

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

Informatikai Kar

Numerikus Analízis Tanszék

Cauchy-formula és diszkrét változatai a racionális függvényekkel történő approximációban

Témavezető:

Filipp Zoltán István

mestertanár, programtervező matematikus

Készítette:

Hegedüs Norbert

programtervező informatikus BSc

Budapest, 2020



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

SZAKDOLGOZAT-TÉMA BEJELENTŐ

Név: **Hegedüs Norbert**

Neptun kód: **D89B81**

Tagozat: **nappali**

Szak: **programtervező informatikus BSc**

Témavezető neve: **Filipp Zoltán István**

munkahelyének neve és címe:

ELTE IK Numerikus Analízis Tanszék

1117 Budapest, Pázmány Péter sétány 1/C.

beosztása és iskolai végzettsége:

mestertanár, programtervező matematikus

A dolgozat címe:

**Cauchy-formula és diszkrét változatai a racionális függvényekkel
történő approximációkban**

A dolgozat témája:

Jel- és képfeldolgozásban, irányításelméletben és EKG görbék elemzésében egyre inkább kiemelt szerepet kapnak a komplex racionális függvényekkel való közelítések. Az elemi racionális függvényekből kiindulva, megfelelő skaláris szorzatterekben a Gram-Schmidt-féle ortogonalizációs eljárással racionális ortogonális rendszereket tudunk generálni (pl.: Malmquist-Takenaka rendszer; MT-rendszerek) és az ezekben előforduló paraméterek megfelelő megválasztásával különböző approximációs feladatokat tudunk megoldani.

Szakedolgozatomban ilyen MT-rendszerek vizsgálatát tűztem ki célul, különösen a Cauchy-formula és diszkrét változatainak felhasználását ortogonális és biortogonális rendszerek szerkesztéséhez és az ezekkel való közelítésekhez. Az MT-rendszerek egyik előnye, hogy számos jól használható explicit formula áll rendelkezésre, mely megkönnyíti a számolásokat és a programozhatóságot. A konstrukciókban szereplő paraméterek alkalmazása, megválasztása, a jelekhez illeszthetősége nagyfokú rugalmasságot biztosít és olyan nyitott kérdéseket vet fel, amelyeknek a megválaszolása napjainkban is fontos kutatási területeket érint (irányításelmélet, EKG görbék modellezése és alkalmazása az orvosi diagnosztizálás területein, zajszűrés, adattárolás és tömörítés, stb...). Az MT-rendszerek jól illeszthetőek a vizsgált jel, függvény, EKG görbe jellegéhez. A diszkrét változatok lehetőségeit vizsgálva, FFT-szerű algoritmusokat is szeretnék használni. A Cauchy-formula diszkrét változatai lehetővé teszik a Fourier együtthatók közvetlen, gyors számolását és az újonnan konstruált approximációk javítását.

Mindezeket az eredményeket egy megfelelő programmal is szemléltetni fogom.

TARTALOMJEGYZÉK

BEVEZETÉS.....	5
Motiváció	5
Program rövid leírása	5
FELHASZNÁLÓI DOKUMENTÁCIÓ	6
Kompatibilitás	6
Telepítés	6
Matematikai háttér	6
Program fő elemei.....	7
Program részletes leírása	7
<i>Shapes</i> fül	8
Alakzat hozzáadása	8
Alakzatokhoz tartozó funkciók.....	9
Alakzatok attribútumai.....	11
Fő felület	11
Alakzatok beállítása a z-síkon.....	12
<i>Equations</i> fül.....	13
Blaschke-függvények megadása	13
Fő felület	14
<i>Menüsáv</i>	16
File menüpont	16
Edit menüpont	19
Settings menüpont.....	20
Info menüpont	20
<i>Egyebek</i>	21
Hiba ablakok.....	21
Állapotsor	21
FEJLESZTŐI DOKUMENTÁCIÓ	22
Fejlesztői eszközök	22
<i>Fejlesztői környezet</i>	22
<i>Felhasznált nyelvek</i>	22
Java.....	22
FXML.....	22
CSS.....	23
<i>Keretrendszerek</i>	23

JavaFX.....	23
JUnit	23
JSON-Simple	23
Program rétegei	23
Modell csomag.....	24
Complex csomag	24
Shapes csomag.....	27
Util csomag.....	28
Nézet csomag.....	31
Scenes csomag	31
Style csomag.....	33
Windows csomag	34
Vezérlő csomag	34
Fő vezérlő	34
Alakzatok vezérlői	37
Egyenletek vezérlői	40
Tesztelés.....	43
<i>Egység tesztelés</i>	<i>43</i>
<i>Manuális tesztelés.....</i>	<i>43</i>
<i>Shapes fül</i>	<i>43</i>
<i>Equations fül</i>	<i>46</i>
Menüsáv	47
ÖSSZEFOGLALÁS.....	50
ÁBRAJEGYZÉK	51
HIVATKOZÁSOK	52

BEVEZETÉS

Motiváció

Tanulmányaim során a komplex számokat és -függvénytant mindig is érdekes témakörnek tartottam, ebből adódóan választottam ehhez kapcsolódó témát szakdolgozatomnak. Emellett egy olyan alkalmazást szerettem volna készíteni, ami megkönnyíti azoknak a munkáját, akik ezen a területen dolgoznak.

Az orvostudományban az EKG görbék vizsgálata elengedhetetlen különböző betegségek diagnosztizálásához, viszont ezeknek a feldolgozása, zajszűrése, elemzése korántsem egyszerű feladat. A komplex racionális törtfüggvények és függvényrendszerek képe meglehetősen hasonlít az EKG görbe részeihez, így ez napjainkban is fontos kutatási területnek számít. Az ilyen függvények és ortogonális függvényrendszerek, például: *Blaschke*-függvények és -szorzatok, illetve *Malmquist-Takenaka* rendszerek (rövidebben MT-rendszerek) lényeges szerepet töltenek be ebben.

Témaválasztás után fontos volt számomra, hogy olyan eszközöket használjak a programom készítése során, amik megfelelnek a mostani elvárásoknak, így elkezdtem azon tündődni, hogy milyen programozási nyelvben lenne célszerű ezt megvalósítani. Töprengések után úgy döntöttem, hogy JavaFX keretrendszerben fogom megvalósítani az alkalmazásom. Ennek a keretrendszernek előnye, hogy a megjelenési réteget könnyedén el lehet különíteni a háttérben futó rétegtől a beépített technológiák segítségével. Így egy letisztult, könnyen kezelhető, jó megjelenésű asztali alkalmazást lehet készíteni, így növelve a felhasználói élményt.

Program rövid leírása

A programomban *Blaschke*-függvények és -szorzatok vizsgálatával és szemléltetésével foglalkozom. Az applikációban lehetőség van létrehozni alakzatokat tetszőlegesen állítható attribútumokkal és ezen alakzatok transzformálására a felhasználó által megadott *Blaschke*-függvények paraméterével, továbbá lehetséges ezen eredmények tárolása, olvasása, könnyed szerkesztése, illetve a transzformált

alkalmazatok exportálása kép formátumban. *Blaschke*-szorzatokkal való első-, másod- és harmadfokú egyenletek megoldására és vizuális reprezentációjára is lehetőség van.

FELHASZNÁLÓI DOKUMENTÁCIÓ

Kompatibilitás

A program platformfüggetlen, viszont igényel Java 13 vagy újabb verziójú futtató környezetet, amit az Oracle weboldaláról lehet ingyenesen letölteni. Microsoft Windows 10 Home vagy újabb kiadás az ajánlott operációsrendszer. Felhasználó élmény növeléséhez 1920x1080 felbontású FullHD monitorok ajánlottak.

Telepítés

Az alkalmazást nem szükséges telepíteni, a szoftver azonnal használhatóként van létrehozva. Az indítandó fájl a „Blaschke.jar” néven található meg a kicsomagolás után. A felhasználó egy parancsikon előállításával szabadon választott helyről indíthatja az alkalmazást a könnyebb hozzáférés érdekében. Az felhasználó által létrehozott fájlokat tetszőleges mappában lehet tárolni.

Matematikai háttér

Ebben az alfejezetben fogom bemutatni a *Blaschke*-függvényekről és -szorzatokról az alapvető tudnivalókat, amik a szoftver használatához szükségesek.

A komplex számok halmazát a megszokott módon jelölje \mathbb{C} , illetve legyenek az alábbiak a komplex számsík részhalmazai:

$\mathbb{D} := \{z \in \mathbb{C} : |z| < 1\}$, a nyílt komplex egységkör-lemez, másnéven **diszk**,

$\mathbb{T} := \{z \in \mathbb{C} : |z| = 1\}$, a komplex egységkör, másnéven **tórusz**,

$\overline{\mathbb{D}} := \{z \in \mathbb{C} : |z| \leq 1\}$, a zárt komplex egységkör-lemez, azaz $\mathbb{D} \cup \mathbb{T}$.

Egy $z = a + b\mathbf{i}$ komplex szám algebrai alakja esetén jelölje \mathbf{i} az imaginárius egységet, a valós részét $\operatorname{Re}(z)$, illetve a képzetes részét $\operatorname{Im}(z)$, ahol $\operatorname{Re}(z) = a$ és $\operatorname{Im}(z) = b$ áll fenn ($a, b \in \mathbb{R}$).

Legyen az $a \in \mathbb{D}$ paramétertől függő **Blaschke-függvény** az alábbi leképezés:

$$\mathcal{B}_a(z) := \frac{z - a}{1 - \overline{a}z} \quad (z \in \mathbb{C})$$

ahol az α szám a \mathcal{B}_α zérushelye, illetve $\alpha = 0$ választás esetén $\mathcal{B}_\alpha(z) = z$ identikus leképezést kapjuk vissza.

Továbbá $n \in \mathbb{N}$ és $\alpha_0, \dots, \alpha_n \in \mathbb{D}$ paraméterek mellett legyen a **Blaschke-szorzat** véges sok Blaschke-függvény szorzata, azaz

$$\mathcal{P}_{\alpha_0, \dots, \alpha_n}(z) := \prod_{k=0}^n \mathcal{B}_{\alpha_k}(z) \quad (k \in \mathbb{N}, z \in \mathbb{C})$$

Itt az $\mathcal{P}_{\alpha_0, \dots, \alpha_n}(z)$ zérushelyei az $\alpha_0, \dots, \alpha_n \in \mathbb{D}$ komplex számok, azaz n darab zérushelye van továbbá, ha $\alpha_k = 0 \Rightarrow \mathcal{P}_{\alpha_0, \dots, \alpha_n}(z) = z^n$ hatványfüggvényt kapjuk vissza.

Program fő elemei

A programot három fő rész alkotja.

- Menü sáv, ahol az alapvető funkciók érhetők el.
- *Shapes* fül, ami két részre bontható:
 - Bal oldali terület, ahol azokat az alakzatokat tudjuk felvenni és beállítani, melyeket ábrázolni szeretnénk.
 - Fő terület, ahol az alakzatokat és a képeit, illetve a Blaschke függvényt tekinthetjük meg.
- *Equations* fül, aminek a részei:
 - Bal oldali területen a Blaschke függvények paramétereit állíthatjuk be, illetve azt a $c \in \mathbb{C}$ konstans, amivel az $\mathcal{P}_{\alpha_0, \dots, \alpha_n}(z) = c$ egyenletet szeretnénk megoldani.
 - Fő területen pedig a függvények grafikonjait láthatjuk.

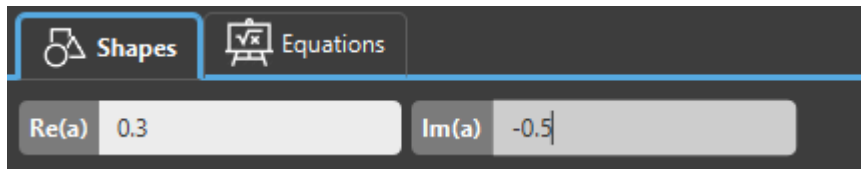
Későbbiekben fogom pontosan elmagyarázni a funkciók működését és használatát a programból kivágott képekkel illusztrálva.

Program részletes leírása

A füléken való navigáció kattintással történik, ahol az alkalmazás nagyobb részeit lehet használni.

Shapes fül

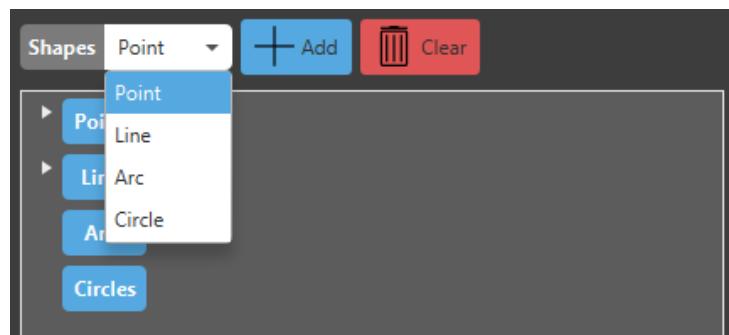
A Blaschke-függvény a komplex paraméterét az alábbi szövegdobozokban lehet beállítani. Itt a $\operatorname{Re}(a)$ a valós rész és $\operatorname{Im}(a)$ a képzetes rész.



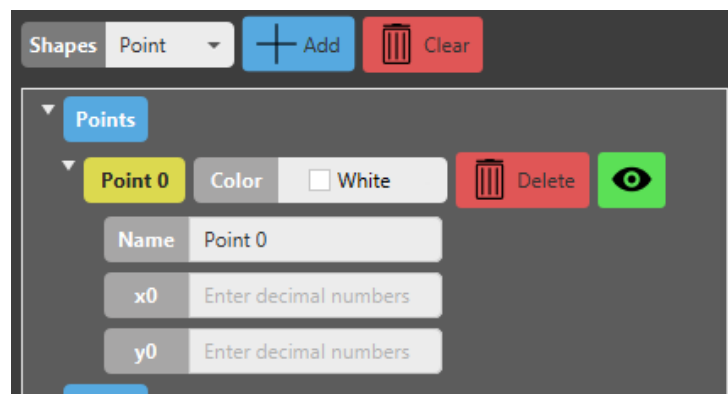
1. ábra: Blaschke paraméter megadása

Alakzat hozzáadása

A „Shapes” címke melletti lenyíló menüben választható ki az alakzat, amit az „Add” gombra való kattintással lehet hozzáadni. Az összes hozzáadott alakzatot a „Clear” gomb segítségével lehet kitörölni. Ezeket a 2. ábra, illetve 3. ábra ábrázolja.



2. ábra: Alakzat kiválasztása

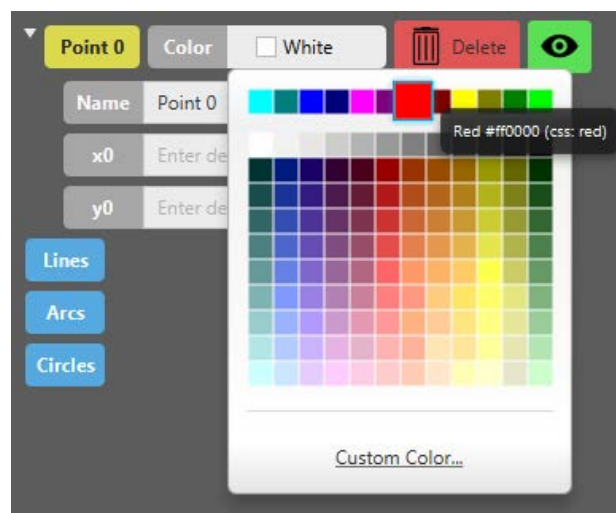


3. ábra: Alakzat hozzáadása

A hozzáadott alakzatok a fa nézetben a megfelelő típusuknál fog megjelenni, amiket a címkék melletti háromszögre való kattintással vagy a címkékre való kattintással lehet összecsukni vagy kinyitni.

