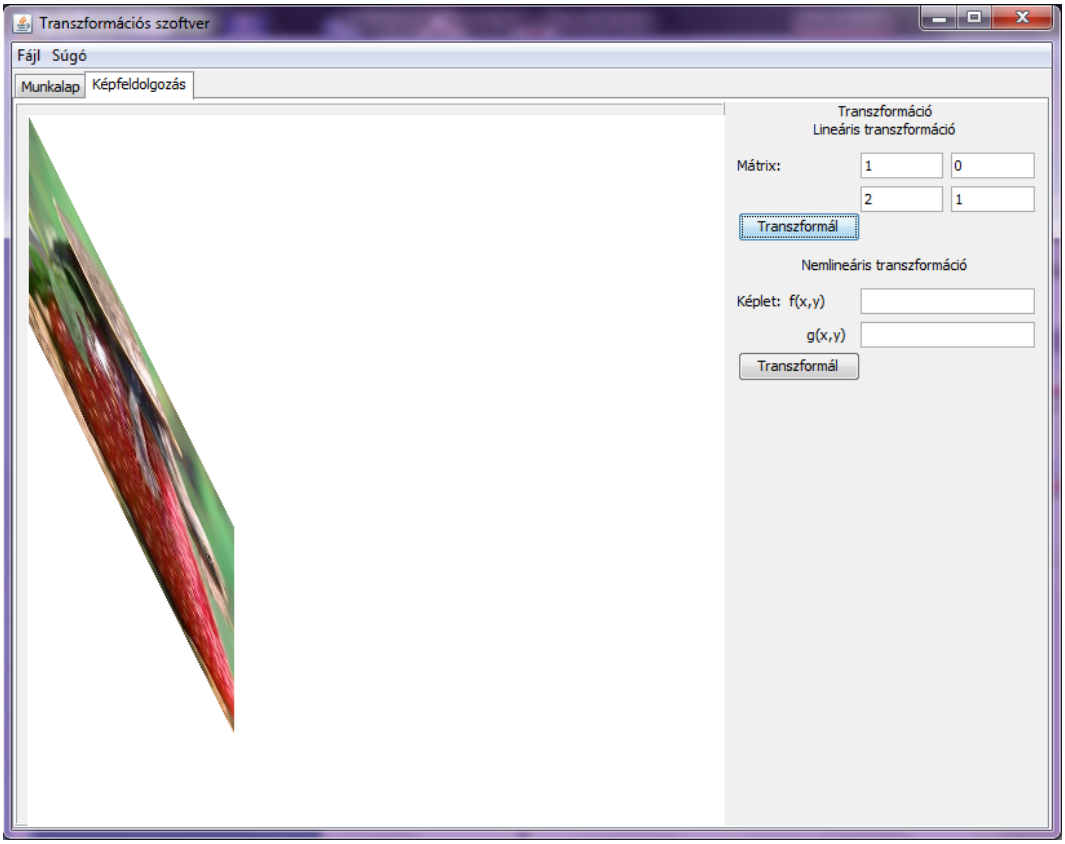


2.12. ábra: Kép mozgatása

Ha már megnyitottunk egy képet, következhet annak transzformációja.

Kép lineáris transzformációja

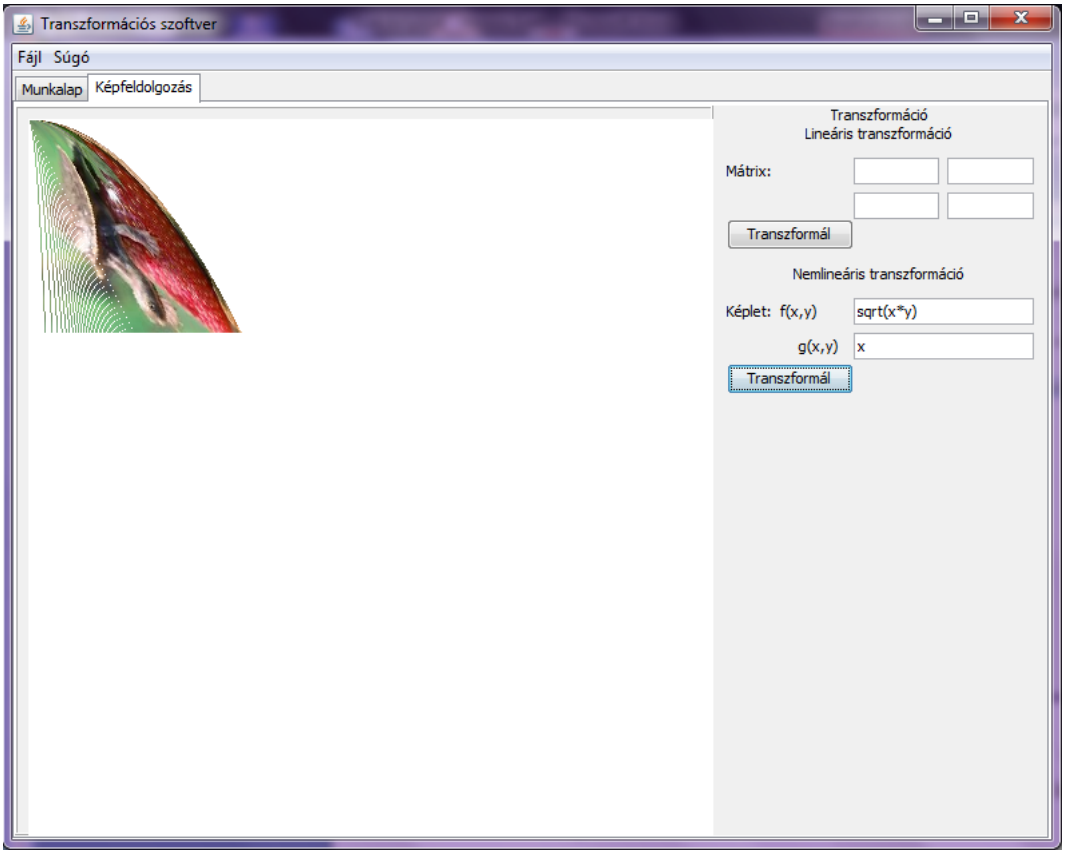
A lineáris transzformáció ugyanúgy történik, mint az alakzatok transzformálása esetében, vagyis transzformációs mátrix segítségével. Szintén négy valós számot kell megadni, ahol a tizedesvesszők helyett pontot használhatunk, és fontos, hogy ne írjunk zárójeleket a rubrikákba.



2.13. ábra: Kép lineáris transzformációja

Kép nemlineáris transzformációja

A nemlineáris transzformáció szintén hasonlóan működik, mint a Munkalap esetében.



2.14. ábra: Kép nemlineáris transzformációja

Két függvény megadására van szükség, mely az új x' és y' koordinátákat határozza meg. Figyeljünk, hogy csak olyan függvényeket használjunk, amelyek szerepelnek az előzőekben felsoroltak között.

Ha szeretnénk visszaállítani az eredeti képünket, a Fájl/Visszavonás gomb megnyomásával ezt megtehetjük.

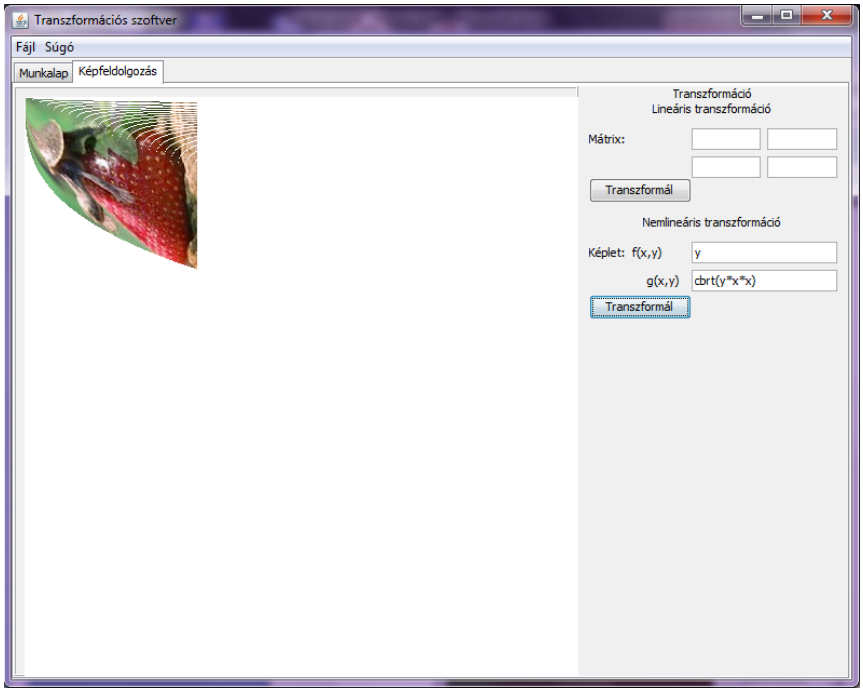
Kép mentése

A kép mentése a Fájl/Mentés menüpont alatt történhet, ahol a transzformált kép mentésére van lehetőségünk.

