Elektromos áramkör szimuláció

Házi feladat dokumentáció

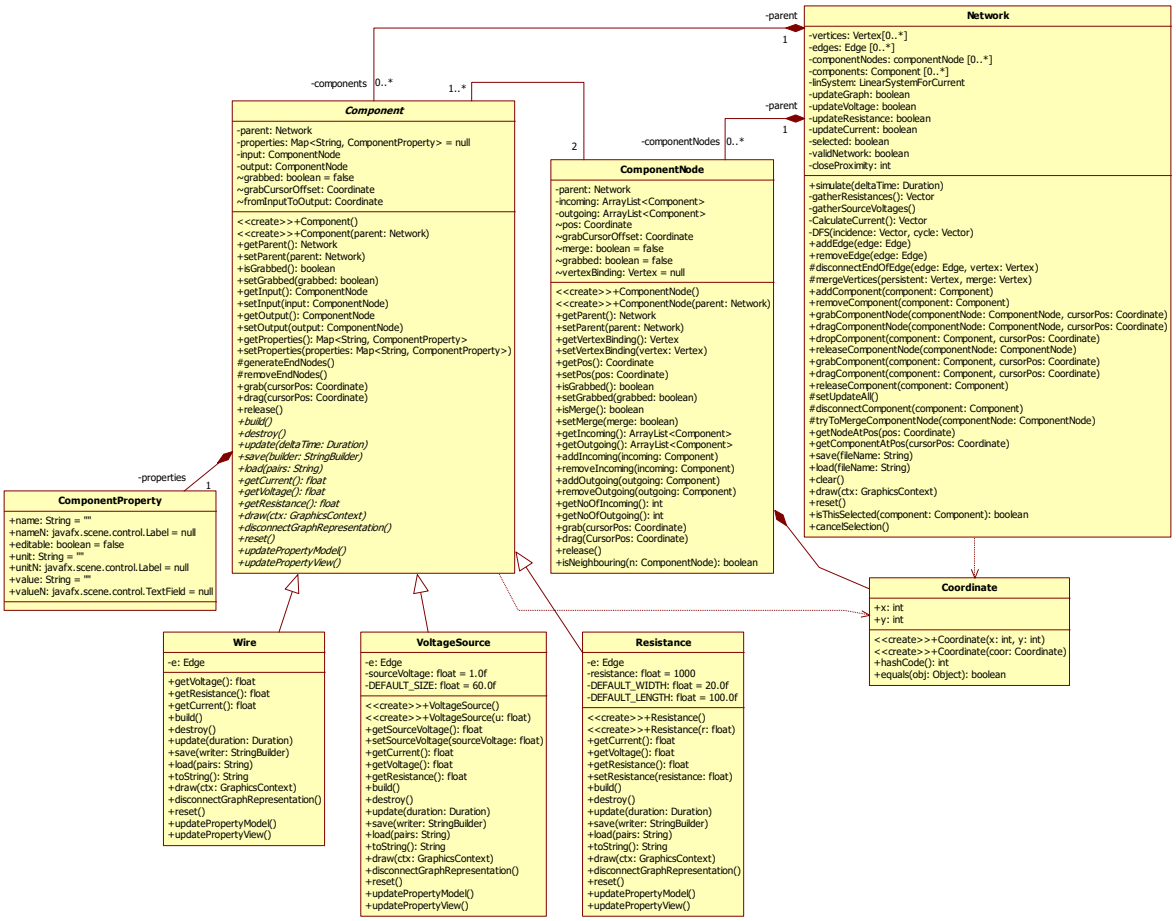
Készítő: Simon Zoltán
Neptun: HRSNUG

Adatszerkezetek

Bemutatásra kerül az Elektromos áramkör szimulációt megvalósító alkalmazásom magját képező osztály hierarchia. A program működésének megértését elősegítendő három absztrakciós szintbe soroltam az osztályokat. A három szint közt a mindegyikben szereplő Network osztály teremti meg a kapcsolatot.