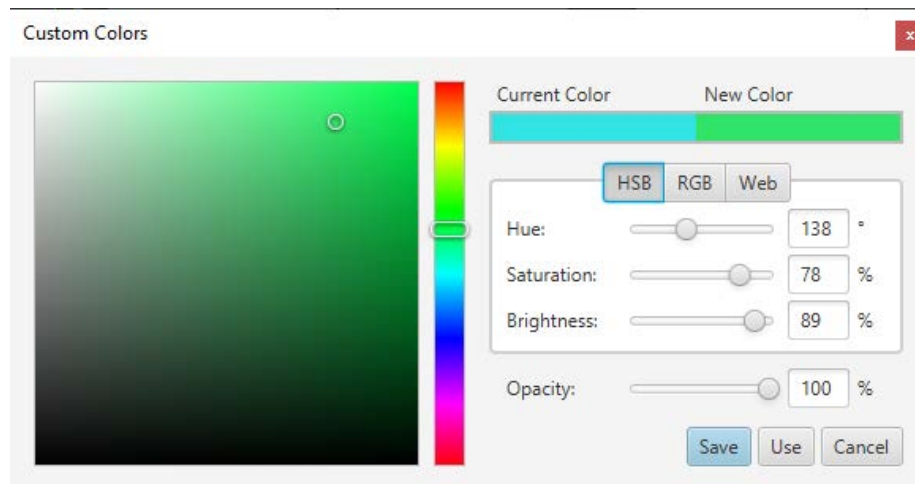
Alakzatokhoz tartozó funkciók

Négy alakzat adható hozzá a fa nézethez: pont („*Point*”), szakasz („*Line*”), körív („*Arc*”) és a kör („*Circle*”). Ezeknek számos tulajdonságát lehet megadni a beviteli mezők segítségével, illetve majd a z-síkon belül (erről részletesebben később). Minden alakzathoz tartozik egy címke, ami a nevét tartalmazza. Alapértelmezetten ez az alakzat neve és egy sorszám, azonban ez megváltoztatható a „*Name*” nevű beviteli mezővel. Továbbá meg lehet változtatni az alakzatok színét is a „*Color*” nevű gombbal. Minden alakzatnak az alapértelmezett színe a fehér. A felhasználónak lehetősége van előre megadott színek, és manuálisan megadott színek közül választani, ha a „*Custom color...*” nevű hivatkozásra kattint.



4. ábra: Színválasztó

A 4. ábra alapján itt láthatók az alapszínek, amiket kattintással van lehetősége kiválasztani az alkalmazás használójának. Ha rövid ideig a kurzort egy szín felett tartja, megtekinthető a szín neve és kódja. Esetleg, ha saját konkrét színt kívánunk megadni, akkor korábban említett hivatkozásra kattintva az alábbi ablak tárul elénk:



5. ábra: Saját szín megadása

Itt a „*Current Color*” alatt az aktuális szín, a „*New Color*” alatt az újonnan kiválasztott szín látszik. Kétféleképpen lehet megadni a színeket. A függőleges sávban az alap kiinduló színek látszanak, a tőle balra lévő négyzetben pedig ennek a kiválasztott árnyalatai találhatók. Ezek között kattintással tudunk választani. Ezen felül még tudjuk a színek áttetszőségét állítani az „*Opacity*” címke melletti csúszkával vagy a mellette lévő számbeviteli mezővel százalékban megadva. A másik módja a színek megadásának a paramétereik alapján lehetséges. *Web* fülnél manuálisan megadható a színkódja. *RGB* fülnél a szín vörös, zöld és kék attribútumait lehet megadni a csúszkával vagy a beviteli mezőkkel, ahol 0 és 255 közti értékek vannak behelyettesítve. A „*HSB*” fülnél a „*Hue*” opció állítása az alapkiinduló színt változtatja meg a csúszka mozgatásával vagy az értékek beírásával, a „*Saturation*” beállítás felel a szín élénkségeért és a „*Brightness*” pedig a szín fényerősségeért, amiket százalékos értékek beírásával vagy a csúszka mozgatásával befolyásolhatóak. Az ablak bal alsó sarkában lévő „*Use*” gombjával tudjuk beállítani az alakzatnak a színét véglegesen, továbbá a „*Save*” gombbal le is lehet menteni későbbi használatra, ez a program bezárásáig elérhető lesz, a „*Cancel*” gombbal pedig eldobhatjuk a változtatásokat.

Ezekon felül még két darab gomb tartozik minden kreált alakzathoz. A „*Delete*” gombbal törölni tudjuk az alakzatot, a mellette lévő gombbal pedig ki-be tudjuk kapcsolni a láthatóságot a z- és w-síkokon.

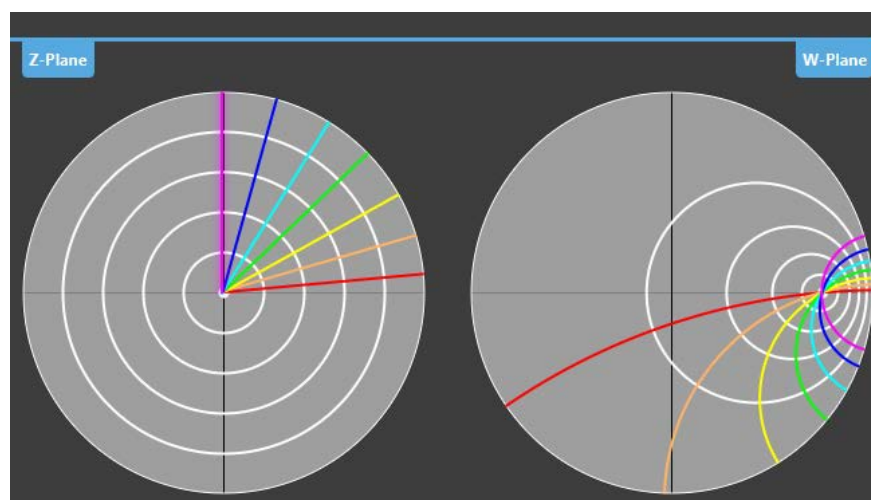
Alakzatok attribútumai

Minden alakzatnak van egy kiinduló- vagy középpontja. Ezeket az x_0 , illetve y_0 mezők jelölik. Ezen felül a szakasznak vannak végpontjai melyekért az x és y mezők felelősek. Kör és a körív rendelkezik sugár attribútummal, amit az r mezők állítanak be, továbbá a körív még rendelkezik egy központi szöggel, illetve egy kiindulási szöggel, melyeket az „Angle” és „Start” beviteli mezőkkel lehet változtatni. Ezeket a szögeket fokban kell megadni. Erre egy példát az alábbi ábrán láthatunk.

6. ábra: Példa az alakzatok attribútumaira

Fő felület

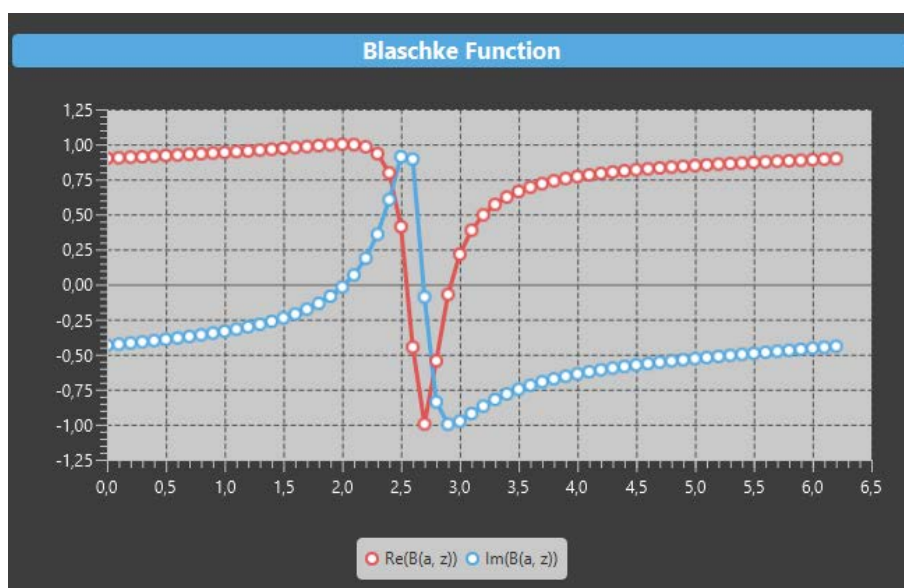
Ebben a szoftverben a *Blaschke*-függvények $\overline{\mathbb{D}} \rightarrow \overline{\mathbb{D}}$ leképezése van ábrázolva, ahol a z -sík az értelmezési tartomány és a w -sík az értékkészlet, azaz a felhasználó által létrehozott alakzatok a z -síkon fognak megjelenni, míg a *Blaschke*-függvény által transzformált alakzatok a w -síkon, ahogy ez az alábbi ábrán is látszódik.



7. ábra: Alakzatok és képük $a = -0,75$ paraméter esetén

Itt szépen látszódik, hogy az egyeneseket és a köröket, milyen alakzatokba viszi át egy *Blaschke*-függvény.

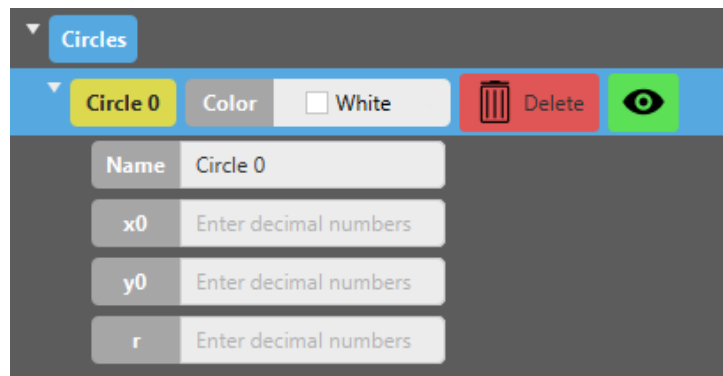
Komplex racionális törtfüggvényeknél a \mathbb{T} -n felvett értékeivel foglalkozunk, illetve legtöbbször elkülönítve ábrázoljuk a valós- és képzetes részt. A komplex számok argumentuma, azaz $[0, 2\pi)$ intervallum és a \mathbb{T} között a $\varphi \mapsto e^{i\varphi}$ hozzárendelés segítségével lehet szemléltetni. A *Blaschke*-függvény a paraméterének megadása után a „*Blaschke function*” címke alatt fog látszódni a leképezés, ahol a vízszintes tengely az $[0, 2\pi)$, piros színnel a valós-, kék színnel a képzetes rész jelenik meg, amit a 8. ábra is szemléltet.



8. ábra: *Blaschke*-függvény szemléltetése $a = -0.75 + 0.4i$

Alakzatok beállítása a z -síkon

Az alakzatok változtatása és mozgatása az egér segítségével is lehetséges a felhasználó számára. Először is ki kell jelölni a kívánt alakzatot a fa nézetben belül kattintással, ami ki lesz emelve a z -síkon a már korábban felvett alakzatok között. A alábbi ábrán látszódik, hogy néz ki egy ilyen kijelölt alakzat.

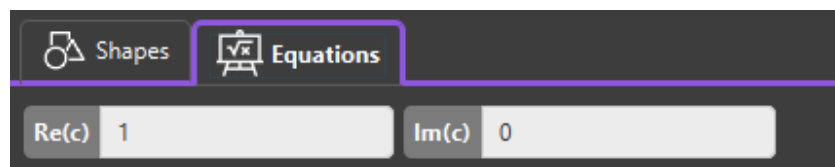


9. ábra: Alakzat kijelölése

Miután kijelölte a felhasználó az alakzatot, a Shift billentyű és a bal egérgomb nyomva tartásával, illetve az egér mozgatásával tudjuk mozgatni az alakzatot, ami az x_0 és y_0 attribútumok folyamatos átállítását eredményezi. A Shift billentyű lenyomása nélkül a bal egérgomb kattintásával az x_0 és y_0 pontokat lehet beállítani mozgatas nélkül. Azonban a szakasz, körív és a kör alakzatoknál a bal egérgomb nyomva tartásával és az egér mozgatásával különböző adatokat is lehet változtatni. Szakasz esetében a végpontot lehet ily módon megadni, körívnek a szögét, körnek pedig a sugarát.

Equations fül

Az egyenlet konstans tényezőjét az alábbi beviteli mezőkben tudjuk beállítani. Itt a $Re(c)$ a valós rész és $Im(c)$ a képzetes rész.



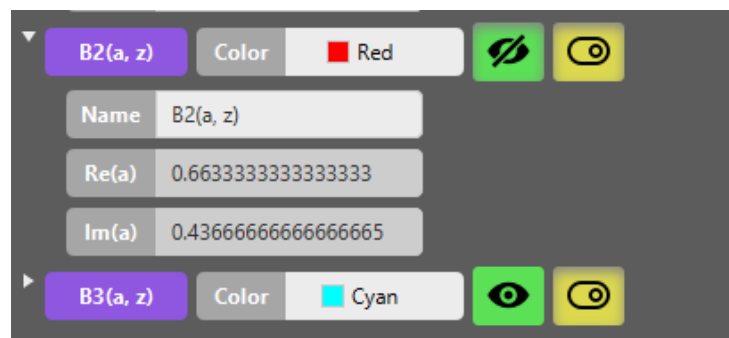
10. ábra: Konstans megadása

Ahogy korábban írtam, az alkalmazás az egy, kettő és három tényezős *Blaschke*-szorzatokból felírt egyenletek megoldására fókuszál.

Blaschke-függvények megadása

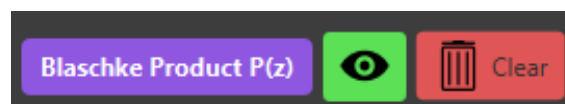
A fa nézetben láthatók a *Blaschke*-függvények adatai. Ezeknek is meg lehet változtatni a nevét, ami alapértelmezetten „B sorszám (a, z)” egybeírva. A felhasználó az α paramétert beállíthatja explicit szám szerint, a valós- és képzetes részének megadásával, vagy az egységkörön való kattintással (később részletesebben írok erről). Ezen felül tartozik két gomb mindegyik *Blaschke*-függvényhez, illetve egy színválasztó az

α paraméter pontjához az egységkörön. Az első gomb a függvény láthatóságát állítja, a második pedig a *Blaschke*-szorzat tényezőjeként adja hozzá. Alapértelmezetten minden függvény látható és egyik sincs hozzáadva a *Blaschke*-szorzathoz, azaz egységnek veszi a tényezőket, tehát ha egy darab függvéynél van ez a gomb bekapcsolva, akkor egy lineáris egyenletet fog megoldani a program, ha kettő darab, akkor egy másodfokú egyenletet, ha mindegyiknél, akkor egy harmadfokút. Hasonlóan, mint az alakzatok fa nézeténél, itt ugyanolyan módon össze lehet csukni a függvények adatait. Ezt demonstrálja az alábbi ábra.



11. ábra: Blaschke-függvények az egyenleteknél

Blaschke-szorzat is egyben egy *Blaschke*-függvény, amit szintén ábrázol a program, így szintén lehet állítani a láthatóságát. Emellett található még egy „Clear” gomb, ami kinullázza a függvények α paraméterét. Ezeket a gombokat mutatja a 12. ábra.