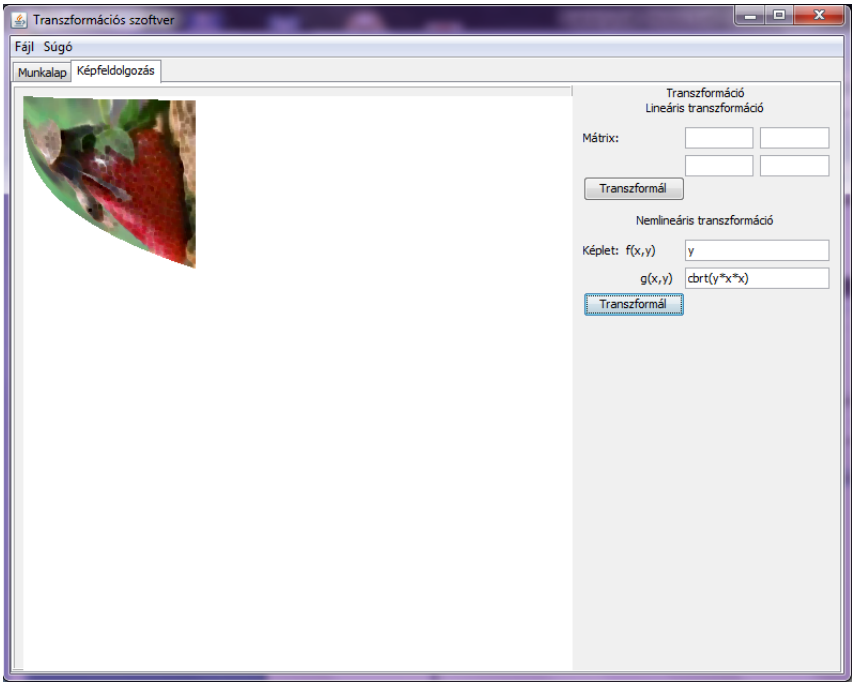
Kép javítása

Mivel a különböző transzformációk során minden pixelnek új helyet keresünk, ezért előfordulhat, hogy a transzformált pixelek messze kerülnek egymástól, ezért a transzformált képben nagy üres területek jelenhetnek meg. Szemléltetés szempontjából ez lehet előnyös, hiszen szépen látszik, hogy melyik pixelnek melyik a képe, viszont előfordulhat, hogy a kép teljesen felismerhetetlenné válik.

A transzformált kép javítására lehetőségünk nyílik, ha rákattintunk a jobb egérgomb segítségével a képre, és a felugró menüből kiválasztjuk a Kép javítása menüpontot. A kép többszöri javításával jobban felismerhető képet kaphatunk, de előfordulhat az is, hogy a kép teljesen szétfolyik, ezért ez csak bizonyos esetekben javasolt, mivel a visszavonására nincs lehetőség.



2.15. ábra: Transzformált kép javítás előtt



2.16. ábra: Transzformált kép javítás után

Súgó

Ha munkánk során valahol elakadnánk, és segítségre lenne szükségünk, a Súgó menüpontra kattintva hasznos információkat érhetünk el a program használatára vonatkozóan. Itt csak a keresendő címszót szükséges beírni, majd a Keresés gomb segítségével könnyedén böngészhetünk a leírásban.

2.3. A program megvalósításának folyamata

2.3.1. Tervezés

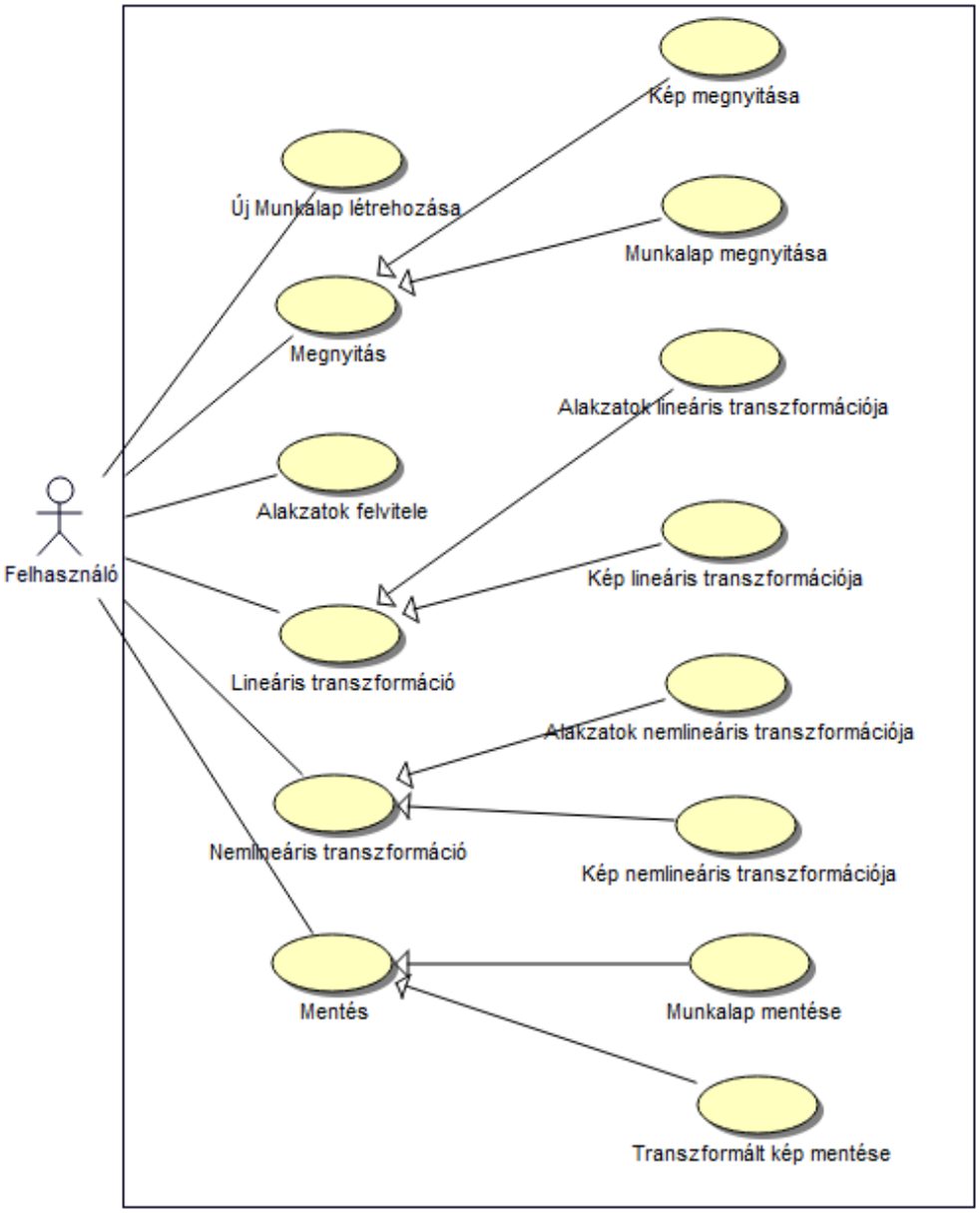
Olyan rendszer megalkotására volt szükség, amely segítséget nyújt a lineáris és nemlineáris transzformációk bemutatásában.

Első lépésként szükség volt a felhasználói igények felmérésére, a szükséges, beépítendő funkciók feltárására, majd ebből egy részletes modell megalkotására. A felhasználók igényei közé tartozott többek között egy olyan program létrehozása, mely képes adatfájlokkal dolgozni, koordináta-rendszeres megjelenítésre és képek transzformációjára.

Az első tervezési szakaszban tehát az alábbi funkciókat gyűjtöttem össze:

- Új Munkalap létrehozása: Egy üres koordináta-rendszer létrehozása.
- Alakzatok felvitele: Pontok, vonalak, négyszögek, körök, függvény felvitele
- Alakzatok lineáris transzformációja: Alakzatok transzformációja mátrix alapján
- Alakzatok nemlineáris transzformációja: Alakzatok transzformálása függvény alapján
- Munkalap mentése: Szerkesztett munkalap mentése
- Munkalap megnyitása: Már létező Munkalap beolvasása
- Kép megnyitása: Transzformálandó kép megnyitása
- Kép lineáris transzformációja: Kiválasztott kép lineáris transzformációja
- Kép nemlineáris transzformációja: Kiválasztott kép nemlineáris transzformációja
- Transzformált kép mentése: A lineárisan vagy nemlineárisan transzformált kép mentése

A funkciókból use case diagramot készítettem:



2.17. ábra: Use case diagram

2.3.1.1. Usecase leírások

Név: Új Munkalap létrehozása

Szereplők és érdekeik:

Felhasználó: célja egy új, üres Munkalap létrehozása

Elsődleges aktor: Felhasználó

Előfeltétel: -

Standard lefutás (Szcenárió):

1. A felhasználó kiválasztja az Új munkalap létrehozása menüpontot.
2. A rendszer egy üres Munkalapot jelenít meg.

Kivételes esetek (Alternatív Szcenárió):

1. a, Az előző módosított Munkalap még nincs elmentve.
    1. A program tájékoztatja a felhasználót a mentésre váró Munkalapról
    2. A felhasználó elmenti az előző Munkalapot, vagy elutasítja a mentést.

Speciális követelmények: -

Sikeres végrehajtás: A rendszer üres koordináta-rendszert jelenít meg számunkra.

Technológiai/adatformátum követelmények: -

Gyakoriság: gyakran

Név: Alakzatok felvitele

Szereplők és érdekeik:

Felhasználó: célja alakzatok felrajzolása a koordináta-rendszerbe

Elsődleges aktor: Felhasználó

Előfeltétel: -

Standard lefutás (Szcenárió):

1. A felhasználó megadja a felvinni kívánt alakzatok adatait.
2. A rendszer felrajzolja, és elhelyezi az alakzatokat a koordináta-rendszerben.