Legfelső absztrakciós szint:

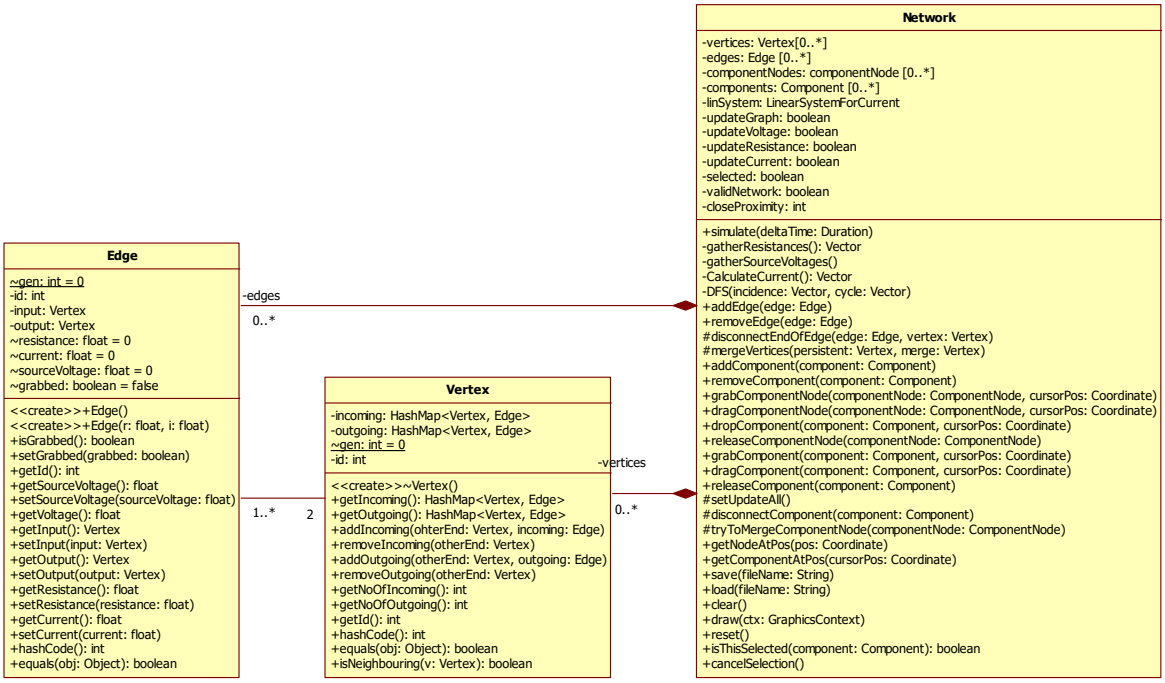
Ezen a szinten a hálózatot elektromos komponensekből összekapcsolódásának tekintem. Ezek a komponensek ismerik a fizikai tulajdonságaik. A felhasználó közvetlenül érintkezik velük. Mozgatja őket és írja-olvassa a paramétereik.



1. ábra Legfelső absztrakciós szint

Középső absztrakciós szint:

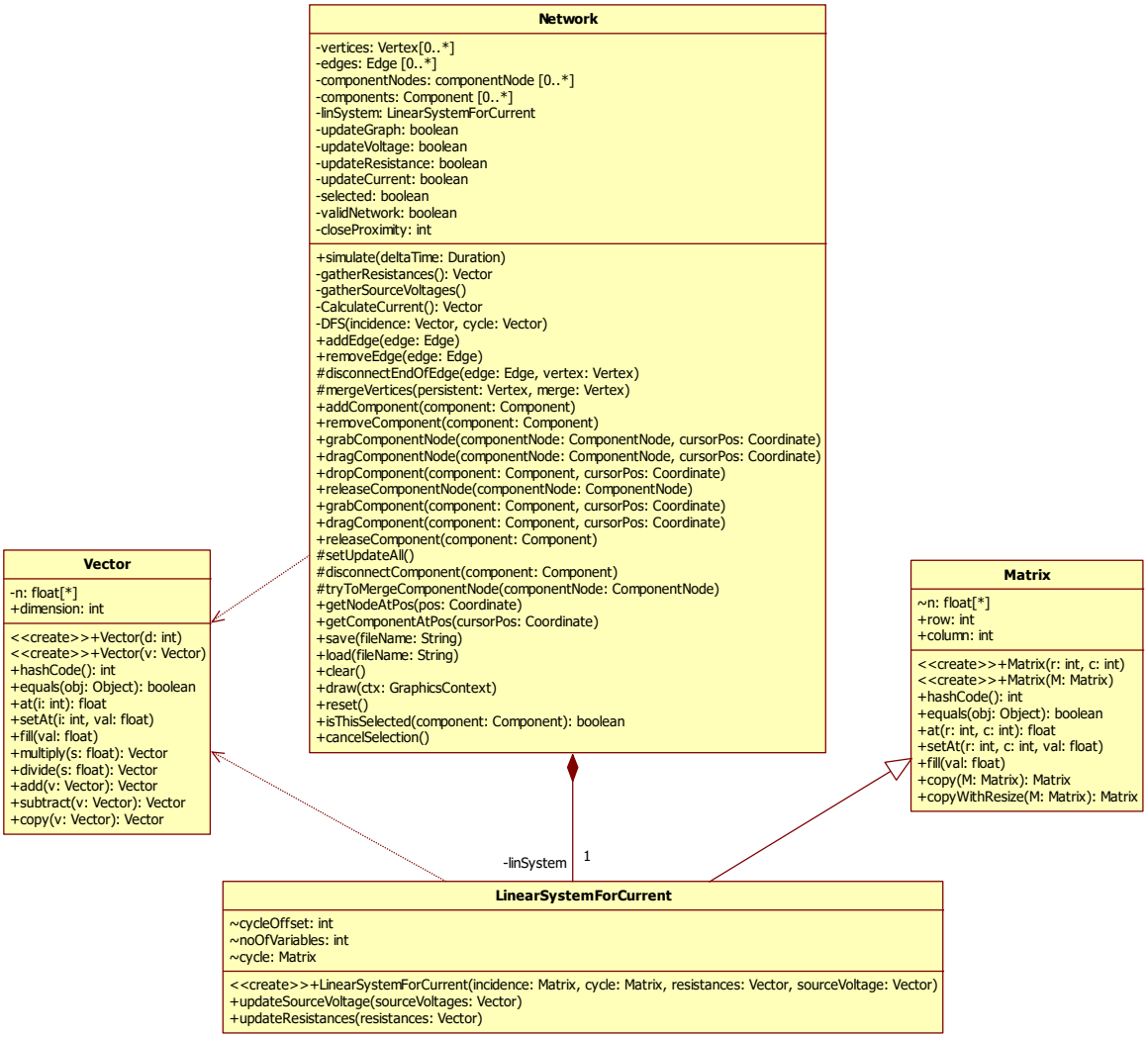
Ezen a szinten a hálózatot egy gráfként kezelem. Ezen végzek mélységi keresést. A gráf élei annyiban speciálisak, hogy ismerik a rajtuk átfolyó áramot, az ellenállásuk és tudják mekkora feszültségforrást képviselnek.



2. ábra Középső absztrakciós szint

Legalsó absztrakciós szint:

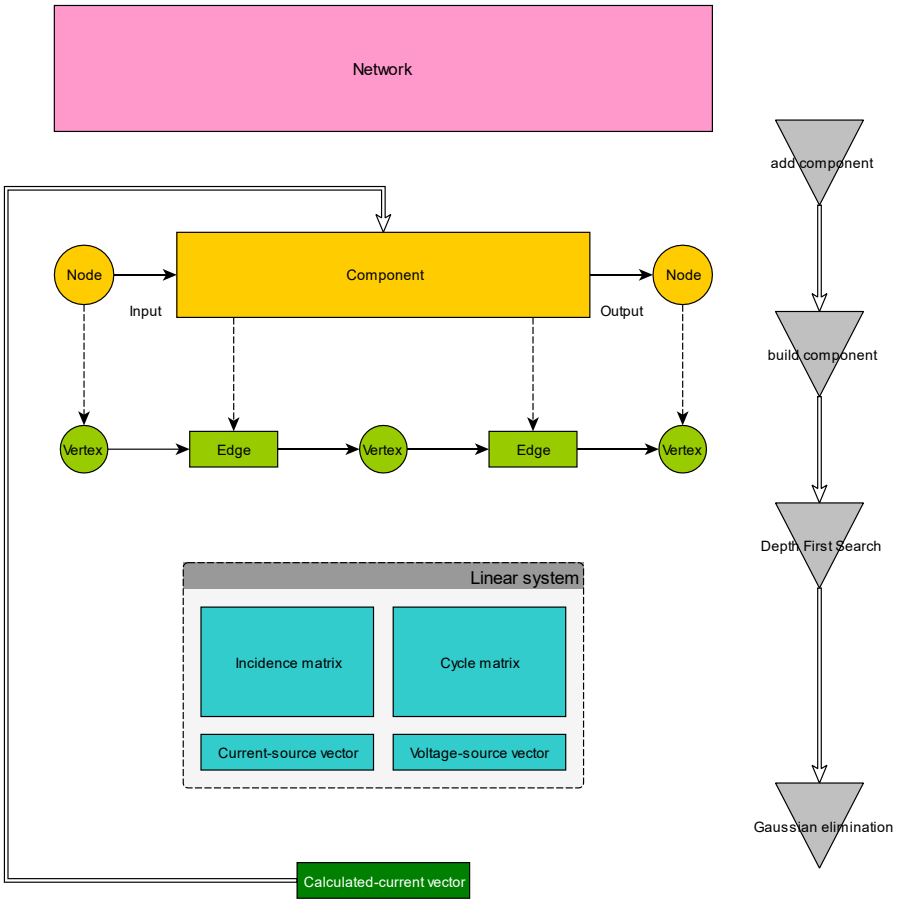
Ezen a szinten a hálózatot egy mátrixként kezelem. Erre egy speciális mátrixból leszármazott osztályt használok, ami a hálózat speciális tulajdonságait is ismeri.



