

Házi feladat specifikáció

Készítő: Simon Zoltán

Neptun: HRSNUG

Feladat ismertetés: Elektromos áramkör szimuláció

Az applikáció egyszerű elektromos kapcsolási rajzok vizualizációját és szimulációját teszi lehetővé. Grafikus felületen lehet különböző áramköri elemeket elhelyezni és összekapcsolni. Az elemeket egy menüből választhatjuk ki és helyezhetjük el egy 2D felületen. Valós időben szimulálható az áramkör működése. Ekkor az egyes elemek paramétereit a grafikus felületen lehet leolvasni. Ilyen paraméterek az átfolyó áram, feszültség esés ellenállás stb. A szimulációt bármikor meg lehet állítani, vagy szüneteltetni. A komponensek a felületen a felhelyezésük után is tovább mozgathatók az egérrel. Megváltoztathatjuk, hogy melyik más komponensekhez kapcsolódnak. Ez értelemszerűen a rendszer szimuláció közbeni viselkedését változtatja. Az elkészített áramkört fájlba lehet menteni, hogy máskor visszatölthessük. Az aktuális modellünk komponenseit a grafikus felületen kívül egy listából is elérhetjük. A listában vagy a grafikus felületen egy komponensre kattintva annak beállításait érhetjük el. Itt olyan paramétereket állíthatunk, mint az ellenállás vagy feszültségforrás nagysága.

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 a *tábláról*, vagy a kiválasztott *komponens* esettében egy külön mezőből. *Paraméterek* lehetnek az átfolyó áram, feszültségesés, ellenállás stb.

#UC06 Komponens paramétereinek állítása

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

#UC07 Szimuláció indítása

Az áramköri *szimuláció* elindítható a dedikált a dedikált gombra való kattintással vagy billentyűzetről.

#UC08 Szimuláció

Az áramköri *szimuláció* valós időben fut. Közben a *komponensek paraméterei* megfigyelhetők, leolvashatóak.

#UC09 Szimuláció szüneteltetése

A *szimuláció szüneteltethető* a dedikált gombra való kattintással vagy billentyűzetről. 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 vagy billentyűzetről. Ez a *szimuláció szüneteltetés* pillanatában érvényes állapotából folytatja a *szimulációt*.

#UC11 Szimuláció leállítása

A *szimuláció véglegesen leállítható* a dedikált gombra való kattintással vagy billentyűzetről. Ez után újonnan *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.

Megoldási ötlet ismertetése

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

A program magját az áramkör szimulációjának matematikai megoldása adja. Ez gráfelméleti alapokon nyugszik. Az áramkört egy gráfként interpretálok, 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 egyenletrendszerét, ahol a baloldalon az ismeretlenek a komponenseken átfolyó áramok, az együtthatók 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 gráfrepresentációt kell használni. Ez a körmátrix. Ebben az oszlopok szintén éleknek, a sorok köröknek felelnek meg. Írjon ez le egy olyan egyenletrendszerét, 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 nagyrészt már C++-ban megvalósítottam. Ezeket fogom Javára átírni.)

A matematikai modell fölött egy magasabb szintű modell fogja szolgálni az elérhetőséget a felhasználó számára. Ebben a rétegben a komponensek és ezek végpontjai külön objektumokként fognak szerepelni. 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 fogom tárolni. Elsősorban a végpontoknak fontos, hogy tudják a pozíciójuk, mivel az összekapcsolásuk a pozíció alapján is történhet. Ha egy végpont mozgatását befejezzük, akkor az megnézi, nem került-e egy másik végpont „közvetlen közelségébe”. 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 különböző automatikus reakciók megvalósításához **observer pattern** használnék. A kirajzolást igyekszek függetleníteni az adatmolektól (**Modell-nézet-vezérlő** szerű megoldás).