Kivételes esetek (Alternatív Szcenárió):

1 a, A felhasználó helytelenül adja meg az adatokat.

1.A rendszer tájékoztatja a felhasználót a hibáról.

Speciális követelmények: -

Sikeres végrehajtás: A rendszer felrajzolja az alakzatokat a koordináta-rendszerbe.

Technológiai/adatformátum követelmények: -

Gyakoriság: gyakran

Név: Munkalap mentése

Szereplők és érdekeik:

Felhasználó: célja a szerkesztett Munkalap elmentése a számítógépen

Elsődleges aktor: Felhasználó

Előfeltétel: -

Standard lefutás (Szcenárió):

1. A felhasználó kiválasztja a Munkalap mentése menüpontot.
2. A rendszer bekéri a mentendő Munkalap nevét.
3. A felhasználó megadja a kívánt nevet.
4. A rendszer elmenti a Munkalapot a megadott néven.

Kivételes esetek (Alternatív Szcenárió): -

Speciális követelmények: -

Sikeres végrehajtás: A Munkalap mentésre kerül a rendszerben.

Technológiai/adatformátum követelmények: -

Gyakoriság: gyakran

Név: Munkalap megnyitása

Szereplők és érdekeik:

Felhasználó: célja egy már létező Munkalap transzformációja, szerkesztése

Elsődleges aktor: Felhasználó

Előfeltétel: -

Standard lefutás (Szcenárió):

1. A felhasználó kiválasztja a Munkalap megnyitása menüpontot.
2. A rendszer kilistázza a Munkalap fájlokat.
3. A felhasználó kiválasztja a kívánt Munkalapot.
4. A rendszer megjeleníti az adatfájlból beolvasott alakzatokat.

Kivételes esetek (Alternatív Szcenárió):

2.a Nincs kilistázandó fájl.

Speciális követelmények: -

Sikeres végrehajtás: A Munkalap tartalma megjelent a koordináta-rendszeren.

Technológiai/adatformátum követelmények: -

Gyakoriság: ritkán

Név: Kép megnyitása

Szereplők és érdekeik:

Felhasználó: célja a transzformálandó kép megnyitása

Elsődleges aktor: Felhasználó

Előfeltétel: -

Standard lefutás (Szcenárió):

1. A felhasználó kiválasztja a Kép megnyitása menüpontot.
2. A rendszer kilistázza a kép fájlokat.
3. A felhasználó kiválasztja a kívánt képet.
4. A rendszer megjeleníti a kiválasztott képet.

Kivételes esetek (Alternatív Szcenárió):

2.a Nincs kilistázandó fájl.

Speciális követelmények: -

Sikeres végrehajtás: A kiválasztott kép megjelent a képernyőn.

Technológiai/adatformátum követelmények: -

Gyakoriság: ritkán

Név: Transzformált kép mentése

Szereplők és érdekeik:

Felhasználó: célja a transzformált kép elmentése

Elsődleges aktor: Felhasználó

Előfeltétel: Létezik transzformált kép

Standard lefutás (Szcenárió):

1. A felhasználó kiválasztja a Kép mentése menüpontot.
2. A rendszer bekéri a menteni kívánt kép nevét, és helyét a számítógépen.
3. A felhasználó megadja a kép nevét és helyét.
4. A rendszer elmenti a képet.

Kivételes esetek (Alternatív Szcenárió): -

Speciális követelmények: -

Sikeres végrehajtás: A transzformált kép elmentésre került a rendszerben.

Technológiai/adatformátum követelmények: -

Gyakoriság: ritkán

Név: Alakzatok lineáris transzformációja

Szereplők és érdekeik:

Felhasználó: célja a koordináta-rendszeren lévő alakzatok lineáris transzformációja

Elsődleges aktor: Felhasználó

Előfeltétel: Van felrajzolt alakzat az adott Munkalapon.

Standard lefutás (Szcenárió):

1. A felhasználó kiválasztja a lineáris transzformáció funkciót.
2. A rendszer bekéri a transzformációs mátrixot.
3. A felhasználó megadja a transzformáció mátrixát.
4. A rendszer megjeleníti a transzformált alakzatokat a koordináta-rendszeren.

Kivételes esetek (Alternatív Szcenárió):

3.a A felhasználó nem a megfelelő módon adja meg a mátrixot.

1. A rendszer tájékoztatja a felhasználót a hibás adatbevitelről, majd újra kéri a mátrixot.

4.a A Munkalapon nincs transzformálható alakzat.

1. A rendszer tájékoztatja a felhasználót a transzformálandó alakzatok hiányáról.

Speciális követelmények: -

Sikeres végrehajtás: A transzformált alakzatok megjelentek a koordináta-rendszeren.

Technológiai/adatformátum követelmények: -

Gyakoriság: ritkán

Név: Alakzatok nemlineáris transzformációja

Szereplők és érdekeik:

Felhasználó: célja a Munkalapon lévő alakzatok nemlineáris transzformáltjának megjelenítése a koordináta-rendszerben.

Elsődleges aktor: Felhasználó

Előfeltétel: Létezik felvitt alakzat a Munkalapon

Standard lefutás (Szcenárió):

1. A felhasználó kiválasztja a Nemlineáris transzformáció funkciót.
2. A rendszer bekéri az x koordináta transzformációjához, és az y koordináta transzformációjához szükséges függvényeket.
3. A felhasználó megadja a szükséges függvényeket.
4. A rendszer megjeleníti a függvények által meghatározott képet a koordináta-rendszerben.

Kivételes esetek (Alternatív Szcenárió):

3.a A felhasználó helytelenül adja meg a kért függvényeket.

1. A program tájékoztatja a felhasználót a helytelen működésről, és újra kéri a függvényeket.

Speciális követelmények: -

Sikeres végrehajtás: A Munkalap tartalma megjelent a koordináta-rendszeren.

Technológiai/adatformátum követelmények: -

Gyakoriság: ritkán

Név: Kép lineáris transzformációja

Szereplők és érdekeik:

Felhasználó: célja a kiválasztott kép lineáris transzformáltjának megjelenítése.

Elsődleges aktor: Felhasználó

Előfeltétel: -

Standard lefutás (Szcenárió):

1. A felhasználó kiválasztja a Kép lineáris transzformálása funkciót.
2. A rendszer bekéri a lineáris transzformáció mátrixát.
3. A felhasználó megadja a mátrixot.
4. A rendszer megjeleníti a lineárisan transzformált képet.

Kivételes esetek (Alternatív Szcenárió): -

Speciális követelmények: -

Sikeres végrehajtás: A transzformált kép megjelenik a képernyőn.

Technológiai/adatformátum követelmények: -

Gyakoriság: ritkán

Név: Kép nemlineáris transzformációja

Szereplők és érdekeik:

Felhasználó: célja a kiválasztott kép nemlineáris transzformációjának megjelenítése

Elsődleges aktor: Felhasználó

Előfeltétel: -

Standard lefutás (Szcenárió):

1. A felhasználó kiválasztja a Kép nemlineáris transzformációja funkciót.
2. A rendszer bekéri a transzformációs függvényeket.
3. A felhasználó megadja a bekért függvényeket.
4. A rendszer megjeleníti a függvények alapján transzformált képet.

Kivételes esetek (Alternatív Szcenárió): -

Speciális követelmények: -

Sikeres végrehajtás: A transzformált kép megjelenik a képernyőn.

Technológiai/adatformátum követelmények: -

Gyakoriság: ritkán

A funkciók összegyűjtése után a technikai döntések következtek.

2.3.1.2. A Java nyelv bemutatása

A program tervezése során fontos volt a megfelelő programozási nyelv kiválasztása is. Mivel mindenképpen egy olyan rendszer létrehozása volt a cél, amely platformfüggetlen, hordozható, és grafikus felülettel rendelkezik, ezért a választásom a Java nyelvre esett.

A Java programozási nyelv egy általános célú, több szálú programozást támogató, osztály alapú objektum-orientált nyelv.

A Java platform lehetőséget nyújt arra, hogy a program újrafordítás nélkül bármelyik operációs rendszeren futtatható legyen.

A fordítás során a forráskódból platformfüggetlen bájt kód lesz, amelyet a virtuális gép (Java Virtual Machine - JVM) értelmez, és fordít gépi kódra. A JVM felelős a futtatásért is. Ennek eredményeképpen egy Javaban írt program mindig

lassabban fut egy natív nyelvű programnál, illetve több memóriát használ. A lefordított program bármilyen számítógép architektúrán és operációs rendszeren futtatható, amelyre elérhető a virtuális gép.

2.3.1.3. Fejlesztő környezet

A programozási nyelv kiválasztása után a fejlesztői környezet meghatározása volt a következő lépés. Hosszas keresgélés és információgyűjtés után az Eclipse IDE tűnt a legmegfelelőbb választásnak. Ennek a fejlesztői környezetnek nagy előnye, hogy rengeteg kiegészítés tartozik hozzá, ezért jelentősen megkönnyíti a programozó munkáját. Mivel grafikus interfész fejlesztése volt a cél, ezért szükségem volt egy Java GUI Designer-re is, amely a WindowBuilder Eclipse plug-in formájában adott volt.

2.3.2 A program megvalósítása

A programozási nyelv és a megfelelő fejlesztő környezet kiválasztása után következhetett a megvalósítás.

A program létrehozása több iteráción keresztül folyt.