3. ábra Legalsó absztrakciós szint

Összefoglaló az absztrakciós szintekről

- Elektromos komponensek
- Gráf
- Mátrix



4. ábra Absztrakciós szintek

Tervezési megfontolások

Az alkalmazást **Java** SE Development Kit 11 segítségével, **Eclipse** IDE for Java Developers környezetben készült. A GUI **JavaFX** könyvtár felhasználásával valósult meg. A mentés fájlok **txt** formátumúak **UTF-8** kódolással.

A program magját az áramkör szimulációjának matematikai megoldása adja. Ez gráfelméleti alapokon nyugszik. Az áramkört egy gráfként interpretálom, aminek az egyes élei az áramkör egyes komponenseinek felelnek meg. A gráf csúcsai az áramkör olyan pontjai, ahol két vagy több komponens kapcsolódik össze. A feladat a komponenseken átfolyó áram és eső feszültség meghatározása. A feszültségesést Ohm törvénye szerint a $V = I \cdot R$ összefüggéssel kapjuk, ahol V a komponensen eső feszültség az I az átfolyó áram és R a komponens elektromos ellenállása. Így a feladat csak az áram meghatározása. Ehhez a Kirchhoff-törvényeket lehet használni. Az első törvény, hogy a csomópontba befolyó áramok összege megegyezik az onnan elfolyó áramok összegével. A második törvény, hogy bármely zárt hurokban a feszültségek előjeles összege nulla. A gráfot leírhatjuk illeszkedési mátrixszal. Ekkor a mátrix egyes oszlopai az gráf éleinek, a sorok a gráf csúcsainak feleltethetők meg. Ez tökéletes az első Kirchhoff-törvény matematikai megfogalmazására. Írjon le a mátrix egy lineáris egyenletrendszert, ahol a baloldalon az ismeretlenek a komponenseken átfolyó áramok, az együtthatóik egy vagy mínusz egy attól függően, hogy milyen irányban folyik az áram. Az egyenletek jobb oldalán csupa nulla áll. A második Kirchhoff-törvény leírásához egy másik gráfreprezentációt kell használni. Ez a körmátrix. Ebben az oszlopok szintén éleknek, a sorok köröknek felelnek meg. Írjon ez le egy olyan egyenletrendszert, ahol az ismeretlenek a komponenseken folyó áramok az együtthatók egy, ha a komponens irányítása megegyezik a körével, mínusz egy, ha ellentétes. Az egyenletek jobb oldalán álljon nulla vagy, ha a körben van feszültségforrás, akkor ennek megfelelő feszültségérték. Mivel a feszültség Ohm-törvénye szerint kifejezhető az árammal, ezért ebben az egyenletrendszerben az ismeretleneket áramra, az együtthatókat pedig ellenállásra (vagy mínusz ellenállásra) cserélhetjük. Most már az első és második Kirchhoff-törvények alapján kapott egyenleteket egy egyenletrendszernek tekinthetjük és Gauss-eliminációval oldhatjuk.

A mátrixok létrehozásakor alkalmazhatunk néhány gráfelméleti tételt, amelyek miatt elég a gráfunk egy feszítőfáját leírni az illeszkedési mátrixszal, és elég egy ún. alapkörrendszert leírni a körmátrixszal. Ezek generálására a mélységi keresés (DFS) algoritmusát használom.

(A matematikai adatstruktúrákat nagyrészét már C++-ban megvalósítottam. Ezeket fogom Javára átírni.)