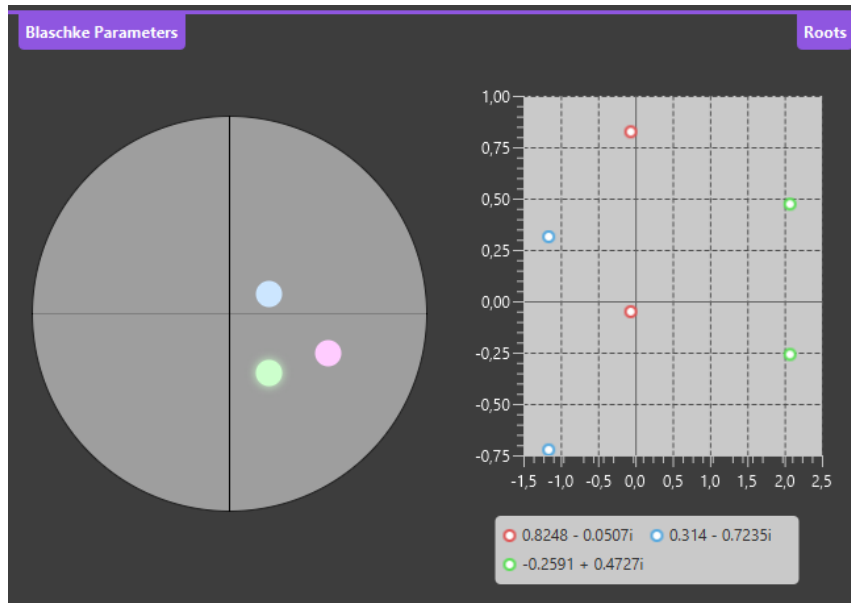
12. ábra: Blaschke-szorzat

Fő felület

A fő felületen a *Blaschke*-függvények α paraméterei megjelennek az egységkörön is. A korábban említett színválasztó ezeknek a pontoknak a színét változtatja meg. Ezeket a pontokat a bal egérgomb kattintásával is be lehet állítani, illetve, ha nyomva tartjuk, akkor az egér mozgásával lehet mozgatni ezeket. Ehhez ki kell választania a felhasználónak egy *Blaschke*-függvényt a fa nézetben, hasonlóan, mint az alakzatoknál.

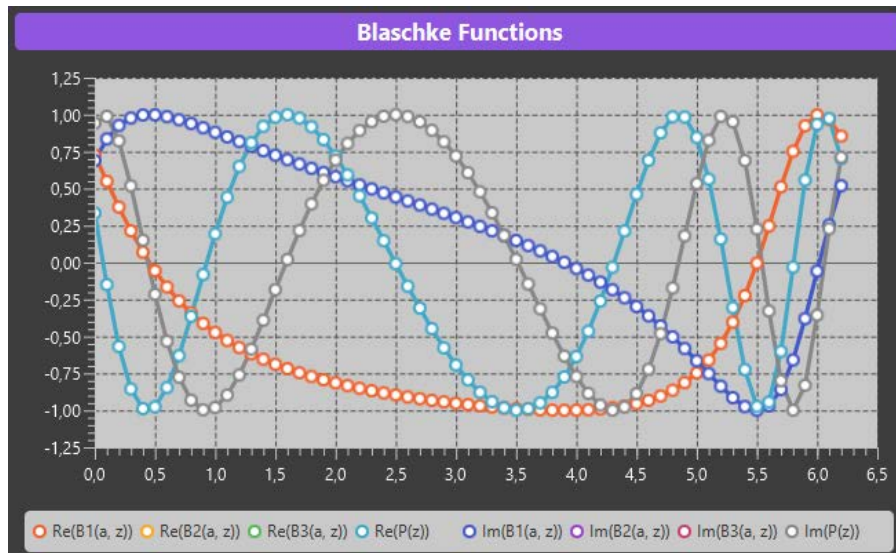
Az egységkör mellett pedig az egyenletek megoldásait láthatjuk ábrázolva és számszerűen is tízezredekre kerekítve. Itt a vízszintes tengely a komplex számok

argumentumát jelöli, az megoldások értékét pedig a függőleges tengely mutatja külön-külön, de egy színnel a valós- és képzetes részt. A 13. ábra demonstrál erre egy példát.



13. ábra: Háromtényezős Blaschke-szorzat megoldása $c = 0.2 + 0.3i$ konstansra, $a_n = ((0.5 - 0.2i), (0.2 - 0.3i), (0.2 + 0.1i))$ paraméterekkel

A fenti felület alatt láthatók a Blaschke-függvények és a -szorzat képei. Itt a grafikon alatti jelmagyarázat mutatja, hogy melyik függvényhez, mely színű görbe tartozik, amit az alábbi ábra mutat.



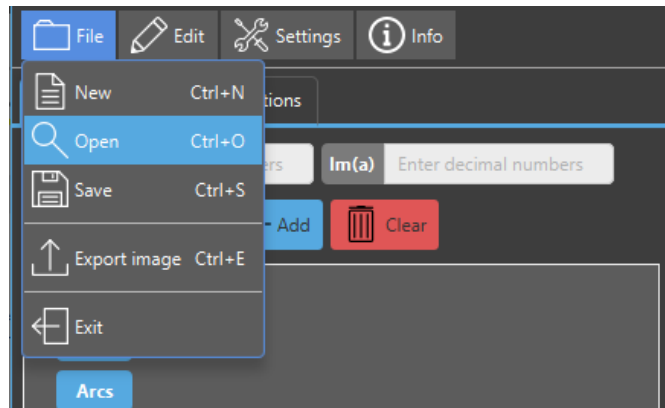
14. ábra: Előző egyenletnél a $\mathcal{B}_{a_0}(z)$ és $\mathcal{P}(z)$ képe

A függvények láthatósága megadásával a felhasználó kedvére kiválaszthatja mely függvények képét szeretné megtekinteni.

Menüsáv

File menüpont

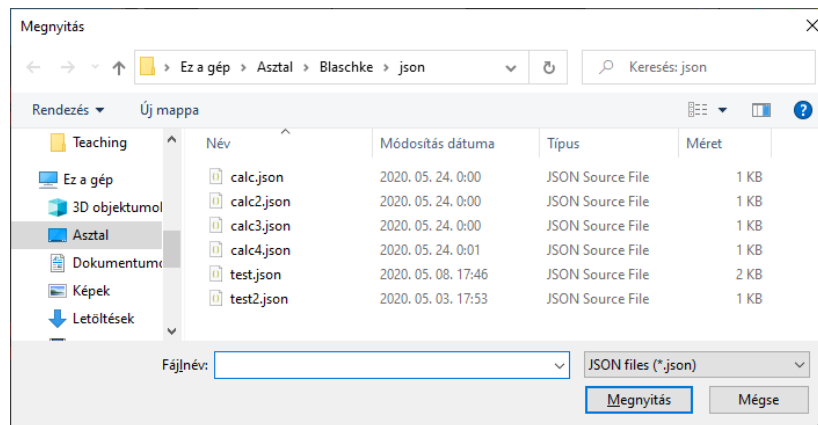
Az első menüpont, amit láthatunk az a „File” menüpont, amire kattintva lenyitódik az általános menü, ahol egérmozgatással vizuálisan látszódik épp melyik almenüpont van kijelölve. Itt több almenüpontot láthatunk, amit a 15. ábra be is mutat.



15. ábra: „File” menüpont

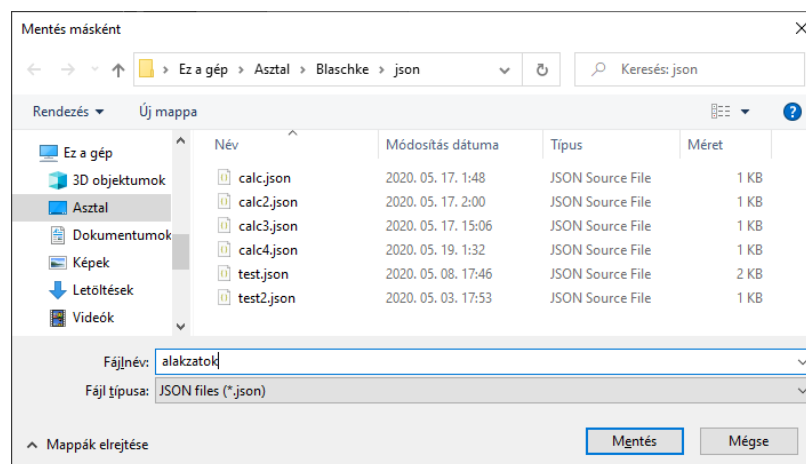
A „New” almenüpont a hozzáadott alakzatokat törli ki visszavonhatatlanul a „Shapes” fülön, illetve az „Equations” fülön a Blaschke függvények paramétereit visszaállítja az alapértelmezett paraméterekre, ami (0,0). Más szóval olyan, mint ha a két fül „Clear” gombját nyomná meg a felhasználó egy friss kezdő állapotot elérve. Ez a „Ctrl+N” billentyűkombinációval is elérhető.

Az „Save” almenüpont a hozzáadott alakzatokat menti ki JSON formátumban, ami egy mentés ablak felugrását váltja ki, ahogy ezt a 16. ábra is mutatja. Ezt a „Ctrl+S” billentyűkombináció szintűgy kiválthatja. Az ilyen lementett JSON fájlokat pedig az „Open” almenüponttal lehet megnyitni, ami szintén egy ablak felugrását eredményezi megnyitás névvel.



16. ábra: Példa a JSON fájl megnyitására Windows-on

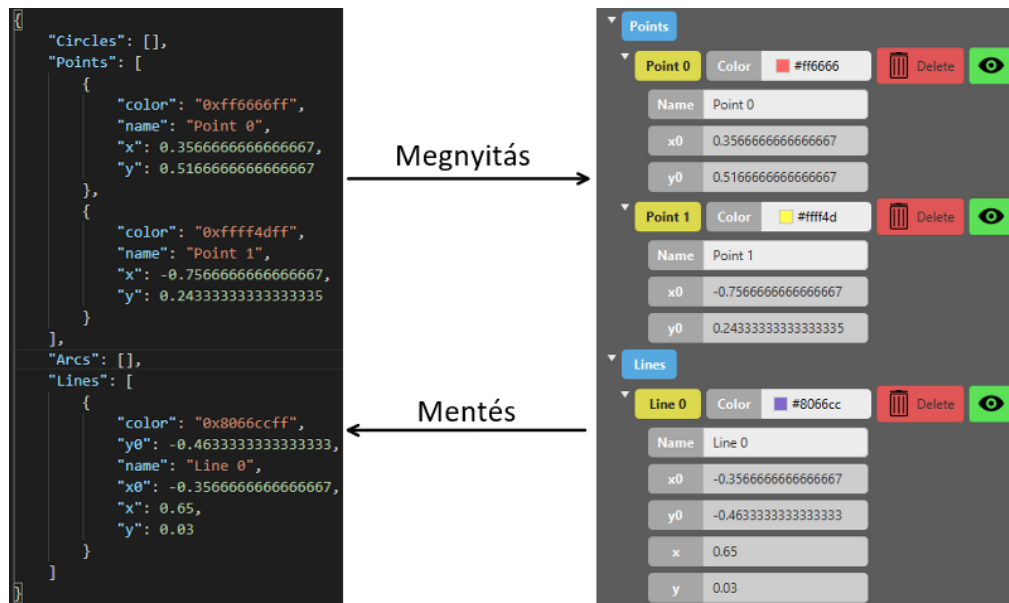
Itt a megszokott módon tud a felhasználó navigálni a lemezei és a mappái közt, és dupla kattintással, vagy a fájl kijelölése után a „Megnyitás” nevű gombra kattintva tudja megnyitni a kívánt fájlt, amiből a program egyből betölti az alakzatokat. A fájl választó automatikusan JSON fájlformátumra szűr. A Mentés esetében hasonló az eljárás és az ablak is, annyi különbséggel, hogy a felhasználónak meg kell adnia a fájl nevét, utána pedig a „Mentés” gombra kell kattintani. Mindkettő ablaknál a „Mégse” opcióra kattintva meg lehet szüntetni a folyamatokat, ahogy ez az alábbi képen is látható.



17. ábra: Példa az alakzatok mentésére JSON fájlba Windows-on

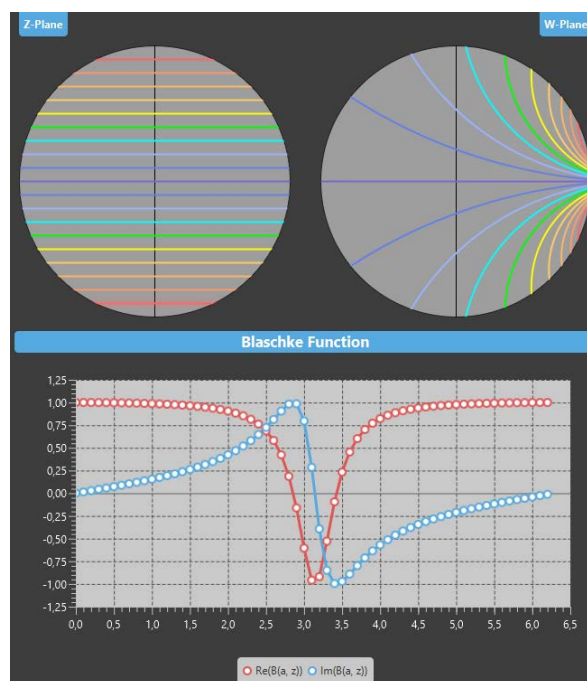
A mentés az alakzatok attribútumait menti le a JSON fájlba, azaz a nevüket, színüket és a geometriai tulajdonságaikat, megnyitáskor pedig ennek a folyamatnak a fordítottja történik. Ezeket a JSON fájlokat bármilyen egyszerűbb szövegszerkesztővel meg lehet nyitni és módosítani, például: Jegyzettömb, Notepad++, stb.

A 18. ábra röviden be is mutatja a mentés és a megnyitás folyamatát.



18. ábra: Megnyitás és mentés folyamata

Az „Export image” nevű almenüpont a „Shapes” fül fő felületéről készít képet, amit PNG formátumban lehetősége van a felhasználónak lementeni. Ezt a „Ctrl+E” gyorsbillentyű paranccsal ugyanúgy elő lehet idézni. Ennek a felugró ablaknak a megjelenése teljesen megegyezik a mentéssel, annyi különbséggel, hogy itt PNG formátumokra fog szűrni a fájlkezelő. Ezzel az opcióval az alakzatok transzformációjáról könnyedén képet lehet készíteni külső programok, vagy az operációs rendszer képmetszője nélkül. Egy ilyen exportált kép látszódik az alábbi ábrán.

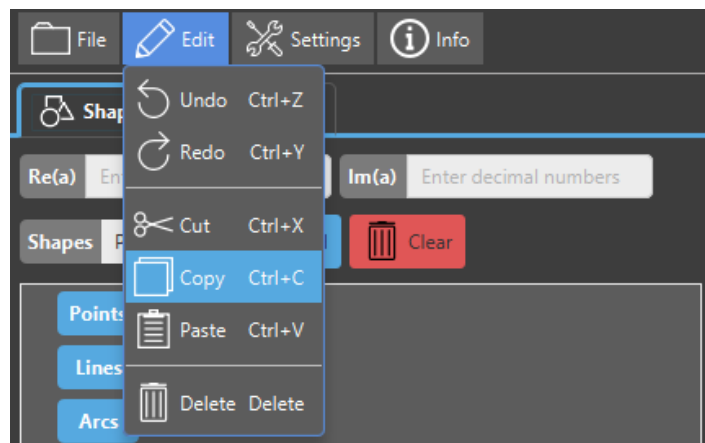


19. ábra: Exportált kép $a = -0.75$ paraméter esetén

Az „Exit” menüpont az ablakon lévő bezárás gombjával ekvivalens működést vált ki. Ennek hatására felugrik egy ablak, ahol megkérdezi a felhasználótól, hogy biztosan ki szeretne-e lépni, és ha igen, akkor le tudja menteni az alakzatokat JSON formátumban. Az ablakon belül három gomb található: „Save”, „Discard” és a „Cancel”. Az első gomb segítségével lehet lementeni a fájlokat, ami után bezáródik a program. A második gomb a mentés nélküli kilépés opciója, végül az utolsó gombbal van lehetőség visszatérni a programba.