A fejlesztés iterációinak meghatározása:

0. iteráció:
   - Use case diagram létrehozása
   - Megfelelő eszközök kiválasztása, használatuk elsajátítása
1. iteráció
   - Kezdetleges grafikus interfész létrehozása
   - Koordináta-rendszer felrajzolása
   - Alakzatok felrajzolásának lehetősége a koordináta-rendszerre
2. iteráció
   - Kép megnyitása
   - Kép, alakzatok lineáris transzformációja
3. iteráció
   - Alakzatok fájlból történő megnyitása
   - Kép, alakzatok nemlineáris transzformációja
   - Menüsor létrehozása

4. iteráció
    - Hibakezelés, tesztelés

Az első megvalósítandó feladat egy kezdetleges grafikus interfész létrehozása volt. Erre a java.awt és a java.swing package-ek voltak a segítségemre.

Először létrehoztam egy fő ablakot, amelyen elhelyeztem minden, később szükséges grafikus elemet.

A legnagyobb kihívást itt az elemek helyes elrendezése jelentette. Ahhoz, hogy meghatározzam a komponensek helyét, szükségem volt egy úgynevezett *LayoutManager* (elrendezés) használatára, amelyet minden tároló elem esetén be kellett állítanom a *setLayout()* függvény segítségével. Az elrendezések közül panelek esetében a *GroupLayout*-ot használtam, mert nagy szabadságot nyújt az elemek pozicionálásában. A fő ablakom (*MainFrame*) esetében az alapértelmezett beállítást választottam, ami a *BorderLayout* volt.

A program főképernyőjét a *MainFrame* osztály valósítja meg, mely a *JFrame* osztályból származik. Ez az osztály két további úgynevezett *Container* osztályt tartalmaz.

Az egyik ilyen osztály a *TransformationSwMenuBar*, amely a *JMenuBar* osztályból származik, és a program menüsorát implementálja. Ez az osztály két *JMenu* objektumot tartalmaz. Az egyik a Fájl menüért, a másik pedig a Súgó menüért felelős. A Fájl menüt megvalósító objektum további *JMenuItem*eket tartalmaz, amelyek a menüpontokat írják le.

A másik ilyen tároló osztály a *StorageTabbedPane* osztály, amely a *JTabbedPane* osztályból származik. Ez az osztály teszi lehetővé a fülek (tabok) közötti váltásokat. Természetesen ez az osztály további tároló elemeket tartalmaz, amelyek a fülek váltogatása során felváltva jelennek meg. A programunk esetében két fülre volt szükségünk, ezért két ilyen tároló elem osztályunk van, a *WorkspaceStoragePanel* és az *ImageStoragePanel*, amelyek szintén 2-2 tároló elemet tartalmaznak. A *WorkspaceStoragePanel* osztály a Munkalap nézet megvalósításáért, az *ImageStoragePanel* pedig a Képfeldolgozó nézet implementálásáért felelős. Mivel a Munkalap nézet esetében szükségünk van egy koordináta-rendszer megjelenítésére alkalmas panelra, és egy eszköztárra is, ezért

további két osztály létrehozására volt szükség. A koordináta-rendszer felrajzolásáért és szerkesztéséért a *WorkspacePanel* felelős, az eszköztár megjelenítése pedig a *WorkspaceToolbarPanel* feladata. A Képszerkesztő nézet is két különálló részből áll. A kép megjelenítése az *ImagePanel* feladata, míg a képhez tartozó eszköztár megvalósítása az *ImageToolBar* osztály feladata. A Toolbar osztályok további *JTextField*eket, *JLabel*eket és *JButton*okat tartalmaznak.

Koordináta-rendszer felrajzolása

Miután elkészült a program grafikus váza, a koordináta-rendszer felrajzolása volt a következő feladat. A *WorkspacePanel* osztály *drawCoordinateSystem()* metódusát használtam erre a célra, majd a panelra rajzolást a panel *paint()* metódusának újraírásával valósítottam meg, amelyen belül meghívtam többek között a *drawCoordinateSystem()* függvényt is.

Először is érdemes pár szót ejteni a *paint()* függvényről, ami a továbbiakban a *WorkspacePanel*ra rajzolásért felelős, bármilyen ábráról legyen is szó. Ez a függvény egy *Graphics* típusú paramétert vár. Első lépésként létre kell hoznunk egy *Graphics2D* objektumot, amelyet a paraméterben kapott objektumból hozunk létre típuskonverzió segítségével. Az így létrehozott *Graphics2D* objektumot *mainGraphics*-nak neveztem el. Ezután nincs más dolgunk, mint a *mainGraphics* objektum segítségével meghívogatni a *draw()* függvényeket. Első lépésként definiáltam a koordináta-rendszer origóját, és a 4 végpontját, ezek a pontok voltak segítségemre az alakzatok elhelyezésekor. A koordináta-rendszer origójának a panelünk középpontját jelöltem ki, a koordináta-rendszer beosztását pedig alapértelmezettként 10 pixelenként rajzoltattam ki. Ezt az értéket egy úgynevezett *rateSpace* nevű változóban tároltam, hogy a későbbiekben módosítani tudjam, ha esetleg nagyobb beosztást szeretnék. A fix 10 pontos beosztásnak előnye, hogy a pont kirajzolása során az *x*, és az *y* koordináták magasabb értékeket vehetnek fel, hátránya viszont az, hogy a pontokat kizárólag egy tizedes jegy pontossággal tudjuk ábrázolni. Ezután következett a kirajzolás a már említett *drawCoordinateSystem()* függvény segítségével. Itt először a tengelyeket rajzoltam fel a végpontok segítségével, majd a tengelyek végén elhelyezkedő

nyilakat, végül pedig a beosztást. Mivel a koordináta-rendszer csak vonalakból áll, ezért mindenhol a *mainGraphics.drawLine()* függvényt használtam a megfelelő paraméterekkel. A program lehetőséget nyújt rácsvonalak berajzolására is, amely programozási szempontból a következőképpen néz ki. Ha az *isGrid* változó igaz értéket kap, az origótól számolva mindkét irányban beosztásnyi (*rateSpace*) közönként vonalakat rajzolunk a tengelyekkel azonos hosszúsággal.
A koordináta-rendszer megrajzolása után következhetett az alakzatok felvitele.

Alakzatok felrajzolása

Első körben létrehoztam a *WorkspacePanel* osztályon belül egy *components* nevű listát, amely a felrajzolni kívánt objektumok tárolására szolgál.

A rajzolás a már említett *paint()* osztály segítségével történik, ahol a lista elemének típusa alapján felrajzolja az alakzatokat a panelra valamilyen *draw()* függvény segítségével (, például *drawPoint(), drawLine(),* stb.).

Mivel minden alakzatot az általam rajzolt koordináta-rendszerben szerettem volna elhelyezni, ezért szükség volt az alakzatok koordináta – transzformációjára, hiszen a panelünk koordináta-rendszerének origója a bal felső sarokban van. Ez azt jelenti, hogy minden egyes pont, alakzat felrajzolása során ki kell számolni a pont tényleges helyét a panelen. Ezt a *calculatePoint()* függvény végzi. A Java nyelv rendelkezik *Point2D,* illetve *Line2D* osztályokkal, amelyeket igyekeztem hasznomra fordítani, viszont a körre, illetve a poligonokra nincs olyan osztály, amely képes lenne *double* értéket tárolni, ezért ezekre saját osztályt kellett létrehoznom *MyPolygon* és *MyCircle* néven. A felvinni kívánt koordinátákat az eszköztár szövegdobozain keresztül adhatja meg a felhasználó. Minden ilyen szövegdoboz mögött egy *JTextField* objektum áll. Az alakzatok felviteléhez a felhasználónak a Kirajzol gombra kell kattintania, amelyet egy *JButton* objektum ír le. A megvalósítás úgy történik, hogy a *JButton*hoz hozzáadunk egy *ActionListene*r, amely figyeli a gombot, és ha az megnyomásra kerül, végrehajtja az *ActionPerformed()* függvényében definiáltakat. Ebben a függvényben először is, a *TextField*ek tartalma alapján létrehozunk *Point2D, Line2D, MyCircle* vagy *MyPolygon* objektumokat attól függően, hogy kitöltött-e a felhasználó a ponthoz, szakaszhoz, körhöz tartozó rubrikákat. Ezután hívódik meg a *WorkspacePanel*

addComponentToComponents() függvénye minden egyes létrehozott objektumra. Ez a metódus a *WorkspacePanel components* nevű listájába pakolja bele az újonnan létrehozott objektumokat. Ezután meghívjuk a *WorkspacePanel repaint()* metódusát, amely a módosított *components* lista alapján újrarajzolja a panel tartalmát.

Az alakzatok kirajzolása típustól függően más-más elgondolás szerint történik. Egy pont kirajzolására saját függvényt írtam, hiszen egy pixel lerakása nem túl szemléletes. Ehelyett a megoldás helyett a *drawPoint()* függvény meghívásával teszünk fel pontot a koordináta-rendszerünkbe, melyet egy kis *x* jelöl, így valójában a *drawLine()* függvény többszöri meghívása bújik meg egy pont felrajzolása mögött. A poligonok megrajzolása is *drawLine()* függvények segítségével történik, melyet minden két egymást követő csúcsra meghívunk.