A matematikai modell fölött egy magasabb szintű modell szolgálja az elérhetőséget a felhasználó számára. Ebben a rétegben a komponensek és ezek végpontjai külön objektumokként szerepelnek. A DFS algoritmus ebből fogja létrehozni a két mátrixot. A mátrixokat ki kell egészíteni az egyenletek jobb oldalával és a körmátrix esetében az ellenállásokkal, amit az egyes komponensektől kérdezhetünk meg. (Ez lehetőséget ad, hogy később időben változó ellenállású komponenseket is implementáljak, mivel az a modell működését nem befolyásolja, ha a komponens belső ellenállása idővel megváltozik.) Végül a Gauss-elimináció futtatása után visszaadjuk a komponenseknek a rajtuk átfolyó áramot.

Amikor a felhasználó mozgat egy komponenst, akkor annak pozíciója megváltozik. A pozíciókat is csak a magasabb szintű modellben tárolom. Elsősorban a végpontok számára fontos, hogy tudják a pozíciójuk, mivel az összekapcsolásuk a pozíció alapján történik. Ha egy végpont mozgatását befejezzük, akkor az megnézi, nem került-e egy másik végpont „közvetlen közelébe". Ha igen akkor a két végpont összeolvad: Az egyik átadja a hozzá kötött komponensekről tárolt információt a másiknak, majd megsemmisíti önmagát. Hasonlóan történik, amikor egy komponenst valahogyan "kiszakítunk" a kapcsolásból. Ilyenkor a végpontjai lemásolódnak elfelejtve a többi hozzájuk kapcsolt komponenst. Erre azért van szükség, hogy egy komponens akkor se maradjon végpontok nélkül, ha nem kapcsolódik másik komponensekhez.

A kirajzolást elkülönítettem az adatmodellektől (**Modell-nézet-vezérlő** szerű megoldás).

Osztályok leírása

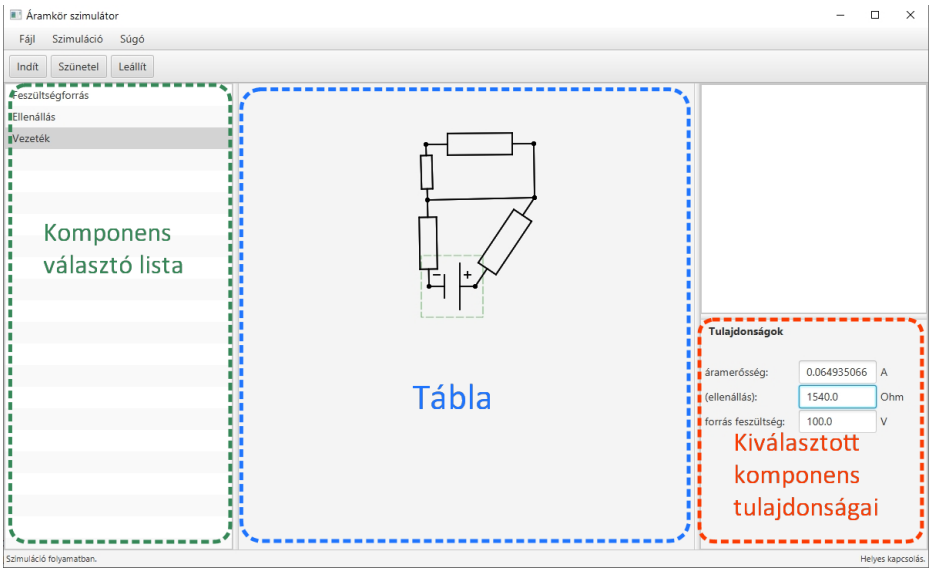
JavaDoc mellékelve.

Felhasználói leírás

Az applikáció egyszerű áramkörök modelljének létrehozását és szimulálását teszi lehetővé. A grafikus felület baloldalán elhelyezkedő komponens választó listából „drag and drop” segítségével hozható létre új komponensek a táblán. A táblán lévő elemek bal egérgombbal foghatóak meg. A végpontjaiknál fogva összekapcsolhatóak. A komponens testét megfogva mozgatható az. Ilyenkor lekapcsolódik a többi komponensről. Elengedve újra összekapcsolódik a végpontjai közelében lévő csomópontokkal. Az utoljára megfogott elem kiválasztva marad. A kiválasztott elem tulajdonságai leolvashatók és szerkeszthetők a jobb oldali tulajdonságok mezőben.

Az áramkör szimulációja az „Indít” gombbal indítható. A „Szünetel” gombbal várakoztatható a futása. A végleges leállításhoz a „Leállít” gomb vehető igénybe. Ezek a funkciók a szimuláció almenüből is elérhetőek.