Edit menüpont

A második menüpont, amit láthatunk az „Edit”, azaz a szerkesztés menüpont, melyekben az alábbi almenüpontok vannak.



20. ábra: „Edit” menüpont

Ezek a műveletek a „Shapes” fülhöz tartoznak, ezen belül a fa nézetben lévő alakzatokhoz. A „Delete” opció, amit a „Delete” billentyű lenyomásával is elő lehet idézni, az az alakzatok törléséért felel a fa nézetben. Tulajdonképpen az alakzatok mellett lévő „Delete” gombra való kattintással egyezik meg, annyi különbséggel, hogy előbb ki kell jelölni az alakzatot, mielőtt törölnénk.

A „Copy”, azaz a másolás menüpont a „Ctrl+C” gyorsbillentyű kombinációval is kiváltható. Ha a felhasználó kijelöl egy alakzatot és ezt a műveletet hajtja végre, akkor ennek az alakzatnak a JSON formátuma a vágólapra kerül. A „Cut” nagyon hasonló ehhez, annyi különbséggel, hogy ez kivágja onnan az alakzatot, azaz kitörli és a JSON alakját vágólapra helyezi. Ez a „Ctrl+X” kombinációval is használható.

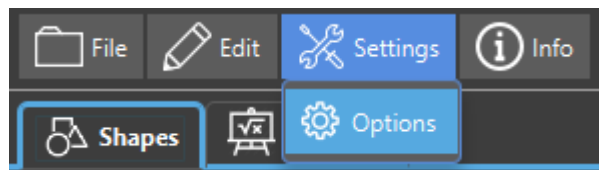
Ezekhez erősen kapcsolódik a „Paste” opció, ami a vágólapon lévő alakzat JSON formátuma alapján létrehozza az alakzatot és beilleszti. Így akár egy JSON fájlból egy

alakzatot is könnyedén hozzá lehet adni a fa nézethez. Ennek a funkciónak a „Ctrl+V” a gyors kombinációja.

Az „Undo” és a „Redo” funkciók az alakzatokkal való tevékenységek visszavonását, illetve ezeknek újra végrehajtását teszi lehetővé. Ezek a „Ctrl+Z” és a „Ctrl+Y” billentyűkombinációval egyaránt kiválthatóak.

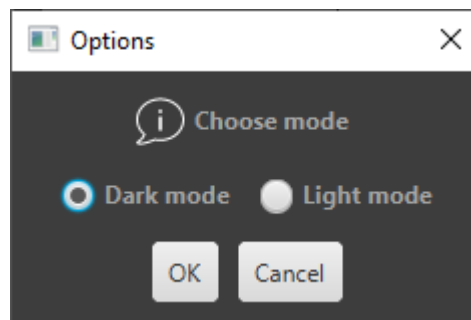
Settings menüpont

A „Settings” menüpontban érhetők el a beállítások. Itt az „Options” lehetőséget választva ugrik fel egy ablak, amit a 21. ábra mutat.



21. ábra: Settings menü

A felugró ablakban a felület témáját lehet megváltoztatni a rádiógombok segítségével, amit az alábbi képen láthatunk. Az alapértelmezett mód a sötét, amit az eddigi képek ábrázolnak.



22. ábra: Felület témájának kiválasztása

Info menüpont

Az „Info” menüpontban két opció kerül a felhasználó elé. Egy „Help” menüpont, ami egy dokumentumot nyit meg, mely segítséget nyújt a program használatához. Az „About” almenüpontra kattintva egy ablak ugrik fel, melyben általános információ található az alkalmazásról.

Egyebek

Hiba ablakok

A korábban említett felugró ablakok mellett a lehetséges hibák történése is kiválthat egy felugró ablakot, például JSON fájl beolvasáskor, ha hibás JSON formátumot ad meg a felhasználó, vagy a súgó dokumentum megnyitása sikertelen.

Állapotsor

Az alkalmazás legalján lévő színes sávon, azaz az állapotsor jobb oldalán, az egérmutató koordinátái láthatók a z-síkon, ha a „*Shapes*” fülön tartózkodik a felhasználó. Ha az „*Equations*” fülön, akkor pedig az egységkör koordinátái.

A bal oldalán pedig a beviteli mezőkkel kapcsolatos hibák kerülnek kiírásra például, ha betűket adunk meg egy kör sugarának.

FEJLESZTŐI DOKUMENTÁCIÓ

Az alkalmazás JavaFX keretrendszer eszközeivel készült, és Java nyelvben íródott különböző könyvtárak kiegészítésével. IntelliJ IDEA 2020.1.1 (Ultimate Edition) fejlesztői környezetet használtam a kivitelezéshez. A program készítése közben törekedtem a nyelvek konvencióinak betartására, objektum-orientált programozási nyelvek szabályainak betartására, és az átlátható, öndokumentáló kód készítésére, redundanciát kerülve. Ezeket az eszközöket a következő alfejezetben fogom részletezni.

Fejlesztői eszközök

Fejlesztői környezet

Ahogy korábban említettem, a program megvalósítását IntelliJ IDEA 2020.1.1 (Ultimate Edition) fejlesztői környezetben végeztem. Ennek az eszközei ideális környezetet teremtenek a fejlesztés kivitelezésében, így a további fejlesztéseket is ajánlott ebben végezni, ezen felül magas szintű támogatást nyújt teszteléshez, illetve a külső könyvtárakhoz egyaránt.

Felhasznált nyelvek

Java

A Java egy általános célú objektumorientált programozási nyelv, aminek az előnye, hogy bármilyen operációs rendszeren lehet futtatni, ami támogatja a Java-t egyéb fordítási lépések nélkül, így a fejlesztők a programalkotásra tudnak fókuszálni. A nyelv szintaxisban nagyon hasonló a C, illetve C++ programozási nyelvekhez. Ellenben az előbb említett nyelvekkel, fordításkor a Java forráskód bájtkóddá alakul, amit a Java virtuális gép (röviden JVM) futtat, így elérve a platformfüggetlenséget. Az alkalmazás fejlesztése JDK 13-mas verziójával zajlott.

FXML

Az FXML egy XML általános célú leíró nyelv variánsa, amit az Oracle Corporation hozott létre. Ennek a segítségével sokkal egyszerűbben létre lehet hozni a JavaFX alkalmazások grafikus felhasználói felületének a statikus részét és vázát.

CSS

A CSS (*Cascading Style Sheets*) egy stílusleíró nyelv, amelyben bármilyen XML alapú elemek stílusát le lehet írni, így az FXML fájlban létrehozott elemek megjelenését is kényelmesen lehet változtatni.

Keretrendszerek

JavaFX

A JavaFX egy nyílt forráskódú szoftver platform, aminek a segítségével lehet modern, hardveresen gyorsított grafikus felhasználói felületet készíteni, ami platformfüggetlen. Ezzel a keretrendszerrel lehetőség van asztali, webes, illetve mobil alkalmazások készítésére is. Az applikációt JavaFX 13.0.2-es verziójában fejlesztettem.

JUnit

A JUnit egy egységteszt keretrendszer a Java-hoz. Ennek a célja, hogy automatizált egységtesztekkel lehessen biztosítani, a függvények, metódusok és az osztályok helyes működését. A programban a JUnit 4.13-mas verzióját használtam.

JSON-Simple

A JSON-Simple egy Java eszköztár a JSON (*JavaScript Object Notation*) formátum használatához. Ennek a segítségével könnyedén tudunk kódolni és dekódolni JSON formátumú szöveget.

Program rétegei

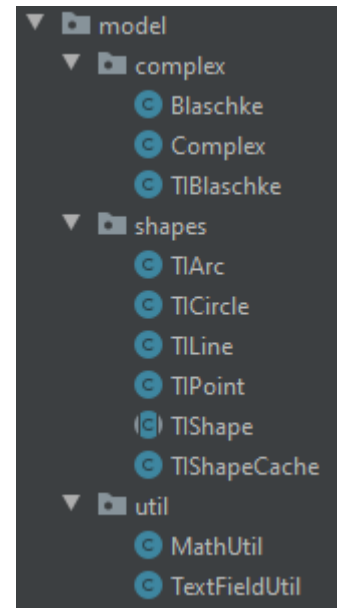
A program fejlesztésekor törekedtem a Model-View-Controller (röviden MVC) architektúra követésére, azaz a modell, nézet és a vezérlők elkülönítésére. Ennek a haszna, hogy ha az egyik réteget változtatni szeretné a fejlesztő, akkor a többi réteget ne kelljen átírni. A modell rész teljesen független a felhasználói felülettől, így különböző nézeteket lehet kapcsolni ugyanahhoz a modellhez. A vezérlő jellemzően ezt a két réteget köti össze. Ez a felhasználótól kapott bemeneteket dolgozza fel, ami kivált eseményeket a modellben, ami alapján változik a nézet. Ezt a mintát a csomagok létrehozásában is követtem.

Modell csomag

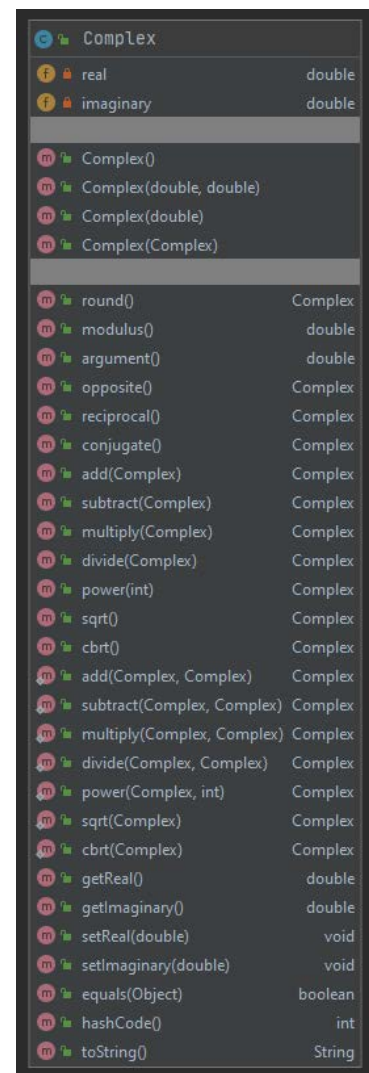
A „*model*” csomag az alkalmazás nézettől független osztályait és függvényeit tartalmazza. Ennek a csomagnak a felépítését a [23. ábra](#) mutatja. A „*complex*” csomagban találhatóak a komplex- számokkal és függvényekkel kapcsolatos osztályok. A „*shapes*” csomag az alakzatok osztályait foglalja magába, míg az „*util*” csomag olyan függvények osztályait tartalmazza magában, amiket többször is felhasznál a program.

Complex csomag

A „*Complex*” osztály a komplex számokat és a velük való műveleteket implementálja. A valós- és képzetes részzel definiáljuk az osztály objektumait. Ehhez tartozik egy üres konstruktor, kétparaméteres konstruktor, ahol az előbb említett valós- és képzetes részt inicializáljuk példányosításkor, illetve egy egyparaméteres konstruktor, ahol a komplex szám argumentumát adjuk meg, végül egy másoló konstruktor is. Az alapl műveleteknek, hatványozásnak és négyzet- illetve köbgyökvonásnak mind két változata van. Vannak az egyparaméteresek, melyeket a példányon keresztül lehet meghívni, amik megváltoztatják az objektumot, illetve vissza is térnek vele, ezáltal láncolni is lehetséges ezeket. A másik opció a kétparaméteresek. Ezek statikus függvények, azaz csak az osztályon keresztül lehet meghívni őket. Ennek hatására a paraméterben megadott komplex számokkal elvégzi a műveleteket és az eredménnyel visszatér. A gyökvonások az első gyököt adják vissza a négyzet- és a köbgyöknél is. A műveleteken kívül metódusokkal le tudjuk kérdezni egy komplex szám hosszát, argumentumát, konjugáltját,



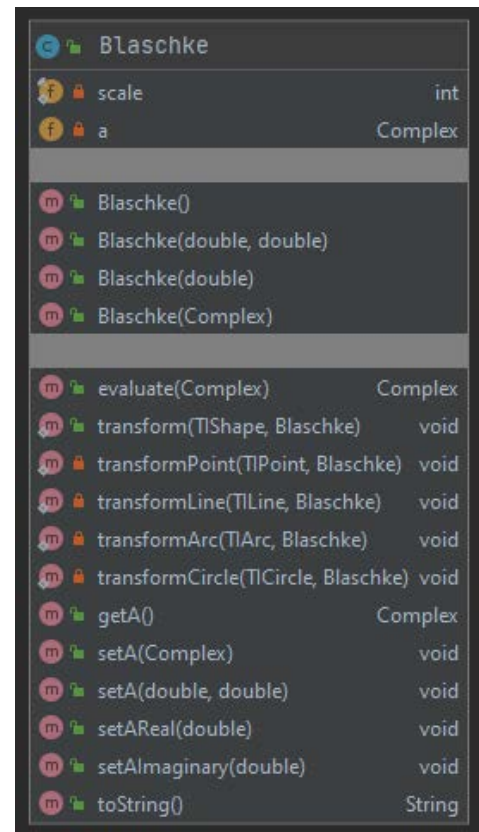
23. ábra: Modell csomagjai



24. ábra: Komplex számok osztálydiagram

reciprokát, illetve az ellentettjét. A [24. ábra](#) mutatja ennek az osztálydiagramját.

A „*Blaschke*” osztály az α paraméterű *Blaschke*-függvényeket reprezentálja, ahol ez az α paraméter adattagja is egyben. Ehhez a komplex számok osztályával megegyező konstruktorok tartoznak. Ez az osztály felel az alakzatok transzformációjáért a „*transform*” nevű metódussal, továbbá $\mathcal{B}_\alpha(z)$ helyettesítési értékek kiszámolásáért az „*evaluate*” nevű függvénnyel. A teljes osztálydiagrammot a [25. ábra](#) mutatja.



25. ábra: *Blaschke* osztálydiagram

Minden alakzathoz külön transzformáló függvény tartozik, így a „*transform*” függvény a kapott paraméter alapján eldönti, hogy milyen alakzatról van szó, ezután pedig meghívja a megfelelő függvényt az alakzatra. Ezeket a transzformációs függvényeket máshonnan kívülről nem hívjuk meg, így privát az elérhetősége, azaz csak az osztályon belül lehetséges meghívni. Továbbiakban ezeket a transzformációs algoritmusokat fogom részletezni. Fontos leszögezнем, hogy mivel itt a *Blaschke*-függvényeket $\overline{\mathbb{D}}$ -n vizsgáljuk és az alkalmazásban lévő z -sík valójában 150 egység sugarú, így minden transzformáció előtt az adatokat le kell kicsinyíteni, transzformáció után pedig felnagyítani a várt eredmény érdekében, erre szolgál a „*scale*” statikus adattag.