Kép megnyitása

A kép megnyitása a menübáron lévő Fájl menü/Kép megnyitása menüpont segítségével történik, így ehhez csak a Kép megnyitása mögött lévő *JMenuItem*hez kellett *ActionListene*rt rendelnem, melynek ismét az *ActionPerformed()* metódusa végzi a „piszkos munkát”. Ahhoz, hogy ki tudjunk választani egy képet, egy fájlválasztó ablakot kellett létrehoznunk, amelyet a *FileChooser* osztály valósít meg, így egy ilyen típusú objektumot hoztam létre. Mivel megnyitásról van szó, ezért az objektum *showOpenDialog()* függvényére van szükség, amely arra szolgál, hogy kilistázza az opciókat. Azért, hogy megkapjuk, hogy a felhasználó melyik képet választotta, a *getSelectedFile()* metódus felel. A választás után a választott fájlt átadjuk paraméterként az *ImagePanel setPictureOnPanel()* függvényének, amely a file alapján, a *paint()* függvény segítségével felrajzolja a képet a panelra. Ez úgy történik, hogy létrehozunk egy *BufferedImage* típusú objektumot, amelybe beolvassuk a file tartalmát, majd ezt a képet tudjuk felrajzolni a panelunkra.

Kép lineáris transzformációja

Mivel rendelkezünk már transzformálható képpel, ezért jöhet annak transzformációja. Minden transzformációt egy *Transformation* osztály végez. A

lineáris transzformáció egy mátrix alapján történik, amelyet a felhasználó négy szövegdoboz segítségével adhat meg, ez programozási szempontból négy *TextField*et jelent. A mátrix megadásának befejező momentuma a Transzformálás gombra kattintás, amely a kódban, hasonlóan az alakzatok felvitelénél, ismét egy *JButton*, hozzárendelt *ActionListener* és az *actionPerformed()* függvény segítségével történik. Itt megvizsgáljuk, hogy minden *TexField* tartalmaz-e helyes karaktersorozatot, és ha igen, meghívjuk az *ImagePanel linearTransformation()* függvényét a *TextField*ek tartalmával paraméterezve. Ez a metódus csak annyit tesz, hogy meghívja a *Transformation* osztály *linearTransformationOfImage()* metódusát a mátrix adataival, és a *pictureOnPanel BufferedImage*-el, amely a panelen lévő képet reprezentálja.

A *Transformation* osztály végzi a munka nagyobbik felét. A *linearTransformationOfImage()* függvény pixelenként transzformálja a képet. Minden egyes pixelre meghívjuk a *linearTransformationOfPixel*() metódust, amely egy mátrix szorzás segítségével meghatározza a pixel régi koordinátái alapján annak új helyét. Ezután Az újonnan megkapott koordináták helyére beállítjuk a *setRGB()* függvény segítségével a pixelünk értékét, majd a függvény visszaadja a transzformált képet. Az *ImagePanel linearTransformation()* metódusán belül beállítjuk a panel *transformedImage* változójának értékét az előző függvény visszatérési értékére, illetve a *pictureOnPanel* értékét is ezzel tesszük egyenlővé, majd meghívjuk a *repaint()* metódust, ami kirajzolja a transzformált képet. A *transformedImage* változóra a visszavonás lehetősége miatt van szükség.

Alakzatok lineáris transzformációja

Az alakzatok lineáris transzformációja hasonlóan történik a kép lineáris transzformációjához, itt is az eszköztár szövegdobozai segítségével kapjuk meg a transzformációs mátrixot. A különbség annyi, hogy itt a *WorkspaceToolbar* osztályban a megfelelő *JButton* megnyomása után lefutó *actionPerformed()* függvényben a *WorkspacePanel linearTransformation()* függvényét hívjuk meg a mátrix adataival. Ez a metódus annyit tesz, hogy meghívja a *Transformation* osztály *linearTransformationOfWorkspace()* metódusát a mátrix adataival és a

components listával paraméterezve. Minden alakzattípushoz létrehoztam egy transzformáló függvényt (például Line2D transzformációja a *transformLine()* függvény segítségével történik.). Mivel a lineáris transzformáció egyenes tartó, ezért elég csak a vonalak két végpontját transzformálni, majd ezeket újra összekötni. Ez jelentősen megkönnyíti a munkánkat. A pontok transzformációja szintén mátrixszorzással történik a pont koordinátái alapján. A végeredmény határozza meg a transzformált pont koordinátáit. A függvény tehát annyit tesz, hogy végig megy a komponens listán, és a komponens típusának megfelelően transzformálja azt, majd az így kapott új objektumot beletesszük egy segédlistába, amely majd az alakzatok transzformáltját tartalmazza.
Ennek a metódusnak a visszatérési értéke a transzformált objektumok listája. A *WorkspacePanel linearTransformedShapes* listájához a panel *linearTransformation()* metódusában hozzáadjuk a visszatérési értékként visszakapott lista elemeit, majd meghívjuk a *repaint()* metódust.

Alakzatok felrajzolása adatfájlból

A felhasználói igények közé tartozott, hogy adatfájlból is felvihetőek legyenek az alakzatok, ezért létre kellett hozni egy olyan menüpontot, ahol ezt a felhasználó könnyedén megteheti, ami újabb *JMenuItem*et jelent a File nevű *JMenu*höz. Itt is az *ActionListener – actionPerformed()* páros a nyerő. Az *actionPerformed()* metóduson belül létrehozunk ismét egy *JFileChooser*t, majd a kiválasztott fájlt paraméterül használva meghívjuk a *WorkspacePanel addComponentsFromFile()* metódusát. A tényleges fájlbeolvasást viszont egy külön osztály, *OpenFile* végzi. Ennek az osztálynak két attribútuma van, egy objektumokból álló lista, és egy *File* típusú. Konstruktorként megadhatjuk a beolvasandó fájlt, majd meghívjuk a *start()* metódusát. Ez a függvény felel az adatok beolvasásáért. Ez a beolvasás úgy történik, hogy első körben tudjuk a rögzített feltételek miatt, hogy pontokat olvasunk be, ezért létrehozunk a kiolvasott adatok segítségével egy *Point2D* objektumot, majd ezt beletesszük a listánkba. Addig olvassuk a pontokat, amíg a beolvasott sztring nem egyenlő „***” karaktersorozattal. Ezután jöhet a vonalak beolvasása, itt *Line2D* objektumokat hozunk létre. A többi alakzat beolvasása is hasonlóan történik. Ha a fájlból kiolvastuk az utolsó „***” karakterláncot, a

függvény végrehajtása befejeződik. Ezután a *WorkspacePanel*en belül lekérhetjük a *getReadedComponents()* metódus segítségével a beolvasott komponenseket, majd az így kapott listát egyenlővé tettük a panelünk *components* listájával, majd ismét ráhívunk a *repaint()* függvényre. Ennek a függvénynek a segítségével az alakzatok megjelennek a koordináta-rendszerben.

Megkönnyíti a felhasználó dolgát, hogy kommenteket írhat az objektumok megadása mögé, így később is könnyebben átlátja az adatfájlt, ha újabb alakzat felvitelére van szükség.

Kép nemlineáris transzformációja.

A legnagyobb problémát a nemlineáris transzformáció okozta, mivel itt a transzformáció függvények alapján történik. Ehhez arra van szükségünk, hogy egy bemeneti karaktersorozatból egy értelmes matematikai kifejezést hozzunk létre, majd ezt a kifejezést számoljuk ki minden *(x,y)* pontra. Erre a problémára több megoldás is található az interneten, de ezek közül nem találtam olyan megvalósítást, ami teljes mértékben megfelelt volna az elvárásaimnak. A legtöbb, erre a célra tervezett program általában csak egyváltozós függvényekre működött, és nem volt egyértelmű a módosíthatóságuk, illetve akadt közöttük olyan is, amely bizonyos pénzösszeg befizetése után vált csak elérhetővé. Mivel egyik megoldás sem volt megfelelő, ezért egy saját beolvasó szoftvert írtam, mely kétváltozós függvények megoldása esetén is működőképes. Ezt a részprogramot egy külön osztályban valósítottam meg *ExpressionSolverProgram* néven. Az osztály konstruktorában két *double* típusú változót *(x, y)* és egy sztringet (megoldandó kifejezés) vár.

A részprogram működésének lényege, hogy fokozatosan dolgozza fel a sztringet, először a zárójeles kifejezéseket számolja ki, majd sorban oldja meg a matematikai kifejezéseket, figyelembe véve azok precedenciáját.

Első lépésben átalakítjuk a sztringet a *switchXYToNumbers()* metódus segítségével. Ez a függvény kicseréli a kifejezésben szereplő x-et, és y-t az aktuális értékekkel. Ez után minden – operátort kicserélünk „+-„-ra, hogy meg tudjuk különböztetni a negatív kifejezéseket a kivonás operátortól.

Az átalakítások után jöhet a számolás, amelyet két, *braceResolution()* és *expressionSolver()* függvény végez.

A *braceResolution()* függvény végzi a zárójelek felbontását, méghozzá a következő módon: a függvény megkeresi a legbelső, vagy az egyik belső zárójeles kifejezést, majd ezt átadja az *expressionSolver()* metódusnak megoldásra. A *expressionSolver()* visszatérési értéke az adott zárójeles kifejezés eredménye lesz, melyet újra visszaírunk a sztringbe, és újból keresünk egy zárójeles kifejezést. A zárójelfelbontás addig fut, amíg a kifejezés tartalmaz zárójelet. Ha már nincs zárójeles kifejezés, a végleges sztringre ismét meghívjuk a *expressionSolver()* függvényt, ami most már a végeredményt fogja visszaadni nekünk. De nézzük meg, hogyan is működik a *expressionSolver()* függvény, vagyis a számolásért valóban felelős metódus.