Az elkészített modell fájlba menthető a „Fájl” almenü „Ment…” menüpontjával. A korábban elmentett modellek visszatölthetőek a „Fájl” almenü „Megnyit…” menüpontjával. Ugyanezen almenü „Új” menüpontjával üres tábla hozható létre. Az ablakot bezárva vagy a „Fájl” almenü „Bezár” pontjával kiléphetünk az alkalmazásból.



5. ábra GUI magyarázat

Use-case-ek

- **#UC01** Komponens felrakása
- **#UC02** Komponens mozgatása
- **#UC03** Komponensek összekötése
- **#UC04** Komponens kiválasztása
- **#UC05** Komponens paramétereinek leolvasása
- **#UC06** Komponens paramétereinek állítása
- **#UC07** Szimuláció indítása
- **#UC08** Szimuláció
- **#UC09** Szimuláció szüneteltetése
- **#UC10** Szimuláció folytatása
- **#UC11** Szimuláció leállítása
- **#UC12** Modell fájlba mentése
- **#UC13** Modell fájlból olvasása

Use case-ek kifejtése

#UC01 Komponens felrakása

Egy felhasználó egy listából válogathat a lehetséges *komponens* típusok közül. A választott elemet a *táblára* rakhatja.

#UC02 Komponens mozgatása

A *táblára* rakott komponensek egérrel meg lehet fogni és mozgatni lehet.

#UC03 Komponensek összekötése

A *táblára* rakott *komponensek* összeköthetők egymással, ha az egyik *komponens végpontjait* másik *komponensek végpontjaira* húzzuk.

#UC04 Komponens kiválasztása

A *táblára* rakott *komponensek* kiválaszthatók kattintással. Ilyenkor *kiválasztott* állapotba kerülnek. Ennek a komponens paramétereinek olvasásában és szerkesztésében van szerepe.

#UC05 Komponens paramétereinek leolvasása

A *komponensek* különböző *paramétereit* leolvashatjuk. *Paraméterek* lehetnek az átfolyó áram, feszültségesés, ellenállás stb.

#UC06 Komponens paramétereinek állítása

A *kiválasztott komponens paraméterei* szerkeszthetőek az erre szolgáló mezőben.

#UC07 Szimuláció indítása

Az áramköri *szimuláció* elindítható a dedikált gombra kattintva, vagy menüből.

#UC08 Szimuláció

Az áramköri *szimuláció* valós időben fut. Közben a *komponensek paraméterei* megfigyelhetőek, leolvashatók.

#UC09 Szimuláció szüneteltetése

A *szimuláció szüneteltethető* a dedikált gombra való kattintással. Ez a *szimulációnak* a *szüneteltetés* pillanatában érvényes állapotát merevíti ki.

#UC10 Szimuláció folytatása
Ha a *szimuláció szüneteltetve* van, akkor *folytatható* a dedikált gombra való kattintással. Ez a szimuláció *szüneteltetés* pillanatában érvényes állapotából folytatja a *szimulációt*.

#UC11 Szimuláció leállítása
A szimuláció véglegesen *leállítható* a dedikált gombra való kattintással. Ez után újonnani *indítás* (#UC07) egy semleges, kezdeti állapotból *indítja* a *szimulációt*.

#UC12 Modell fájlba mentése
A készített áramkör *modell elmenthető* a menüsor megfelelő elemére kattintva.

#UC13 Modell fájlból olvasása
A korábban készített áramkör *modell beolvasható* a menüsor megfelelő elemére kattintva.

Bemenet/Kimenet formátuma
A program az épített áramkört TXT dokumentumba menti. Ez soronként tartalmaz egy komponenseket. A sor első eleme a komponens típusát azonosítja Java canonical class name formátumban. Ez és a többi elem pontosvesszővel van elválasztva. Whitespace karakterek meg vannak engedve. (Feldolgozás során nem lesznek figyelembe véve.) Az adott elemek megnevezést és értéket tartalmaznak. Ezek kettősponttal vannak elválasztva. Ha az érték egy koordináta, akkor az szögletes zárójelben, vesszővel elválasztva van ábrázolva.

Példa:

```
class: main.java.network.Resistance; resistance: 10.0; inputPos: [70, 50]; outputPos: [40, 50]
class: main.java.network.Wire; inputPos: [70, 80]; outputPos: [70, 50]
```