A pontokat komplex számoknak tekintve, a *Blaschke*-függvénybe behelyettesítve kapjuk meg a képét. Ennek a számításnak a kódjá látható az [alábbi](#) ábrán.

```
private static void transformPoint(TIPoint tiPoint, Blaschke blaschke) {
    Circle zPoint = tiPoint.getZShape();
    Circle wPoint = tiPoint.getWShape();
    Complex center = blaschke.evaluate(new Complex(
        real: zPoint.getCenterX() / scale, imaginary: zPoint.getCenterY() / scale
    ));
    wPoint.setCenterX(center.getReal() * scale);
    wPoint.setCenterY(center.getImaginary() * scale);
}
```

26. ábra: Pont transzformációja

A szakaszokat a *Blaschke*-függvények általánosságban körívekbe viszik át, így a szakaszoknak a kezdő-, felező- és végpontját eltranszformálva kapjuk meg azt a három pontot, amire fel lehet rajzolni a három ponton átmenő kört. Itt azonban azokra a külön esetekre is figyelni kell, amikor egy egyenes jön létre. Ennek egy részét mutatja az alábbi kód.

```
double midX = (zLine.getStartX() + zLine.getEndX()) / 2;
double midY = (zLine.getStartY() + zLine.getEndY()) / 2;
Complex start = blaschke.evaluate(new Complex(
    real: zLine.getStartX() / scale, imaginary: zLine.getStartY() / scale
));
Complex mid = blaschke.evaluate(new Complex(
    real: midX / scale, imaginary: midY / scale
));
Complex end = blaschke.evaluate(new Complex(
    real: zLine.getEndX() / scale, imaginary: zLine.getEndY() / scale
));
Shape shape = MathUtil.getCircleFrom3Points(start, mid, end, canBeLine: true);
if (shape instanceof Line) {
    tiline.setWShapeToLine();
    Line line = (Line) shape;
    Line wLine = (Line) tiline.getWShape();
    wLine.setStartX(scale * line.getStartX());
    wLine.setStartY(scale * line.getStartY());
    wLine.setEndX(scale * line.getEndX());
    wLine.setEndY(scale * line.getEndY());
} else {
    tiline.setWShapeToCircle();
    Circle circle = (Circle) shape;
    Circle wCircle = (Circle) tiline.getWShape();
    wCircle.setCenterX(scale * circle.getCenterX());
    wCircle.setCenterY(scale * circle.getCenterY());
    wCircle.setRadius(scale * circle.getRadius());
}
```

27. ábra: Szakasz transzformációja kódrészlet

A köríveket körívekbe, a köröket pedig körökbe viszik át a *Blaschke*-függvények. Itt is az eredeti alakzatokon választott három pont segítségével rajzolja ki a transzformált képet a program.

A köríveknél ez a három pont a körív kezdőpontja, a körív felezőpontja és a végpontja. Ezeket a pontokat használja a program a számításhoz, amit a 28. ábra mutat.

```
Complex p1 = blaschke.evaluate(new Complex(
    real: (zCenterX + zR * Math.cos(Math.toRadians(zStartAngle))) / scale,
    imaginary: (zCenterY - zR * Math.sin(Math.toRadians(zStartAngle))) / scale
));
Complex p2 = blaschke.evaluate(new Complex(
    real: (zCenterX + zR * Math.cos(Math.toRadians(zStartAngle + zAngle / 2))) / scale,
    imaginary: (zCenterY - zR * Math.sin(Math.toRadians(zStartAngle + zAngle / 2))) / scale
));
Complex p3 = blaschke.evaluate(new Complex(
    real: (zCenterX + zR * Math.cos(Math.toRadians(zStartAngle + zAngle))) / scale,
    imaginary: (zCenterY - zR * Math.sin(Math.toRadians(zStartAngle + zAngle))) / scale
));
Circle circle = (Circle) MathUtil.getCircleFrom3Points(p1, p2, p3, canBeLine: false);
double wCenterX = scale * circle.getCenterX();
double wCenterY = scale * circle.getCenterY();
double wR = scale * circle.getRadius();
```

28. ábra: Kör transzformációja kódrészlet

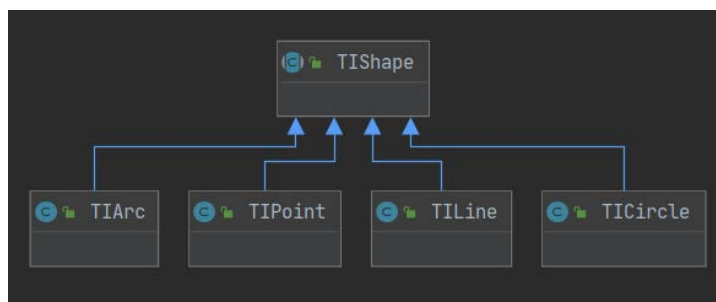
A köröknél pedig bármelyik tetszőleges három pont lehet ez, a program a vízszintes átmérő két végpontja, illetve a függőleges átló egyik pontja alapján számol, ahogy ezt az alábbi kódrészlet is mutatja.

```
Complex p1 = blaschke.evaluate(new Complex( real: (r + centerX) / scale, imaginary: centerY / scale));
Complex p2 = blaschke.evaluate(new Complex( real: centerX / scale, imaginary: (r + centerY) / scale));
Complex p3 = blaschke.evaluate(new Complex( real: (-r + centerX) / scale, imaginary: centerY / scale));
Circle circle = (Circle) MathUtil.getCircleFrom3Points(p1, p2, p3, canBeLine: false);
wCircle.setCenterX(scale * circle.getCenterX());
wCircle.setCenterY(scale * circle.getCenterY());
wCircle.setRadius(scale * circle.getRadius());
```

29. ábra: Kör transzformációja részlet

Shapes csomag

A csomagban lévő „TIShape” a JavaFX keretrendszerben lévő „TreeItem<>” osztályból származik le, típusparaméterének a „HBox” lett kiválasztva, ugyanis a „Shapes” fülön az alakzatoknak az attribútumai a fa nézetben egy ilyen vízszintes dobozban vannak tárolva. Itt minden alakzatnak vannak eltérő attribútumai, azonban a közősek az előbb említett osztály segítségével vannak inicializálva, és majd ebből fognak leszármazni az alakzatok, ezt az 30. ábra diagramja is mutatja. Ez az osztály absztrakt, ugyanis sosem lesz példányosítva, ezen felül vannak absztrakt függvényei, amiket a leszármazottaknak kötelezőn felül kell írni. Az alakzatok közös objektumait mind ebben az osztályban fogja létrehozni a védett konstruktorával, amit a leszármazott osztályok a saját konstruktoruk legelején meg fognak hívni. Az ilyen közös adattagok közé tartoznak az alakzatok neve, törlés gombja, elrejtés kapcsolója, színválasztója, illetve a közös beviteli (név és a kiinduló pontok) mezői. A közvetlen geometriai alakzathoz tartozó számbeviteli mezőket eltároljuk, később ennek a segítségével fogjuk validálni a felhasználótól kapott bemeneteket. Tulajdonképpen minden attribútum és metódus, ami közös a leszármazott osztályokban, azokat ez az osztály tartalmazza. Ami külön említést igényel az a kijelölt geometriai alakzatok effektje. Ez egy „DropShadow” objektummal van megvalósítva, ami a színválasztó színéhez van kötve, ezáltal ahogy megváltozik egy alakzat színe, az effekt színe is meg fog változni. Ezen felül fontos megjegyezni még a JSON formátummal visszatérő „getJSON()” nevű metódust. Ez itt egy absztrakt metódus, amit minden leszármazott felül fog definiálni, és ez a függvény fog felelni az alakzatok megfelelő JSON formátumának visszatérésével.



30. ábra: Alakzatok leszármazása

Így „*TPoint*” már a pont alakzathoz specifikus tényezőket tartalmazza. A pontokat, mint alakzatokat egy négy sugarú körlemez fogja illusztrálni, a sugár a statikus végleges „*radius*” adattagon keresztül van beállítva. Ha ezt meg szeretnénk változtatni, akkor elég csak ezt az adattagot átírni a kívánt értékre. Ebben osztályban kerül sor a pontok inicializálására is, amiknek a színe hozzá lesz kötve a színválasztóhoz.

Ahogy korábban írtam, a szakaszok általánosan körívekké transzformálódnak, de külön esetekben ezek egyenesek is lehetnek, így a „*TLine*” esetében nem egyértelmű, hogy mi lesz a transzformált alakzat, emiatt ez nem a konstruktorban lesz inicializálva. Ennél fogva az alakzatok a „*Shape*” őosztállyal van deklarálva, és majd a megfelelő függvényhívással lesz létrehozva. A szakaszokat kezdő és végpontjuk alapján fogjuk definiálni, így az osztálynak a kiinduló pontjainak beviteli mezői mellett, a végpontoknak is lesz mezője, amiket a konstruktorban hozunk létre.

Körívek esetén a sugárhoz, a kezdő szöghöz és a szöghöz is tartozik egy beviteli mező, így ezek is a konstruktorban jönnek létre. Ez az alakzat ugyan mindig körív lesz, viszont $\alpha \neq 0 + 0i$ esetén, akkor „*Path*” típusú objektumként lesz létrehozva, aminek a segítségével könnyebben felrajzolhatók a körívek, mint geometriai alakzatok.

Körök esetén nincsen ilyen probléma, ugyanis mindig körökké alakulnak ezek. Az őosztálytól eltérő szövegbeviteli mező itt a sugárhoz tartozik, ami az objektum létrehozásakor inicializálódik.

Util csomag

A csomagban lévő osztályok absztraktak, ugyanis sosem lesznek példányosítva, emiatt a függvényeik mind statikusak. A „*TextFieldUtil*” nevű osztályban egyetlen függvény van, ami számbeviteli mezők validációjára szolgál. A másik, „*MathUtil*” nevű osztályban a transzformációkhoz és az egyenletek megoldásához szükséges matematikai függvények találhatók.

Ahogy már előbb említettem, legtöbb esetben az alakzatok körökké alakulnak, így három pont segítségével a transzformált köröket fel tudjuk rajzolni. Ennek a paraméterei ez a három pont, illetve egy logikai változó, ami azt jelöli, hogy az adott alakzat transzformáltja lehet-e egyenes, például szakaszok esetében. Ehhez szükséges két-két pont közötti szakasz felezőpontja. Továbbá kell a szakaszok felezőmerőlegese, amiknek a metszéspontja meg fogja adni a kör középpontját, ezután a kör sugarát pedig a középpont és bármelyik korábbi pont távolságából adjuk meg. Kódban ennek a megvalósítását a 31. ábra mutatja. Itt a kör középpontját a felezőmerőlegesek egyenletéből felírt egyenletrendszer megoldása fogja adni. Ha létezik megoldás, akkor a három ponton keresztül létezik kör, amivel vissza is tér a függvény. Ha a koordináták tömbjébe nem szám értékek kerülnek, akkor nem alkotnak kört, így egy egyenessel fog visszatérni.

```
public static Shape getCircleFrom3Points(Complex start, Complex mid, Complex end, boolean canBeLine) {
    Complex center1 = new Complex(
        real: (start.getReal() + mid.getReal()) / 2,
        imaginary: (start.getImaginary() + mid.getImaginary()) / 2
    );
    Complex center2 = new Complex(
        real: (mid.getReal() + end.getReal()) / 2,
        imaginary: (mid.getImaginary() + end.getImaginary()) / 2
    );
    double slopeNormal1 = -((mid.getReal() - start.getReal()) / (mid.getImaginary() - start.getImaginary()));
    double slopeNormal2 = -((end.getReal() - mid.getReal()) / (end.getImaginary() - mid.getImaginary()));

    double[][] A = {{slopeNormal1, -1}, {slopeNormal2, -1}};
    double[] b = {
        slopeNormal1 * center1.getReal() - center1.getImaginary(),
        slopeNormal2 * center2.getReal() - center2.getImaginary()
    };
    double[] coords = MathUtil.cramerRule(A, b);

    if (canBeLine && (Double.isNaN(coords[0]) && Double.isNaN(coords[1]))) {
        Line wLine = new Line();
        final int delta = 10;
        wLine.setStartX(start.getReal() - delta * (end.getReal() - start.getReal()));
        wLine.setStartY(start.getImaginary() - delta * (end.getImaginary() - start.getImaginary()));
        wLine.setEndX(end.getReal() + delta * (end.getReal() - start.getReal()));
        wLine.setEndY(end.getImaginary() + delta * (end.getImaginary() - start.getImaginary()));
        return wLine;
    }
    Circle wCircle = new Circle();
    wCircle.setCenterX(coords[0]);
    wCircle.setCenterY(coords[1]);
    wCircle.setRadius(MathUtil.getDistance(coords[0], coords[1], start.getReal(), start.getImaginary()));
    return wCircle;
}
```

31. ábra: Kör felírása három pontból részlet

Az előbb említett egyenletrendszerből felépít a program egy $A \in \mathbb{R}^{2 \times 2}$ mátrixot és egy b vektort. A megoldást Cramer-szabállyal végzi a program, ami ekkora méretű mátrixokra elfogadható műveletigénnyel bír. Ennek a kódját mutatja az alábbi kép.


```

public static double[] cramerRule(double[][] A, double[] b) {
    double determinant = A[0][0] * A[1][1] - A[0][1] * A[1][0];
    double x = (b[0] * A[1][1] - b[1] * A[0][1]) / determinant;
    double y = (A[0][0] * b[1] - A[1][0] * b[0]) / determinant;
    return new double[]{x, y};
}

```

32. ábra: Cramer-szabály implementálása 2x2-es mátrixra

Az egyenletek megoldásáért felelős függvények az a *Blaschke*-paramétereket, illetve a $c \in \mathbb{C}$ konstansokat kapja bemenetnek. A lineáris egyenletek megoldása megegyezik a $\mathcal{B}_{-a}(c)$ behelyettesítéssel. A másodfokú egyenletek megoldása komplex együtthatók esetén is hasonlóan történik, mint valós esetben, annyi különbséggel, hogy itt nem csak a valós gyökök keresése a fontos. A függvény előbb a másodfokú polinom együtthatóit határozza meg, ezután pedig a másodfokú formula alapján számolja ki a gyökeket.

A harmadfokú egyenletnél a $z^3 + pz^2 + qz + r = 0$ alakra hozatal után tudunk dolgozni, ezután a program Cardano-módszer alapján számolja ki a gyököket, ahol az alábbi segédváltozók vannak.

```

p = Complex.divide(p, a);
q = Complex.divide(q, a);
r = Complex.divide(r, a);

Complex A = getA(p, q, r);
Complex B = getB(p, q, A);
return getSolutions(p, A, B);

```

33. ábra: Harmadfokú megoldás kódrészlet

Az A és B segédváltozók a formula alapján a következők:

- $$A = \frac{\sqrt[3]{-2p^3 + 9pq - 27r + 3\sqrt{3}\sqrt{-p^2q^2 + 4q^3 + 4p^3r - 18pqr + 27r^2}}}{3\sqrt[3]{2}}$$
- $$B = \frac{-p^2 + 3q}{9A}$$

Ezen segédváltozók segítségével a Cardano-formula alapján a következők lesznek az egyenlet megoldásai, amiért „*getSolutions(p, A, B)*” függvény felel:

- $z_1 = -\frac{p}{3} + A - B$
- $z_2 = -\frac{p}{3} + \frac{-1 - i\sqrt{3}}{2}A - \frac{-1 + i\sqrt{3}}{2}B$
- $z_3 = -\frac{p}{3} + \frac{-1 + i\sqrt{3}}{2}A - \frac{-1 - i\sqrt{3}}{2}B$

Nézet csomag

Scenes csomag

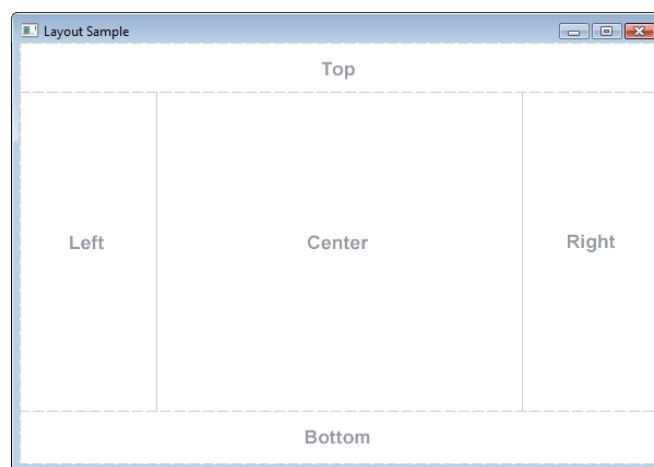
Ebben a csomagban a statikus elemeket tartalmazó FXML fájlok találhatók. A felületi elemek JavaFX-ben hierarchikusan egy fa rendszerben vannak elrendezve. Az FXML fájlok xml verziója és az importok után ezeknek a fájloknak a fejléce olvasható. Ezek tartalmazzák a megjelenés gyökér elemének típusát, a vezérlők és a stílus lapok relatív útvonalát, ezenfelül ebben látszódnak a JavaFX FXML specifikus importok.