Először is figyelnünk kell a kifejezések megoldásának helyes sorrendjére, ezért első körben megnézzük, hogy a sztringünkben szerepel-e valamilyen legmagasabb precedenciájú kifejezés (pl cos, sin, tan, stb.). Ha igen, akkor a kifejezés mögötti számra vonatkozóan megoldjuk az adott kifejezést, majd a megoldást visszaírjuk a kifejezés helyére. Azt, hogy a kifejezés után milyen hosszan terül el a valós számunk, azt egy *isNumber()* metódus segítségével dönthetjük el, amely minden egyes karakterre megvizsgálja, hogy az adott karakter egy valós szám része-e , vagy sem. Ha igen, akkor tovább lép, egészen addig, amíg számot talál.

Amíg szerepel a kifejezésben a legmagasabb precedenciájú kifejezésből, addig azokat oldjuk meg, ha már nem, akkor ugrunk a következő precedenciájú műveletekre, egészen addig, amíg el nem érünk az összeadás, kivonás műveletekig. Mire a függvény a végére ér, már csak egy valós számot tartalmaz a sztringünk, ami az adott, kiszámolandó kifejezés végeredménye lesz.

A kép transzformációja tehát úgy történik, hogy az *ImageToolbarPanel*en az erre a célre létrehozott *TextField*ek és a hozzájuk tartozó *JButton* segítségével megkapjuk a függvényeket leíró sztringeket. Itt is a *JButton*hoz rendelt *ActionListener* és az *actionPerformed()* függvény játszik főszerepet. Itt meghívjuk az *ImagePanel nonlinearTransformation()* metódusát a két *TextField*ből kinyert sztringeket adva paraméterül. Ebben a függvényben a *Transformation* osztály *nonlinearTransformationOfPicture()* függvényét hívjuk a két kifejezéssel, és a

transzformálandó képpel paraméterezve. Ebben a függvényben megkezdődhet a transzformáció. A kép minden pixelére kiszámoljuk az új koordinátákat a *ExpressionSolverProgram* osztály segítségével, amelyet az előzőekben részletesen taglaltunk. Mivel az *x*, és az *y* koordinátát is transzformálnunk kell, ezért mindkét koordinátára alkalmaznunk kell a számolásokat. Az így kapott koordináták jelzik a pixelünk új helyét, melyet a *setRGB()* függvénnyel állítunk be. Az így kapott új képet adja vissza a *nonlinearTransformationOfPicture()* függvény, melyet az *ImagePanel* osztály *nonlinearTransformation()* függvénye segítségével a *nonlinearTransformedImage BufferedImage*-el teszünk egyenlővé, illetve a *pictureOnPanel* változó értékét is erre módosítjuk. Ezután a *repaint()* metódus hívásával rajzoljuk fel a transzformált képet a panelra.

A kép esetében azonban ez a megoldás túl lassúnak bizonyult, ezért szükség volt a számolások külön szálba helyezésére. Ez úgy történik, hogy először a *nonlinearTransformedImage()* függvényen belül lekérjük, hogy a rendelkezésre álló gépünkön hány szálat tudunk indítani.
Ezt a *Runtime.getRuntime().availableProcessors()* függvény segítségével tehetjük meg. Ezután a transzformálandó képünket annyi részre osztjuk, ahány processzorunk elérhető, majd minden egyes képrészletet külön szálba teszünk, ami azt jelenti, hogy egy-egy *TransformationThread* végzi a pixelek új helyének kiszámolását. A transzformáció tesztelése érdekében ebbe az osztályba építettem be egy úgynevezett időzítőt is, amely a futás idejét méri miliszekundumban.

Alakzatok nemlineáris transzformációja

Az alakzatok nemlineáris transzformációja teljesen hasonló a képek nemlineáris transzformációjának megvalósításához. Itt is a *WorkspaceToolbar JButton, ActionListener, actionPerformed()* függvényében kezdődik minden. Itt kiolvassuk az adatokat a *TextField*ekből, majd az előzőekhez hasonlóan átadjuk ezeket a stringeket a *WorkspacePanel nonlinearTransformation()* függvényének. Ez a függvény szintén csak annyit csinál, hogy a panelen lévő komponensek listáját, és a két kifejezést sztringként átadja a *Transformation* osztály *nonlinearTransformationOfWorkspace()* metódusának. Ebben minden alakzatot pontonként transzformálunk, ezért itt is minden alakzatnak megvan a saját

transzformáló függvénye. A pontokat a *nonlinearTransformPoint()* függvény hívásának segítségével tehetjük meg, amely az *ExpressionSolverProgram* osztályt hívja segítségül. Mivel a transzformált alakzatok tulajdonságaikat tekintve nem egyeznek meg az eredeti alakzatokkal, ezért ezeket egy új, *TransformedShape* , *TransformedLine* osztályok írják le. Ezekben az osztályokban tároljuk, hogy az adott alakzat hány pontból áll, illetve a pontjainak listáját is.

A *nonlinearTransformationOfWorkspace()* metódosunkban végig megyünk a komponenseken, majd minden egyes komponensre meghívjuk saját transzformáló függvényét. A transzformált alakzatokat egy újabb listában tároljuk. A függvényünk ezzel a listával tér vissza, amelyet a *TransformationPanel* osztály *nonlinearTransformation()* metódusa dolgoz fel. Ez a függvény hozzáadja a panel *nonlineartransformedShapes* listájához a visszatérési értékben definiált lista elemeit. Ezután következhet az alakzatok kirajzolása, amelyet a *repaint()* függvény segítségével érhetünk el.

Mivel az alakzatok felrajzolása nem feltétlenül egyértelmű, ezért ejtenék még pár szót a *WorkspacePanel*ről.

Három attribútoma van, amely az eredeti, a lineárisan és a nemlineáris transzformált elemeket tartalmazza. Mindhárom attribútum objektum-lista típusú. Az első listát az *addComponentToComponents()* függvény tölti fel, amely minden újonnan hozzáadott alakzatot felfűz a lista végére. A második listában a lineárisan transzformált alakzatokat tároljuk, a nemlineáris alakzatok tárolásáért pedig a harmadik lista felelős.

A kirajzolás a *paint()* függvényben valósul meg, ami az alábbi módon történik. A program végig megy a tömbökön, és az alakzat típusának megfelelően, először a *transformPoint()* függvény segítségével minden, a kirajzoláshoz szükséges pontot transzformál a panel koordináta-rendszerének megfelelően, majd az így kapott pontokat már könnyen ki tudjuk rajzolni a rajzoló (*draw()*) függvények segítségével. Például ha egy vonalat szeretnénk kirajzoltatni, akkor a *drawLine()* függvényt alkalmazzuk,amelynek paraméterében megadjuk a két, már transzformált végpontot. (Ez a transzformáció nem az a transzformáció, amelyet a program hivatott bemutatni, ez kizárólag a program működéséhez, az alakzatok helyes kirajzolásához szükséges.)

Az eredeti alakzatok az alábbi típusúak lehetnek: *Point2D*, ha egy pontról van szó, *Line2D*, ha egy vonalról, szakaszról, *MyPolygon*, ha egy sokszögről és *MyCircle*, amely a kört reprezentálja. Azért volt szükség saját osztályok létrehozására, mert a Java nyelvben meghatározott hasonló osztályok csak egész értékeket képesek értelmezni, nekünk viszont szükségünk van a valós számok kezelésére is a megfelelő pontosság érdekében.

A nemlineárisan transzformált alakzatok kirajzolása nem végezhető el egy egyszerű *drawLine()* metódus meghívásával. Itt három féle objektumtípus fordulhat elő. Egy transzformált alakzat lehet Point2D típusú, ha csak egy pontot transzformáltunk, lehet *TransformedShape* típusú, ha egy alakzatot transzformáltunk, és lehet *TransformedLine* típusú, ha egy vonalat, szakaszt transzformáltunk.

A pontunk felrajzolása nem jelenthet problémát, hiszen az előzőekben leírtak alapján történik. A *TransformedShape* alakzat felrajzolása úgy történik, hogy az alakzat összes pontját transzformáljuk a *transformPoint()* metódus segítségével, majd az így kapott pontokat a *drawLine()* segítségével összekötjük.
Azért volt szükség a *TransformedLine* osztály megalkotására is, mert az ilyen típusú alakzat kirajzolása egy kicsivel másképpen történik, mint ahogy a *TransformedShape* típusúaké. Ebben az esetben az első és az utolsó pont összekötésére nincs szükség.

Kép javítása, pozícionálása

A kép javítása és pozícionálása egy úgynevezett *PopupMenu* segítségével történik, amely különböző *MenuItem*-ekből áll. Három *MenuItem* a kép pozicionálásáért, egy pedig a javításáért felelős. A pozicionálás során nagyon egyszerű a dolgunk, csak a képünk bal felső sarkának pozícióját állítjuk át. A kép javítása során viszont komolyabb munkálatok állnak a háttérben. Itt úgynevezett dilatációt végzünk, amely a pixelek „megduzzasztását” jelenti valójában. (A Linux rendszereken található Gimp képszerkesztő is ezt a megoldást használja a képek javítására.) Ezt egy *ImagePlus* osztály segítségével végezzük. A dilatációt a *dilate()* függvény végzi.

Menübár összeállítása

A Menübár létrehozásának módját már a fejezet elején megemlítettem, sőt a Megnyitás funkcióért felelős menüpontokat is megtárgyaltuk. Vizsgáljuk meg a fennmaradó menüpontokat is.