Az ilyen importok miatt érhetőek el az alábbi keretrendszer specifikus attribútumok:

- ***fx:id*** – Megjelenési elemek azonosítója. Ennek a segítségével lehet hivatkozni az elemekre a vezérlőben. Nem összetévesztendő a sima ***id***-vel, ami segítségével a stíluslapokon lehet konkrét megjelenést biztosítani az elemeknek. Redundancia elkerülése érdekében az ***fx:id*** segítségével is tudunk hivatkozni az elemekre a CSS fájlokban.
- ***fx:define*** – Az ilyen blokkokban tudunk változókat deklarálni, amikre a \$ szimbólum segítségével tudunk hivatkozni, mint például a menüpontok ikonjára.
- ***fx:include*** – Ezeknek a blokkoknak segítségével tudunk beimportálni és összeköttetésbe hozni külön FXML fájlokban lévő nézeteket.
- ***fx:controller*** – A fejlécben ezzel az attribútummal tudunk FXML fájlokban lévő nézetekhez kontrollereket, azaz vezérlőket kapcsolni.

- ***fx:factory*** - Ennek segítségével lehet olyan objektumokat létrehozni, amiknek nincs alapértelmezett konstruktoruk, például a keretrendszerben lévő „*observableArrayList*”.
- ***fx:value*** – Ez hasonló az előzőhöz, annyi különbséggel, hogy ezzel olyan objektumokat szokás létrehozni, amikhez tartozik statikus „*valueOf(String)*” metódus, azaz egy szöveg típus alapján átalakíthatóak.

A kiinduló elemet közvetlen az ablak kapja meg, tehát ez a gyökér elem, ennek nincs szülő eleme. Ennek a szülő elemnek adunk hozzá gyerek elemeket fejlesztés során. A főelem itt egy „*BorderPane*”, aminek felépítését az alábbi kép illusztrálja.



34. ábra: *BorderPane* felépítése

A felső részben az alkalmazás menüsávja, alsó részben az állapotsora, a centerben pedig a „*Shapes*” és a „*Equations*” fülek találhatók. Ezek vannak felosztva, így választódnak el a fülek fő területe, a bal oldali fa nézetet tartalmazó felülettől. A fülek lényeges gyerek elemei a redundancia elkerülése érdekében külön FXML fájlokban vannak létrehozva. Ezen FXML fájlok hierarchikus rendszerét mutatja be a 35. ábra.


```

<TabPane fx:id="tabPane" tabClosingPolicy="UNAVAILABLE" BorderPane.alignment="CENTER">
  <Tab fx:id="tShapes" text="Shapes" graphic="$imgShapes">
    <SplitPane dividerPositions="0.4" >
      <ScrollPane SplitPane.resizableWithParent="false">
        <fx:include source="ShapesSide.fxml" fx:id="vbShapesSide"/>
      </ScrollPane>
      <ScrollPane SplitPane.resizableWithParent="true">
        <fx:include source="ShapesMain.fxml" fx:id="vbShapesMain"/>
      </ScrollPane>
    </SplitPane>
  </Tab>
  <Tab fx:id="tEquations" text="Equations" graphic="$imgEquations">
    <SplitPane dividerPositions="0.4">
      <ScrollPane SplitPane.resizableWithParent="false">
        <fx:include source="EquationsSide.fxml" fx:id="vbEquationsSide"/>
      </ScrollPane>
      <ScrollPane SplitPane.resizableWithParent="true">
        <fx:include source="EquationsMain.fxml" fx:id="vbEquationsMain"/>
      </ScrollPane>
    </SplitPane>
  </Tab>
</TabPane>

```

35. ábra: Main.fxml hierarchiája

Az alakzatok mellék FXML fájlja tartalmazza a *Blaschke*-paraméter beviteli mezőit, az alakzatok kiválasztásához a szükséges legördülő menüt, továbbá a hozzáadás- és a törlés gombját. A fa nézethez statikusan lesznek hozzáadva az alakzatokat tartalmazó fa elemek, amik alapján a megfelelő kategóriába kerülnek majd a dinamikusán létrehozott formák. A fő felületen pedig z- és a w-sík, továbbá a függvény tartalmazó vonaldiagramm kerül létrehozásra.

Az egyenletek mellék FXML fájlja a konstansok beviteli mezőjét, a *Blaschke*-szorzathoz tartozó gombokat és kapcsolókat, illetve *Blaschke*-függvények fa nézetét tartalmazza. Konkrétan nem itt lesznek hozzáadva a *Blaschke*-függvények a fa nézethez, ezek majd a vezérlőben lesznek hozzátéve, ugyanis így egyszerűbb a fejlesztés. A fő FXML fájl pedig a Blaschke-paraméterek megadásához lévő kört, a két vonaldiagrammot tartalmazza, ami a függvényekhez, és az egyenletmegoldásokhoz szükséges.

Style csomag

Ebben a csomagban található a nézethez tartozó CSS fájlok, amik az elemek megjelenését változtatja. Ezen belül külön csomagokban vannak a sötét-, illetve világos kinézethez tartozó fájlok, azonban a világos kinézetben, csak a változtatások vannak megírva a sötét megjelenéshez képest, ami az alapértelmezett is egyben. Ezen felül a két fül színeihez szükséges CSS fájlok is itt találhatók.

Windows csomag

Itt a felugró ablakok, illetve a fájlkezelések ablakához szükséges osztályok lelhetőek fel. A felugró ablakok nem átméretezhetőek, illetve a fókuszt elveszik a fő alkalmazás ablakától, míg ez nem záródik be. A fájlkezelő ablakokra ezek nem igazak, mivel ezek az operációs rendszer megfelelő ablakait hívják elő. Ezek mind absztrakt osztályok, és a statikus megjelenítő függvényükkel lehet őket meghívni. A fájlkezelő ablakok először meghívása a futtatható állomány mellett létrehoz egy „data” nevű mappát, amin belül létrejön egy „json” vagy egy „img” attól függően, hogy mentés vagy kép exportálása opcióra kattintott a felhasználó.

A fájl mentésének a függvénye egy listát vár, ami az alakzatokat tartalmazza. Ez a listában lévő alakzatoknak a JSON formátumát lekéri kategóriánként, ezeket átalakítja egy JSON objektummá, amit a megadott nevű JSON fájlba lement a „data/json” fájlba. Megnyitáskor a megadott fájlból JSON objektummá alakítja át a szöveget a program és ezt feldolgozva hozza létre az alakzatokat.

Vezérlő csomag

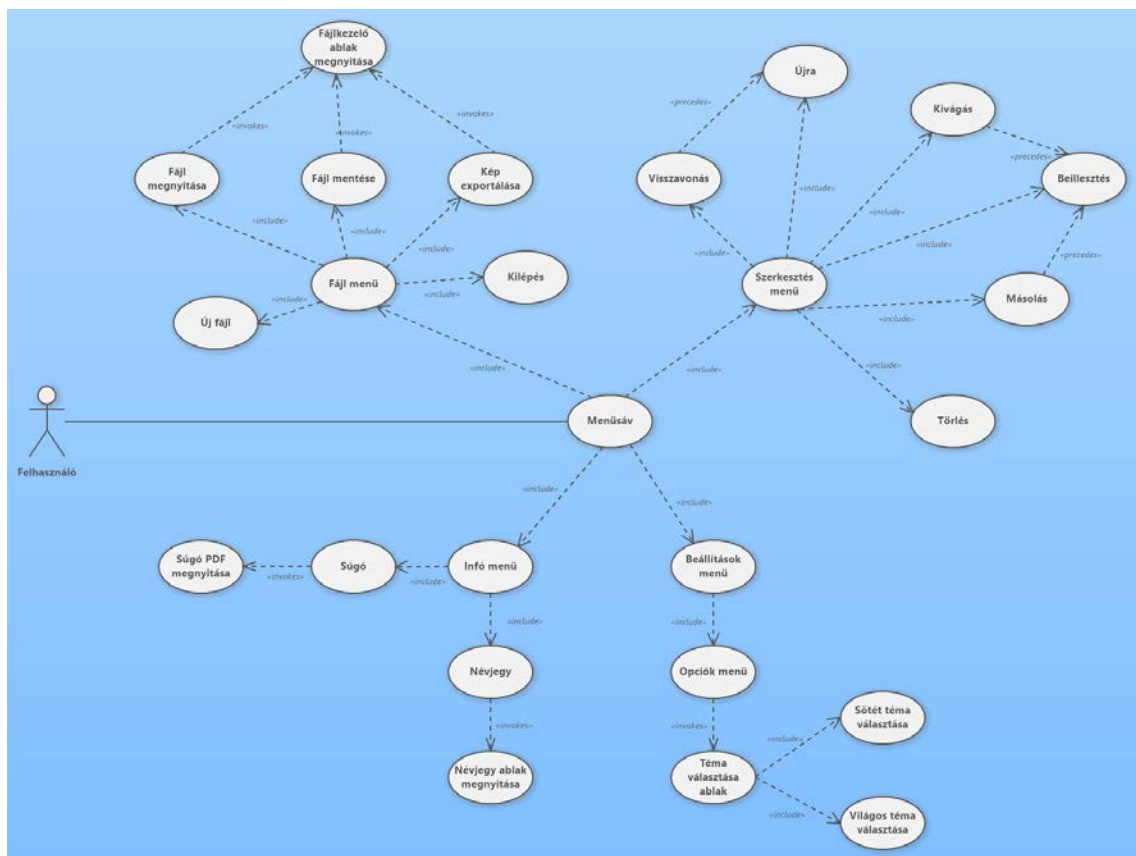
A vezérlő a felhasználótól kapott akciókat dolgozza fel, ami manipulálja a modellt, és ezáltal frissíti a nézetet, mint ahogyan ezt írtam korábban. A felhasználói eseteket egységekként diagrammal fogom ábrázolni, ami után részletezni fogom az egyes opciókat. Az ilyen vezérlők mind implementálják az „Initializable” interface-t, aminek egyetlen metódusát, az „initialize”-t írják felül, ami a program indításakor automatikusan lefut. Ami felett nem szabad eltekinteni az az @FXML annotáció. Ennek segítségével az FXML fájlok hozzáférnek a privát adattagokhoz is. Ennek a módszernek az előnye, hogy így kívülről ezek a függvények nem elérhetőek, azonban a program kritikus részei így is elérik, míg ha nyilvánosak lennének ezek a metódusok, akkor bárhonnán meg lehetne hívni őket.

Fő vezérlő

Ez a vezérlő gondoskodik a menüsáv funkcióiról, illetve további vezérlők összeköttetéséről, továbbá az állapotsoron megjelenő szövegekről. Adattagjai a főnézet alapja, ami a korábban említett „BorderPane”. Ezenfelül tartalmazza a többi nézet fő megjelenési elemét, ami mindegyik esetben egy „VBox”, továbbá ezeknek a vezérlőit.

Az „initialize” metódus kapcsolja össze a füléken belüli nézetek vezérlőit a köztes kommunikáció érdekében. Ezen felül beállítja az aktuális fő felületet a „Shapes” fülének nézetére. Ennek segítségével fogja tudni majd a program a fülönkénti stíluslapok váltását, az egységkörön lévő egérkoordináták, illetve a szövegmezők validációjakor előforduló hibák állapotsorra való tételét megvalósítani. Ennek az implementálásához szükséges egy eseményfigyelőt hozzáadni a „TabPane”-hez, azaz a füleket tartalmazó elemhez, ami azt figyel, hogy épp melyik fülön van a felhasználó.

A következőkben a fő vezérlőben látható metódusokat fogom részletezni, amihez társul egy felhasználói diagram a könnyebb átláthatóság érdekében. A fő vezérlőben a menüsávhoz kapcsolódó metódusok vannak implementálva. Ezek a funkciók láthatók az alábbi ábrán.



36. ábra: Menüsáv felhasználói diagram

Az információs menüre kattintva két opció tárul a felhasználó elé, a súgó és a névjegy. A névjegyre kattintva az „AboutWindow” osztály „display()” statikus metódusa hívódik meg, ami egy ablak felugrását váltja ki, amin az alkalmazás általános információi találhatóak meg. A súgó menüpontra kattintva az alkalmazás létrehoz egy „help.pdf” nevű

dokumentumot a futtatható állomány mellett, ha ez nem létezik, ezt a forrásállományban lévő dokumentumból másolja át. Ezután az alkalmazás automatikusan megnyitja ezt a dokumentumot az alapértelmezett PDF fájl olvasóval. Ez kivételt válthat ki, ha nincs olyan program a felhasználó számítógépén, ami meg tudná nyitni a PDF kiterjesztésű fájlokat, illetve ha olyan platformról fut a program, ahol nem elérhető a „Desktop” osztály.

A beállítások menü alatt, az opciók menüre való kattintás szintén egy felugró ablakot vált ki. Ezen a felhasználó kiválaszthatja az alkalmazás témáját. Ez a módszer kicseréli a stíluslapokat a kiválasztott téma alapján, és az aktuális módot eltárolja egy adattagként. Az alapértelmezett mód a sötét, így az adattag kezdő értéke a „Dark Mode” szöveg.

A szerkesztés menün belül a visszavonás („undo”) és az újra („redo”) menüpontok erős összefüggésben vannak. Az alakzatok hozzáadása, törlése és importálása után lehetséges ezeknek a metódusoknak a meghívása. Visszavonáskor a hozzáadott alakzatok törlődnek, a törlés opcióval eltüntetett alakzatok újból hozzáadódnak. Az újra gomb pedig a visszavont akciókat vonja vissza. Az alakzatok oldal vezérlője külön veremekben tárolja a visszavonható- és újra elvégezhető alakzatokat és műveleteiket, illetve azt, hogy milyen akció történt ezekkel. Ezek rendre az „undoCache” és a „redoCache”. Ezt segíti elő a „TIShapeCache” osztály, ami az alakzatot és az akciót tárolja el „String” formátumban. Visszavonáskor lekérdezi ettől a kontrollertől ezt a vermet, és ha nem üres, akkor kiveszi a verem tetejéről az elemet, beleteszi a „redo” vermébe, megnézi milyen művelet volt végezve az alakzattal, és meghívja az alakzatok oldal vezérlőjének az ellentétes műveletét. A „redo” esetén ugyanez történik analóg módon.