Új Munkalap

Mivel munka közben szükségünk lehet arra, hogy új Munkalapot nyissunk, ezért ezt a funkciót is be kellett építeni a programba. Ezért szintén egy *JMenuItem* felelős, illetve a *JMenuItem ActionListener*ében definiált *actionPerformed()* függvény. Itt adható meg, hogy mi történjen, ha az adott menüpontra kattintunk. Ekkor a program a *WorkspacePanel components* listáját *null* értékre állítja,majd meghívja a panel *repaint()* metódusát. Mivel nem lesz felrajzolandó alakzat a rendszerben, ezért egy üres koordináta-rendszert láthatunk a panelen.

Egy Munkalap, vagy egy transzformált kép mentésének lehetősége is elengedhetetlennek bizonyult, hiszen a Munkalap esetében fontos lehet, hogy ne kelljen mindig újra felrajzolni az alakzatokat, menthessük a Munkalapot, és újra meg tudjuk nyitni azt szerkesztési vagy éppen transzformálási céllal.
A Mentést, hasonlóan az előzőekhez, szintén egy *JMenuItem* osztály segítségével valósítottam meg. Az *actionPerformed()* függvényben itt először létrehozunk egy *FileChoose*r, ami arra szolgál, hogy a felhasználó megkereshesse a számítógépen a menteni kívánt Kép vagy Munkalap helyét, illetve ha rá szeretne menteni egy már létező fájlra, akkor kiválaszthassa azt. A felhasználó ezután megadja a menteni kívánt fájl nevét (hacsak nem egy régebbit választott), és a Mentés gombra kattint. Ekkor a program létrehoz egy fájlt az adott néven (, vagy szerkeszti a már meglévőt.). Ha a jelenlegi felület volt már mentve valamilyen néven, akkor a *FileChooser* ablak nem jelenik meg, hanem a program a háttérben módosítja a már létrehozott fájlt. Ezt úgy valósítottam meg, hogy Munkalap esetében egy *workspaceWasSaved boolean* típusú változót, kép esetében pedig egy *pictureWasSaved boolean* változót hoztam létre, melyek egyszeri mentés után *true* értéket vesznek fel, és ezen túl egy új Munkalap létrehozásáig, vagy egy új kép megnyitásáig mentés során a már tárolt (Workspace esetében *workspaceFile*, Kép esetében *PictureFile*) *File* típusú változóba mentjük a változásokat.

A mentést egy külön *FileSaver* osztály végzi, amely a *savingWorkspace()* metódusa segítségével a paraméterben kapott fájlba beleírja a *components* listában tárolt alakzatok adatait. Először a pontokat koordinátáik szerint, majd a vonalakat stb. írjuk ki a fájlba alkalmazkodva az adatfájl kötelező formai követelményeihez.
Kép mentése esetében a képet írjuk ki a fájlba az *ImageIO.write()* függvény segítségével.

A Mentés másként menüpont mögött szintén az előzőekben említett függvények dolgoznak azzal a különbséggel, hogy itt nem vesszük figyelembe azt, hogy elmentettük-e már az aktuális Képünket, Munkalapunkat, hanem mindenképpen felkínálja az új néven, új helyre történő mentés lehetőségét.
A Visszavonás ebben az esetben azt jelenti, hogy visszaállíthatjuk az eredeti Képünket, Munkalapunkat, hogy újra transzformálni tudjuk azt. A program először is lekéri, hogy melyik fülön van, majd ennek megfelelően dolgozik. Ekkor a háttérben vagy a *WorkspacePanel*-on nullázzuk a transzformált elemek listáját, vagy az *ImagePicturePanel*-on nullázzuk a transzformált képet, majd a *repaint()* függvényünk segítségével újrarajzoljuk az adott panelt.

Kilépés Menüpontot úgy valósítottam meg, hogy a *MenuItem*hez hozzáadott *ActionListener actionPerformed()* függvényében meghívjuk a *System.exit(0)* metódust, amely a programból való kilépést eredményezi.

A Súgó Menüt egy *JMenu* osztály valósítja meg. Ha rákattintunk a Súgó menüre, egy új ablak ugrik elő, melyben egy kis kereső doboz segítségével megadhatjuk a keresni kívánt címszót. Ezt a keresési mezőt egy *TextField* írja le, a szövegezést, amelyben kereshetünk, pedig egy *TextArea* típus. A megtalált szó kiemelését egy *HighLighter* végzi, az összehasonlítás pedig úgynevezett *Pattern*ek alapján megy végbe.

2.3.2.1. Hibakezelés

A program futása során több olyan szituáció állhat elő, amikor a program hibás kimenetet eredményez. Ilyen eset adódhat például a felhasználó figyelmetlenségéből az adatok megadása során, illetve valamilyen gomb véletlenül történő megnyomásával.

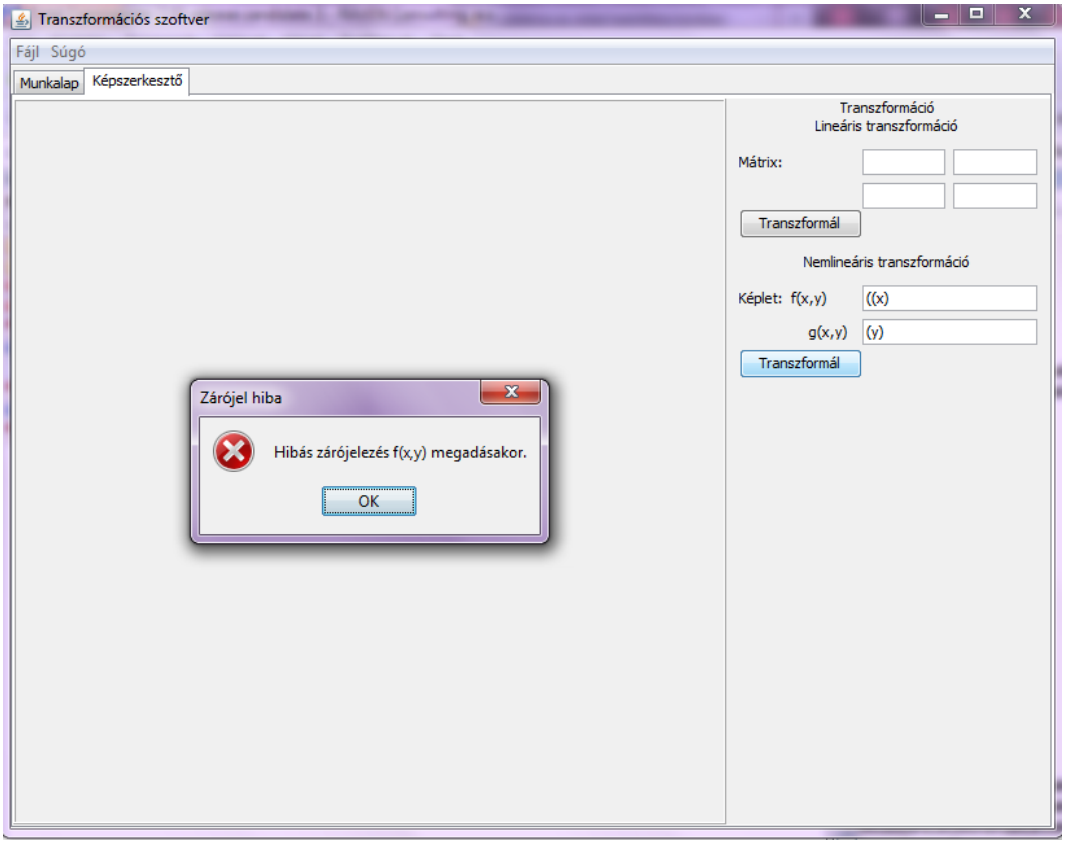
Ahhoz, hogy a programom jól tudja értelmezni a pontokat, és ezek alapján helyesen tudja felrajzolni az ábrákat, szükség van a pontok koordinátáinak megfelelő alakban történő megadására. Ez az alak az (*x*,*y*) alak. Bármilyen ettől eltérő input hibát eredményezhet. Éppen ezért szükség volt a *Textfield*eken keresztül megadott sztringek vizsgálatára. Erre a problémára a reguláris kifejezések jelentették a megoldást. Ennek segítségével megadtam egy olyan reguláris kifejezést, amely leírja, hogy milyen alakban kell megadnunk az adott *TextField*be beírt szöveget. Erre példa a pont esetében a „([1-9]+.?[1-9]*,[1-9]+.?[1-9]*)" kifejezés. Ha erre a kifejezésre illeszkedik az input sztringünk, nagy valószínűséggel a megfelelő formában adtuk meg az adott pontot.

A mátrix megadásánál is hasonló a helyzet, de itt nem tehetjük zárójelbe a bevinni kívánt adatot.

A nagyobb problémát a függvények megadása mező jelentette, mivel itt nem tudtam megadni egyértelműen reguláris kifejezést, ezért másik módszerhez kellett folyamodnom.

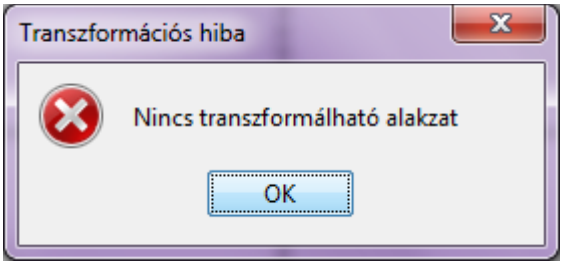
A függvények *TextField*jeit egy külön osztály segítségével figyeltem, illetve olyan *KeyListener*eket hoztam létre, amelyek azt vizsgálják, hogy egy adott karakter begépelése után milyen másik karakter következhet, illetve hogy az adott karaktert egyáltalán beírhatjuk-e a mezőbe. Az elgépelés elkerülését segíti az automatikus kiegészítés is.