A kivágás-másolás-beillesztés hármas a szerkesztés menün megint csak erős függésben vannak, ugyanis egymás kiegészítő műveletei. Másoláskor és kivágáskor az alakzatok oldal vezérlőjétől lekérdezzük a kijelölt alakzatot, ha ez nem „null”, akkor ennek a JSON formátumát az operációs rendszer vágólapjára helyezi a program. Kivágás esetében ezután a kontrolleren keresztül ki is törlődik ez az alakzat. Beillesztéskor ennek a fordítottja történik, azaz az operációs rendszer vágólapjáról a „getString()” metódussal lekérjük a vágólap tartalmát szöveg formátumban, amit megpróbál a program JSON objektummá alakítani. Ha ez sikertelen, akkor egy hibaablak ugrik fel egy hibaüzenettel,

ami leírja, hol történt probléma az átalakítás során. Ha sikeres, akkor a JSON formátum alapján létrehozza a megfelelő alakzatot, hozzáadja a visszavonás vermébe, illetve hozzáadja az alakzatot. A törlés opció pedig lekéri a kijelölt alakzatot, és kitörli az alakzatok vezérlőjével.

A fájl menün belüli új fájl menüpont az alakzatok fül vezérlőjének hívja meg az alakzatok törléséért felelős függvényt, továbbá a visszavonás és az újra vermet kiüríti. Ezenfelül még az egyenletek fül „*clear()*” függvényét is meghívja.

A fájl megnyitása opció a „*FileWindow*” osztály „*openFileWindow()*” statikus függvényét hívja meg, ami segítségével JSON formátumból létrejönnek az alakzatok. Ez a függvény egy „*Map*” objektummal tér vissza, aminek a kulcsa az alakzat típusa szöveg formátumban, és az értéke pedig egy lista, amiben az alakzatok vannak. Ezek után az alakzatok oldal vezérlője segítségével hozzáadja ezeket.

A mentés opció az előbb említett osztály „*saveFileWindow(List<TIShape>)*” statikus metódusát hívja meg, ami az alakzatok oldal kontrollerétől lekért alakzat listát kapja meg paraméterül, amiket lement JSON formátumba.

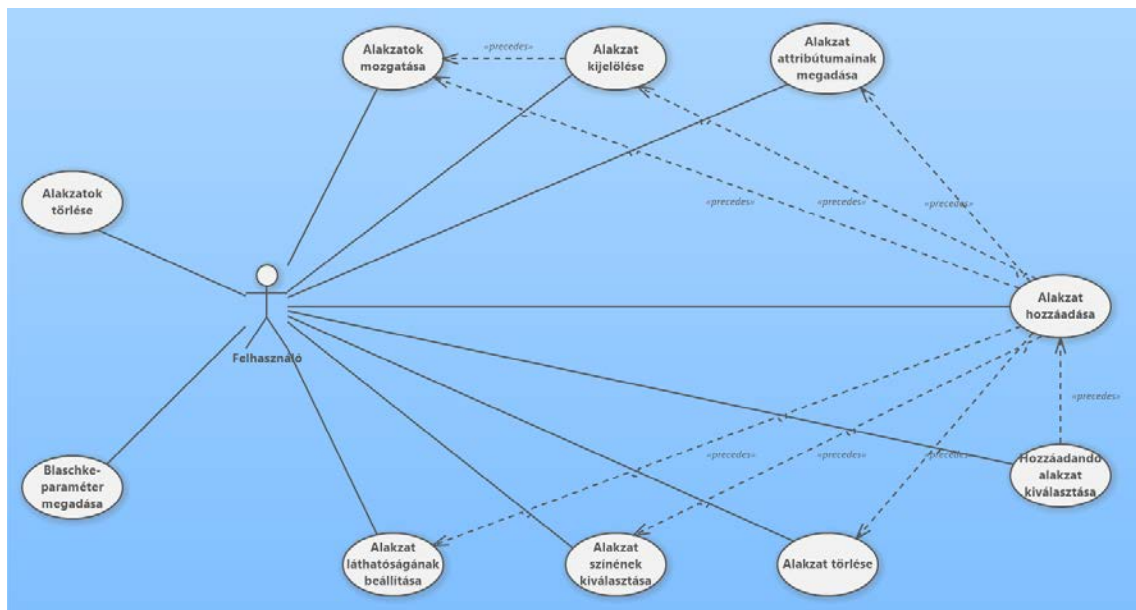
A kép exportálás funkció az aktuális fül fő kinézetéről készített képet ugyanúgy „*FileWindow*” osztály segítségével menti le. Ennek az „*exportImage(WritableImage)*” nevű statikus függvénye felel ezért.

A mentési és megnyitási funkciók esetében felmerülő kivételek esetén egy hibaablak jelenik meg a megfelelő hibaüzenetekkel.

Alakzatok vezérlői

Az alakzatokhoz kettő kontroller tartozik, az egyik a mellékvezérlő, ami az alakzatok kiválasztásáért, hozzáadásáért, törléséért, beállításáért illetve a *Blaschke*-paraméter beállításáért felel, a másol pedig ezeknek a megfelelő kirajzolását, illetve a *Blaschke*-függvény ábrázolását végzi.

A két kontroller funkcióit külön fogom tárgyalni, viszont egy diagramon ábrázolom, amit a 37. ábra mutat. Először a mellékvezérlő elemeit fogom kitárgyalni, később pedig a fő felületét.



37. ábra: Alakzatok fül felhasználói diagram

A mellékvezérlő adattagként eltárolja a fővezérlőt, amit ahogy korábban említettem, a „MainController” segítségével fogunk beállítani. Ezen felül további adattagjai az állapotsor hibaüzenete, Blaschke-paraméter, ennek a szöveges mezői, legördülő menü, fa nézet, ennek elemei, kijelölt alakzat és korábban említett „undo” és „redo” veremek, illetve az alakzatokat tartalmazó lista.

A vezérlő „initialize” metódusában inicializálódnak a nem @FXML annotációval ellátott attribútumok. A fa nézet kijelöléséhez itt lesz hozzá adva az esemény figyelő, ami az aktuális fa elemet - mint alakzatot - beállít a kijelöltre, továbbá meghívja a Blaschke-paraméterhez tartozó szövegmezők szűrőit és tevékenységeit beállító függvényt.

Ebben a metódusban először is be kell állítani, hogy csak számadatra működjön helyesen. Tehát ha a felhasználó helytelen bemenetet ad meg, akkor jelezze neki az állapotsoron a hibát, azaz be kell állítani az hibaüzenethez tartozó attribútumot a megfelelő szövegre. Ezen felül a mező stílusánál, a keretet pirosra állítja, evvel jelezve a hibás bemenetet. Azonban ha helyes a bemenet, akkor a Blaschke-paraméternek megadja a program a valós- és képzetes részét, és frissíti a fővezérlő „refreshShapes()” függvényével a Blaschke-függvény grafikonját és a kirajzolt alakzatok formáját.

Az FXML fájlban az összes alakzat törlése gomb lenyomásához tartozó funkció a visszavonás verméhez hozzáadja a törölt alakzatokat, és kiüríti az újra vermet, ezután az

alakzatok típusához tartozó fa elemek gyerekeit tartalmazó listát és az alakzat listát kiüríti, illetve a fő területen kirajzolt alakzatokat kitörli.

A hozzáad gomb eseménye kiüríti az újra vermet, lekéri a legördülő menüből az alakzat típusát, és ez alapján eldönti, hogy milyen alakzatot kell létrehozni. Minden újonnan létrehozott alakzatnak be kell állítani a törlés gombjához tartozó esemény funkcióját, az elrejtés kapcsoló eseményét, a szöveges beviteli mező szűrőit és tevékenységeit. Ezen függvényeket a következő bekezdésben fogom részletezni. Ezután a függvény hozzáadja a visszavonás verembe, az alakzat listába és a fanézethez az alakzatot.

Az alakzatok törlés gombjának eseménye kiüríti az újra vermet, hozzáadja az alakzatot a visszavonás verméhez, ezenfelül kitörli az alakzatok listájából, a fa nézetből és a fő felületből egyaránt. A láthatósági kapcsoló lenyomására át lesz állítva a kapcsoló ikonja, továbbá a geometriai alakzatra és a transzformáltjára meghívja a „*setVisible(boolean)*” függvényt, amin keresztül a láthatóságot váltogatja. A beviteli mezőknél be lesz állítva a szűrő, ami csak számokat fogad el, továbbá hozzá lesz adva egy eseménykezelő a billentyűlenyomásokra, ami az alakzatok kirajzolásának a függvényét hívja meg a fő felület vezérlőjében, ha helyes az, amit a felhasználó begépett.

Ezen felül van a fájlból hozzáadás, szerkesztés menüpontjaiból származó hozzáadás és törlés műveletek függvényei. Ezek csak kismértékben térnek el az eredeti metódusoktól.

A fővezérlő a mellék kontrollert tárolja el adattagként, ezen felül mezői még az FXML fájlból származó „*Group*” típusú csoport elemeket, amik az egységkörök kirajzolásához szükségesek, ugyanis majd ide lesznek hozzáadva a kirajzolandó alakzatok. Kezdetben ezekben rendre csak két tengely és a kör van benne. Továbbá itt vannak eltárolva a függvények és ezeknek a vonaldiagram felülete, egér koordinátái és a z- és w-síkbeli alakzatokhoz tartozó „*Map*” objektumok, ami a konkrét geometriai alakzatok könnyebb elérésének érdekében szükséges.

Az „*initialize*” függvényben szintén inicializálódnak a szükséges adattagok, ezen felül a függvények nevei is itt lesznek beállítva, és itt lesznek hozzáadva a vonaldiagram gyerekei közé.

A z-síkot reprezentáló „Group” objektumhoz számos egérhez kapcsolódó eseménykezelő lesz kötve. Az egér mozgása, a koordinátáit fogja lementeni az erre létrehozott objektumba. Az egérgomb lenyomására lekéri a mellék kontrollertől a kijelölt objektumot, és ha van ilyen, akkor az alakzatnak lementi kiinduló pontjának ezt a koordinátát, és meghívja az alakzat rajzoló függvényt. Az egérgomb lenyomva tartásával való mozgatsnál két funkció lehetséges: a „Shift” billentyű lenyomására az alakzat áthelyezése fog történni, ha nincs lenyomva, akkor pedig alakzatonként eltérő viselkedés váltódik ki. Pont esetén nem történik semmi, szakasz esetén a végpont beállítása, kör esetén a sugár változtatása, körív esetén pedig a szög megadása fog végbe menni. Az ilyen események során újból meg lesz hívva a kirajzolás függvénye.

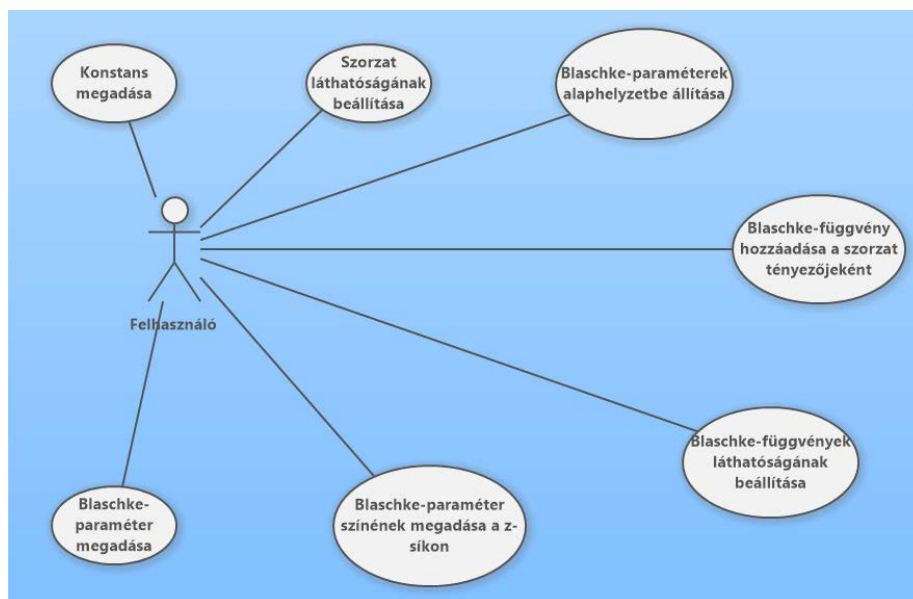
Az alakzat hozzáadása az alábbiak alapján történik. Itt, ha az alakzat és a transzformáltja már létezik a korábban említett „Map” objektumban, akkor kitörli a program, így kikerülve a többszöri kirajzolást. Ezután a „Blaschke” osztály „transform(TIShape, Complex)” függvényét meghívva beállítjuk a transzformált objektumot, végül hozzáadjuk az alakzatot és a transzformáltját a „Map”-hez, illetve a z- és w-síkhöz.

Egy alakzat törlésénél az alakzatot és a transzformáltját kitöröljük a megfelelő „Map”-ből és a síkból. Összes alakzat törlésénél a „Map” kiürítése történik meg, illetve az összes alakzat törlése.

Frissítéskor az oldal elkéri a Blaschke-paramétert a vezérlőtől, ezután kitörli az összes w-síkon lévő alakzatot, utána pedig az új paraméterrel eltranszformált alakzatot fogja hozzáadni a w-síkhöz és a „Map”-hez is egyaránt. Ezután a Blaschke-függvény értékeit kiszámolja a program $[0, 2\pi)$ intervallumon 0.1 egységenként, amik alapján fel fogja rajzolni a függvényt a program. Végül a „showChart(boolean)” függvény felel az alakzatok elrejtéséért.

Egyenletek vezérlői

Az egyenletek fülhöz szintén két vezérlő tartozik. A mellék felületen a Blaschke-függvények és szorzatok beállításához szükséges elemek vannak, míg a fő felületen ezeknek az eredményei láthatók. A kontrollerek funkcióit a [38. ábra](#) mutatja be röviden.



38. ábra: Egyenletek föl felhasználói esetek

Először is az oldal vezérlő eltárolja adattagként a fő vezérlőt, ezen felül mezője még a hibaüzeneteket tartalmazó objektum, a komplex konstans és a szövegmezői, a szorzat láthatósági kapcsolója, a fa nézet és annak gyökere, végül a kijelölt *Blaschke*-függvénye.

Az „*initialize*” metódusban adjuk hozzá a *Blaschke*-függvények fa elemeit az FXML fájl helyett, a redundancia elkerülése érdekében. Minden ilyen elemnek be kell állítani a láthatósági- és szorzathoz adás kapcsolóját, a szövegbeviteli mezőket, végül hozzá kell adni a fa nézet gyökeréhez. Ezen kívül a fa nézet kijelöléséhez hozzá kell adni egy esemény figyelőt, ami lementi a kijelölt alakzatot egy változóba, továbbá a konstanshoz tartozó beviteli mezők is itt kerülnek beállításra.

Az elemek láthatósági kapcsolójának esemény figyelője a következő: ha a kapcsoló be van nyomva, akkor beállítja a láthatóságot hamisra, egyébként pedig igazra, ezen kívül a megfelelő ikonokat váltja. A szorzathoz láthatósági kapcsolójánál hasonló az eljárás, annyi különbséggel, hogy ez az esemény figyelő explicit módon az FXML-en keresztül van megadva.

A szorzathoz adás eseményfigyelője megnézi, hogy a kapcsoló be van-e kattintva, ha igen, akkor a szorzat halmazába behelyezi a *Blaschke* elemet, különben kitörli. Ezen felül a megfelelő ikonokat beállítja. Ezt követve a fő területen meghívja a frissítés függvényt. Ez a függvény 0-val való osztáskor kiválthat egy kivételt, ezt az állapotsoron jelzi a felhasználó felé.

Konstans elem beviteli mezőire egy eseménykezelő lesz állítva, ami a billentyűlenyomásokat figyeli. Ha helyes bemenetet adott meg a felhasználó, akkor a komplex konstans megfelelő része be lesz állítva az értékre, ezután meghívja a fő felület frissítését. Ha hibás a bemenet, akkor az állapotsoron ez jelezve lesz.

Blashcke elemek beviteli mezőinek a beállítása hasonló, annyi különbséggel, hogy ez a fő felületen lévő körben felrajzolja a *Blashcke*-paramétert.