Ekkor még mindig fennállhat a veszélye, hogy a felhasználó elrontotta a zárójelezést. A program ezt is kiszűri, méghozzá úgy, hogy a *checkBracket()* függvényünk megszámolja a nyitó, illetve a záró zárójeleket, mindezt oly módon, hogy végig megy a sztringen, és ha talál egy nyitó zárójelet, akkor növel egy számlálót, ha záró zárójelet talál, akkor csökkenti azt. Ha a számláló a sztring vizsgálatának végeztével 0 értéket vesz fel, akkor a zárójelezés jó, ha nem, akkor a felhasználó rosszul adta meg a kifejezést. Erről szintén tájékoztatást kap.



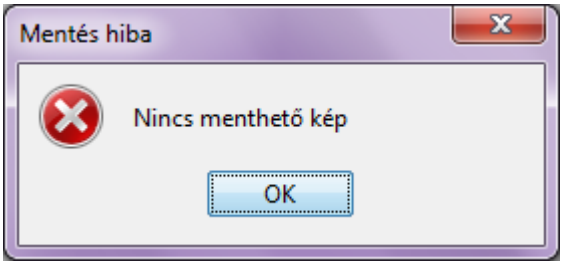
2.18. ábra: Hibás zárójelezéskor felbukkanó hibaüzenet

Figyelnem kellett arra is, hogy ha esetleg a felhasználó úgy szeretne transzformálni, hogy nem adott meg képet, vagy nem vitt fel alakzatot, akkor ezt jelezzem számára. Ezt egy *Dialog* ablak megvalósítása tette lehetővé.



2.19. ábra: Transzformáláskor felbukkanó hibaüzenet hiányzó alakzatok esetén

Ugyan ilyen probléma merülhet fel a Kép mentése esetében is, hiszen ha nem nyitottunk meg képet, és nem transzformáltuk azt, akkor nincs is képünk, amit menteni tudnánk. Ezt szintén egy Dialog ablak adja tudtunkra.



2.20. ábra: Transzformáláskor felbukkanó hibaüzenet hiányzó kép esetén

Bár nem feltétlenül tartozik a hibakezelés témakörébe, én mégis itt szeretném tárgyalni azokat az eseteket, amikor szintén egy *Dialog* ablak segítségével tájékoztatjuk a felhasználót munkájának megkönnyítése érdekében. Egy példa erre, amikor megnyitni szeretnénk egy új Képet/Munkalapot, de az aktuális Képet/Munkalapot már szerkesztettük, és nem mentettük a változásokat. Ilyenkor a program megkérdezi a felhasználót, hogy mentse-e a módosításokat.
Ugyan ez a párbeszéd játszódik le akkor is, ha Új Munkalapot szeretnénk nyitni.

2.3.3. Tesztelés, tesztesetek

A program tesztelésére azért van szükség, hogy egyrészt a felhasználó lássa, hogy milyen lehetőségei vannak, hogy egyes transzformációk milyen inputra milyen képet generálnak, másrészt tudja, mennyi idejébe fog telni egy adott méretű kép transzformálása, és hogy esetlegesen mekkora képet válasszon a vizsgálódáshoz gépének teljesítményéhez mérten. Mivel a Munkalap transzformációja nem vesz el sok időt, illetve a kép lineáris transzformációja sem, ezért a kép nemlineáris transzformációját vizsgálom.
Két különböző számítógépen végeztem méréseket, és az így kapott eredményeket táblázatba szedtem, ami a következőkben látható.

1. számítógép adatai: Intel T2370 @ 1.73 GHz dual core, 2GB ram
2. számítógép adatai: Intel P8600 @ 2.4 GHz dual core, 4 GB ram

	Függvény	Kép mérete	Futási idő
1. gép	$sqrt(x * y)$, $(x + y)$	100 x 100	2668 ms
		200 x 200	7594 ms
	$x^2, (\frac{x}{2} + y)$	100 x 100	2369 ms
		200 x 200	6634 ms
2. gép	$sqrt(x * y)$, $(x + y)$	100 x 100	1201 ms
		200 x 200	2212 ms
	$x^2, (\frac{x}{2} + y)$	100 x 100	774 ms
		200 x 200	1675 ms

2.4. Hasonló programok

Mielőtt hozzákezdtem volna a saját alkalmazásom fejlesztéséhez, olyan matematikai szoftverek után kutattam, melyek funkcionalitása lefedi az általam megtervezett programét, mindezt hiába. Nem találtam olyan alkalmazást, mely alkalmas lett volna a nemlineáris transzformációk szemléltetésére is.
A legismertebb, és a szakdolgozatomhoz is legközelebb álló a GeoGebra nevű program volt, amely nagy segítséget nyújthat az oktatóknak, tanulóknak egyaránt. Szinte a matematika minden területén hasznos lehet.

A transzformációk bemutatása terén viszont nem olyan széles a választék, a geometriai transzformációk közül a centrális tükrözés, a tengelyes tükrözés, a pont körüli forgatás, az eltolás és a centrális nyújtás szerepel a lehetőségek között. Részletesen kidolgozott megjelenítés és sokrétű felhasználhatóság jellemzi, de nem fedi le az általam tervezett szoftver funkcionalitását, hiszen nem képes a nemlineáris transzformációk bemutatására.

A GeoGebra részletes bemutatását az alábbi linken találjuk meg: http://wmi.math.u-szeged.hu/mediawiki/index.php/GeoGebra.

2.5. További lehetőségek

A jövőre nézve szeretném folytatni a programom továbbfejlesztését, mert sok lehetőséget látok benne. Első sorban a koordináta-rendszer pontosságán lehetne javítani, illetve a programba be lehetne építeni egy 3D-s ábrázolást is, mely sokkal izgalmasabbá és érdekesebbé tenné a transzformációk prezentációját.

3. Összefoglalás

Láthatjuk, hogy a transzformációk az élet szinte minden területén jelen vannak, ezért fontos időt fordítani rájuk. A matematikai transzformációk segítségével is rengeteg érdekes képet, ábrát lehet létrehozni, melyeknek rajzolása szórakoztató lehet. Programom lényege éppen ebben rejlik, célja a transzformációknak egy könnyű szemléltetése, amit a felhasználó egyszerűen el tud végezni maga is, és rögtön látja az eredményt, nem kell órákig számolgatnia.

A szoftver fejlesztése során megismerkedtem a Java nyelvvel, különösen a grafikus elemeket megvalósító osztályokkal, a többszálas programok megírásának bonyodalmaival, a reguláris kifejezések használatával, a képfeldolgozásban a kép javítására használt függvényekkel, algoritmusokkal és nem utolsó sorban a transzformációk helyes használatával, és sokszínűségével.

A jövőben, mint ahogy már azt az előzőekben említettem, szeretnék tovább foglalkozni a program fejlesztésével.

Irodalomjegyzék

[1] http://en.wikipedia.org/wiki/Transformation_matrix Transformation matrix (letöltés dátuma 2012. december 8.) Wikipédia információ.

[2] http://www.geo.u-szeged.hu/~bodis/maths/szakdolgozat/ BÓDIS, K. (1999) (letöltés dátuma 2012. november 30.) Geometriai transzformációk, transzformációs egyenletek és alkalmazásuk a geoinformatikában.

[3] http://hu.wikipedia.org/wiki/Affin_transzform%C3%A1ci%C3%B3 Affin transzformáció (letöltés dátuma 2013. április 3.) Wikipédia információ

[4] http://matek.fazekas.hu/mathdisplay/cache/pdf/volume_g_iii.pdf DOBOS, S. , HRASKÓ, A. (letöltés dátuma 2013. május 3.) Geometria 11-12. évfolyam

[5] http://www.math.bme.hu/~hujter/120415.pdf dr. SZALKAI, I.: Szemléltetés és becsapás (letöltés dátuma: 2012. december 1.) Haladvány Kiadvány

[6] http://math.uni-pannon.hu/~szalkai/ dr. SZALKAI, I.(letöltés dátuma 2012. december 1.) Geometriai transzformációk mátrixai

[7] http://mathworld.wolfram.com/AffineTransformation.html Affine Transformation (letöltés dátuma 2012. december 1.) Wolfram MathWorld

Mellékletek

A dolgozathoz mellékelt CD mappaszerkezete és a fájlok listája:

- Szakdolgozat
  - transzformaciok.doc
  - transzformáciok.docx
  - transzformaciok.pdf
- Irodalomjegyzék
  - Affine transzformáció – Wikipédia.htm
  - Affine Transformation – from Wolfram MathWorld
  - Geometria_Dobos_Hrasko.pdf
  - Geometriai transzformációk – Szakdolgozat.pdf
  - GeomTrafo1-jav.gif
  - GeomTrafo2-jav.gif
  - Szemléltetés és becsapás.pdf
  - Transformation matrix - Wikipedia, the free encyclopedia.htm
- Program
  - Adatfájlok
    - sablon.tdif
    - kisember.tdif
  - Java JRE
    - jre.zip
  - Képek
    - eper.png
    - eper_lintrafo.png
    - eper_nemlintrafo.png
  - logfile.log
  - transzformációsprogram.exe
  - transzformációsprogram.jar