A „Clear” gomb esemény figyelője, ha gombnyomást érzékel, akkor visszaállítja az alapértelmezett értékekre a *Blaschke*-paramétereket, ezután meghívja a főfelület rajzoló függvényét.

A fővezérlő eltárolja a mellék kontrollert adattagként. Továbbá eltárolja a „Group” típusú objektumot, ami az egységkört reprezentálja, ezen felül lementi magát a kört. Adattagja még a vonaldiagramok, amik a *Blaschke*-függvényeket és a gyököket fogják megjeleníteni, melyek mind adattagjai az osztálynak. Kettő darab logikai változója van az osztálynak, az egyik a *Blaschke*-függvény helyességét tárolja el például, ha a nevező nullával azonos, akkor ez hamis. A másik pedig azt tárolja el, hogy a szorzat látszik-e. Végül adattag még a kijelölt *Blaschke* elem, az egér koordinátái az egységkörön, illetve egy „Map” objektum, ahol az elemek a kulcsok, az értékek pedig a pontok.

Az „initialize” metódusban lesznek létrehozva a szükséges objektumok, a függvények nevei itt lesznek beállítva, továbbá itt adódnak a vonaldiagramhoz hozzá, mint gyerek elemek.

Az @FXML annotációval ellátott metódusokat fogom ebben a bekezdésben kifejteni. Mindegyik ilyen metódus az egységkörhöz van kötve, és egér eseményekre reagál. Klikk eseményre az oldalsó vezérlőtől lekéri a kijelölt *Blaschke*-elemet, ha ez nem „null”, akkor az elem pontját, azaz a *Blaschke*-paramétert beállítja a kattintás koordinátáira, és megpróbálja kirajzolni ezt a pontot, ha nem lesz nulla a nevező. Egér mozgatására az egér koordinátáit lementi az erre szolgáló objektumba. Az egérgomb nyomva tartásával és egér mozgatásával a pontot lehet áthelyezni.

A láthatóság függvény a paraméterben kapott *Blaschke* elem alapján a megfelelő függvények láthatóságát állítja a második paraméterben kapott értékre. Ha az első

paraméter „null”, akkor a szorzat láthatóságát fogja beállítani. A láthatóság beállítása egy privát függvény segítségével történik.

A rajzolás metódusa a *Blaschke* elem alapján frissíti a függvényt, illetve az egyenletek megoldását is a megfelelő függvények hívásával végzi, továbbá a *Blaschke*-paraméter alapján frissíti az egységkörön a pontot.

Két féle frissítési függvénye van az osztálynak. A „*refreshChart*” nevű függvény a paraméterben kapott *Blaschke*-függvényt frissíti, a „*refresh*” pedig a gyökök frissítéséért felel. Ezen belül történik a gyökök kiszámítása is. Ez a függvény kéri le az oldal vezérlőtől a szorzat tényezőit tartalmazó halmazt, aminek a mérete alapján eldönti, hogy hányadfokú az egyenlet, aminek a megoldásait a „*MathUtils*” osztály megfelelő függvényeinek hívásával felrajzolja a vonaldiagramra, és kiírja az eredményét.

Tesztelés

Egység tesztelés

A *modell* csomagban lévő osztályokra készültek egységtesztek. Az eljárásokhoz és függvényekhez is több teszt eset készült. Az olyan függvények, melyek esetében kivételek is előfordulhatnak, ott ezekre is készültek teszt esetek. Készítettem egy „*TestAll*” nevű osztályt, ami az összes tesztet lefuttatja, de akár osztályonként is lehetséges ezeknek a futtatása.

Manuális tesztelés

Shapes fül

Blaschke-paraméter megadása:

- Helyes *Blaschke*-paraméter megadása
Eredmény: A *Blaschke*-függvény kirajzolódik
- Helytelen *Blaschke*-paraméter megadása, a nevező nullával egyenlő
Eredmény: Az állapotsoron a megfelelő hibaüzenet megjelenik, és piros lesz a beviteli mező kerete
- Helytelen *Blaschke*-paraméter megadása, érvénytelen karakter
Eredmény: Az állapotsoron a megfelelő hibaüzenet megjelenik, és piros lesz a beviteli mező kerete

Alakzat hozzáadása:

- Pont hozzáadása
Eredmény: A pont fa elem hozzáadódik
- Szakasz hozzáadása
Eredmény: A szakasz fa elem hozzáadódik
- Körív hozzáadása
Eredmény: A körív fa elem hozzáadódik
- Kör hozzáadása
Eredmény: A kör fa elem hozzáadódik

Minden alakzat törlése:

- Minden alakzat törlése
Eredmény: Minden alakzat törlődik

Alakzat törlése:

- Pont törlése
Eredmény: A pont fa elem törlődik
- Szakasz törlése
Eredmény: A szakasz fa elem törlődik
- Körív törlése
Eredmény: A körív fa elem törlődik
- Kör törlése
Eredmény: A kör fa elem törlődik

Alakzat láthatóságának állítása:

- Pont láthatóságának állítása
Eredmény: A pont láthatósága megváltozik
- Szakasz láthatóságának állítása
Eredmény: A szakasz láthatósága megváltozik
- Körív láthatóságának állítása
Eredmény: A körív láthatósága megváltozik
- Kör láthatóságának állítása
Eredmény: A kör láthatósága megváltozik

Alakzat színének állítása:

- Pont színének állítása
Eredmény: A pont színe megváltozik
- Szakasz színének állítása
Eredmény: A szakasz színe megváltozik
- Körív színének állítása
Eredmény: A körív színe megváltozik
- Kör színének állítása
Eredmény: A kör színe megváltozik

Alakzat nevének állítása:

- Pont nevének állítása
Eredmény: A pont neve megváltozik
- Szakasz nevének állítása
Eredmény: A szakasz neve megváltozik
- Körív nevének állítása
Eredmény: A körív neve megváltozik
- Kör nevének állítása
Eredmény: A kör neve megváltozik

Alakzat számot váró mezőinek állítása:

- Pont adatainak helyes megadása
Eredmény: A pont adatai megváltoznak és kirajzolódik
- Szakasz adatainak helyes megadása
Eredmény: A szakasz adatai megváltoznak és kirajzolódik
- Körív adatainak helyes megadása
Eredmény: A körív adatai megváltoznak és kirajzolódik
- Kör adatainak helyes megadása
Eredmény: A kör adatai megváltoznak és kirajzolódik
- Pont adatainak helytelen megadása
Eredmény: Az állapotsoron megjelenik a megfelelő hibaüzenet, piros lesz a beviteli mező kerete és az alakzat kirajzolása nem frissül

- Szakasz adatainak helytelen megadása
Eredmény: Az állapotsoron megjelenik a megfelelő hibaüzenet, piros lesz a beviteli mező kerete és az alakzat kirajzolása nem frissül
- Körív adatainak helytelen megadása
Eredmény: Az állapotsoron megjelenik a megfelelő hibaüzenet, piros lesz a beviteli mező kerete és az alakzat kirajzolása nem frissül
- Kör adatainak helytelen megadása
Eredmény: Az állapotsoron megjelenik a megfelelő hibaüzenet, piros lesz a beviteli mező kerete és az alakzat kirajzolása nem frissül

Alakzat adatainak változtatása z-síkon való egér eseményekkel:

- Pont adatainak megadása kattintással
Eredmény: Megfelelő adatok beírása a mezőkbe és helyes kirajzolás
- Szakasz kezdőpontjának megadása kattintással
Eredmény: Kezdőpont megfelelő beírása a mezőkbe és helyes kirajzolás
- Körív középpontjának megadása kattintással
Eredmény: Kezdőpont megfelelő beírása a mezőkbe és helyes kirajzolás
- Kör középpontjának megadása kattintással
Eredmény: Kezdőpont megfelelő beírása a mezőkbe és helyes kirajzolás
- Szakasz végpontjának megadása egérgomb nyomva tartásával és egér mozgatásával
Eredmény: Végpont megfelelő beírása a mezőkbe és helyes kirajzolás
- Körív szögének megadása egérgomb nyomva tartásával és egér mozgatásával
Eredmény: Szög megfelelő beírása a mezőkbe és helyes kirajzolás
- Kör sugarának megadása egérgomb nyomva tartásával és egér mozgatásával
Eredmény: Sugar megfelelő beírása a mezőkbe és helyes kirajzolás

Equations fül

Konstans megadása:

- Helyes konstans megadása
Eredmény: Az egyenlet helyesen lesz kiszámolva

- Helytelen konstans megadása, érvénytelen karakter

Eredmény: Az állapotsoron a megfelelő hibaüzenet megjelenik és piros lesz a beviteli mező kerete

Blaschke-szorzat láthatóságának váltása:

- Helyes konstans megadása

Eredmény: Az egyenlet helyesen lesz kiszámolva

- Helytelen konstans megadása, érvénytelen karakter

Eredmény: Az állapotsoron a megfelelő hibaüzenet megjelenik és piros lesz a beviteli mező kerete

Blaschke-paraméter megadása:

- Helyes *Blaschke*-paraméter megadása

Eredmény: A pont helyes kirajzolása, az egyenlet helyesen lesz kiszámolva

- Helytelen konstans megadása, érvénytelen karakter

Eredmény: Az állapotsoron a megfelelő hibaüzenet megjelenik és piros lesz a beviteli mező kerete

- Helytelen konstans megadása, a nevező nulla

Eredmény: Az állapotsoron a megfelelő hibaüzenet megjelenik és piros lesz a beviteli mező kerete

Blaschke-paraméter színének megadása:

- *Blaschke*-paraméter pontjának színének megadása

Eredmény: A *Blaschke*-paraméter pontjának színe megváltozik

Menüsáv

Fájl menü:

- Helyes JSON fájl betöltése

Eredmény: Alakzatok hozzáadása és helyes kirajzolása

- Helytelen JSON fájl betöltése

Eredmény: Hibaablak felugrása

- Alakzatok mentése JSON fájlba

Eredmény: Alakzatok sikeres kimentése JSON fájlba

- Aktív fő felület exportálása képbe
Eredmény: Kép exportálása sikeresen

Beállítások menü:

- Téma váltása
Eredmény: Sikeres témaváltás

Információ menü:

- Névjegy megnyitása
Eredmény: Névjegy felugró ablaka megjelenik
- Súlyó megnyitása
Eredmény: Súlyó PDF megnyitása

Szerkesztés menü:

- Visszavonás
Eredmény: Legutóbbi akció visszavonása
- Újra
Eredmény: Visszavont akció végrehajtása
- Kivágás, ha van kijelölt alakzat
Eredmény: A kijelölt alakzat JSON formátumának vágólapon kerülése és alakzat törlése
- Kivágás, ha nincs kijelölt alakzat
Eredmény: A vágólap nem változik
- Másolás, ha van kijelölt alakzat
Eredmény: A kijelölt alakzat JSON formátumának vágólapon kerülése
- Másolás, ha nincs kijelölt alakzat
Eredmény: A vágólap nem változik
- Beillesztés, ha helyes JSON formátum van vágólapon
Eredmény: Az alakzat hozzáadódik
- Beillesztés, ha helytelen JSON formátum van vágólapon
Eredmény: Hibaüzenetet tartalmazó felugró ablak megjelenik
- Beillesztés, ha a vágólap tartalma üres
Eredmény: Hibaüzenetet tartalmazó felugró ablak megjelenik

- Törlés, ha van kijelölt alakzat
Eredmény: Alakzat törlése
- Törlés, ha nincs kijelölt alakzat
Eredmény: -

ÖSSZEFOGLALÁS

Mindig is érdekelt a matematika, és érdekesnek tartottam a komplex függvénytant, különösen felkeltette az érdeklődésemet ez a témakör az orvostudományi vonatkozásai miatt, így esett ilyen típusú témakörre a választásom. Ezen felül törekedtem egy szép megjelenésű alkalmazás fejlesztésére, amit az ebben a témakörben jártas emberek használni tudnak könnyedén.

JavaFX keretrendszer egy új környezet volt számomra, és maga a matematikai háttér is, így ez egy kihívás volt, ami a szakmai fejlődésemet is elősegítette, hogy jobb szoftverfejlesztővé váljak.

A szoftver fejlesztése sosem ér véget, mindig van lehetőség egy alkalmazás bővítésére. Erre különösen odafigyeltem fejlesztés közben, ugyanis könnyen kiegészíthető a program, új nézetek és vezérlők hozzáadásával. Ezt elősegíti a fül nézet alkalmazása, ami segítségével különálló egységekbe lehet szervezni a programot, így egy olvasható környezetet teremtve a fejlesztő számára.

ÁBRAJEGYZÉK

1. ábra: Blaschke paraméter megadása	8
2. ábra: Alakzat kiválasztása	8
3. ábra: Alakzat hozzáadása	8
4. ábra: Színválasztó	9
5. ábra: Saját szín megadása	10
6. ábra: Példa az alakzatok attribútumaira	11
7. ábra: Alakzatok és képük $\alpha = -0,75$ paraméter esetén	11
8. ábra: Blaschke-függvény szemléltetése $\alpha = -0.75 + 0.4i$	12
9. ábra: Alakzat kijelölése.....	13
10. ábra: Konstans megadása	13
11. ábra: Blaschke-függvények az egyenleteknél	14
12. ábra: Blaschke-szorzat.....	14
13. ábra: Háromtényezős Blaschke-szorzat megoldása $c = 0.2 + 0.3i$ konstansra,.....	15
14. ábra: Előző egyenletnél a $\mathcal{B}_0(z)$ és $\mathcal{P}(z)$ képe.....	15
15. ábra: „File” menüpont	16
16. ábra: Példa a JSON fájl megnyitására Windows-on	17
17. ábra: Példa az alakzatok mentésére JSON fájlba Windows-on	17
18. ábra: Megnyitás és mentés folyamata.....	18
19. ábra: Exportált kép $\alpha = -0.75$ paraméter esetén.....	18
20. ábra: „Edit” menüpont.....	19
21. ábra: Settings menü	20
22. ábra: Felület témájának kiválasztása	20
23. ábra: Modell csomagjai	24
24. ábra: Komplex számok osztálydiagram	24
25. ábra: Blaschke osztálydiagram	25
26. ábra: Pont transzformációja.....	25
27. ábra: Szakas transzformációja kódrészlet	26
28. ábra: Kör transzformációja kódrészlet	26
29. ábra: Kör transzformációja részlet	27
30. ábra: Alakzatok leszármazása	28
31. ábra: Kör felírása három pontból részlet	29
32. ábra: Cramer-szabály implementálása 2x2-es mátrixra	30
33. ábra: Harmadfokú megoldás kódrészlet	30
34. ábra: BorderPane felépítése	32
35. ábra: Main.fxml hierarchiája	33
36. ábra: Menüsáv felhasználói diagram	35
37. ábra: Alakzatok fül felhasználói diagram	38
38. ábra: Egyenletek fül felhasználói esetek.....	41

HIVATKOZÁSOK

Minden hivatkozás elérhető volt 2020.05.26-án

- Java: <https://www.java.com/en/>
- JavaFX: <https://openjfx.io/>
- IntelliJ IDEA: <https://www.jetbrains.com/idea/>
- JSON-Simple: <https://code.google.com/archive/p/json-simple/>
- JUnit: <https://junit.org/junit4/>
- *Blaschke*-függvények:
 - Dr. Lócsi Levente, „Racionális függvényrendszerek alkalmazása a jelfeldolgozásban”
http://www.tnks.inf.elte.hu/vedes/Locsi_Levente_Tezisek_hu.pdf
 - Dr. Schipp Ferenc, „Racionális ortogonális rendszerek”
https://www.inf.elte.hu/dstore/document/326/schipp_ferenc_ROR_VV.pdf