FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Generátor elektrických obvodů pro předmět ČAO
Student:	Šimon Branda
Vedoucí:	Ing. Pavel Kubalík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Prozkoumejte existující řešení pro kreslení elektrických obvodů.
Vytvořte aplikaci, která bude umožňovat na základě zadaných parametrů automaticky náhodně generovat jednoduché obvody složené ze zdroje napájení, kondenzátorů, rezistorů a cívek.
Aplikace bude umožňovat nastavit hodnoty a typ součástky v obvodu.
Výsledné schéma bude možné překreslit tak, aby se změnila pouze pozice součástky.
Vygenerované schéma bude možné uložit do formátu XML a opětovně načíst.
Aplikace bude umožňovat popsat obvod pomocí rovnic, a to jak v časové oblasti, tak s pomocí fázorů.
Vygenerované rovnice bude možné bez úprav vložit do programu Wolfram Mathematica a zpracovat.
Při generování rovnic bude možné zvolit směr proudu a napětí ručně, popřípadě automaticky.
Výsledné řešení otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Hana Kubátová, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 7. února 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Generátor elektrických obvodů pro předmět ČAO

Šimon Branda

Katedra číslicového návrhu
Vedoucí práce: Ing. Pavel Kubalík, Ph.D.

13. května 2018

Poděkování

Chtěl bych poděkovat panu doktorovi Kubalíkovi za vedení této bakalářské práce a za cenné připomínky při její tvorbě. Dále bych chtěl poděkovat mé rodině za vytrvalou podporu po celou dobu mého studia a také za pomoc při závěrečné korektuře této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2018 .....

Odkaz na tuto práci

Branda, Šimon. *Generátor elektrických obvodů pro předmět ČAO*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato práce se zabývá návrhem a implementací aplikace na generování elektrických obvodů pro předmět BI-ČAO. Výsledná aplikace umožňuje náhodné vygenerování obvodu se zvoleným počtem hran a součástek, výměnu jednotlivých součástek beze změny tvaru elektrického obvodu a například také export do formátu XML. Daná aplikace byla napsána v jazyce C++ s pomocí grafické knihovny Qt. Generátor by měl sloužit k zjednodušení a k zautomatizování testování studentů v daném předmětu.

Klíčová slova náhodný generátor, analogové obvody, metoda uzlových napětí, Kirchhoffovy zákony, BI-ČAO, FIT ČVUT

Abstract

This bachelor thesis deals with design and implementation of application for generating schemes of analog curcuits for class CAO. The resulting application allows generating random analog circuits with various numbers of edges and components, changing any component without changing structure of whole circuit and exporting it into XML format. It was written in C++ language with Qt library. Generator will make testing of students in given subject easier.

Keywords automatic generator, analog circuit, node voltage method, Kirchhoff's law, BIE-CAO, FIT CTU

Obsah

Seznam obrázků

Úvod

Každý člověk, který někdy studoval vysokou školu, ví, že k úspěšnému dokončení studia je potřeba splnit určité povinné předměty. Složení povinných předmětů závisí na druhu a oboru jednotlivých škol. Na Fakultě Informačních Technologií Českého vysokého učení technického v Praze (zkráceně FIT ČVUT) jsou povinné předměty kombinací takových předmětů, které dají základ a rozhled každému člověku, který by se chtěl angažovat v počítačovém průmyslu. Obsahují například základy matematiky, teoretické informatiky, bezpečnosti, databázových systémů a také znalosti hardwaru.

Do posledního okruhu, mimo jiné, patří i předmět Číslicové a analogové obvody (BI-ČAO), který je vyučovaný v prvním semestru bakalářského programu. Ukazuje studentům základy analýzy číslicových a analogových obvodů, jejich návrh a výpočet jejich parametrů. Problémem tohoto předmětu, a všech dalších v prvním semestru, je to, že se na fakultu hlásí spousta studentů a je tedy velice obtížné je všechny kvalitně otestovat. Přednášející a cvičící proto tráví spoustu času vymýšlením, tvořením a poté hlavně opravováním zápočtových a zkouškových testů. Tento problém se dá vyřešit automatickým generováním jejich zadání a poté počítačovou kontrolou, zda jsou odpovědi studentů shodné s výsledky, které vytvořil generátor daného testu. Tento systém testování už existuje v jiných předmětech, např. v Programování a algoritmizace 1 (BI-PA1) a dalších programovacích předmětech. Bohužel pro předmět BI-ČAO ještě nebylo nic vytvořeno.

Díky tomuto problému vyvstalo téma této bakalářské práce. Jejím cílem je analyzovat aktuální možnosti kreslení elektrických obvodů a vytvořit program, který by dokázal sám, nebo jen s minimálním zásahem zkoušejících, vygenerovat analogový obvod a příslušné obvodové rovnice, jak v časové oblasti, tak s pomocí fázorů. Generovaný obvod se může skládat ze zdrojů napětí, rezistorů, kondenzátorů, cívek a zdrojů proudu. Je možné měnit libovolné součástky bez změny rozložení obvodu, měnit jen jejich hodnotu nebo měnit směr napětí a proudu. Pro potřeby vyučovaného předmětu je také nutné, aby bylo možné vygenerovaný obvod uložit a později načíst.

Práce je rozdělena na čtyři kapitoly. První část se zabývá prvotním seznámením s řešením tohoto problému, zkoumáním a porovnáváním stávajících řešení grafické tvorby elektrických obvodů. Druhá kapitola se věnuje návrhům a řešením jednotlivých problémů, které by měla tato práce řešit. Ve třetí kapitole se rozebírá závěrečná realizace aplikace a v poslední části je popsáno testování již vytvořené aplikace, která je připravena k nasazení.

Kapitola 1

Existující řešení

V oblasti kreslení a návrhu elektrických obvodů existuje mnoho kvalitních a naprosto dostačujících programů. Pro používání některých z nich je nutné si pořídit jednorázovou licenci, za další z nich je potřeba platit každý měsíc předplatné a ostatní jsou i zadarmo. Ty jsou nejčastěji vyvíjené jako open-source software (software, u kterého je veřejně dostupný zdrojový kód). I přesto, že je nabízených programů na trhu spousta, žádný nevyhovuje požadavkům předmětu BI-ČAO.

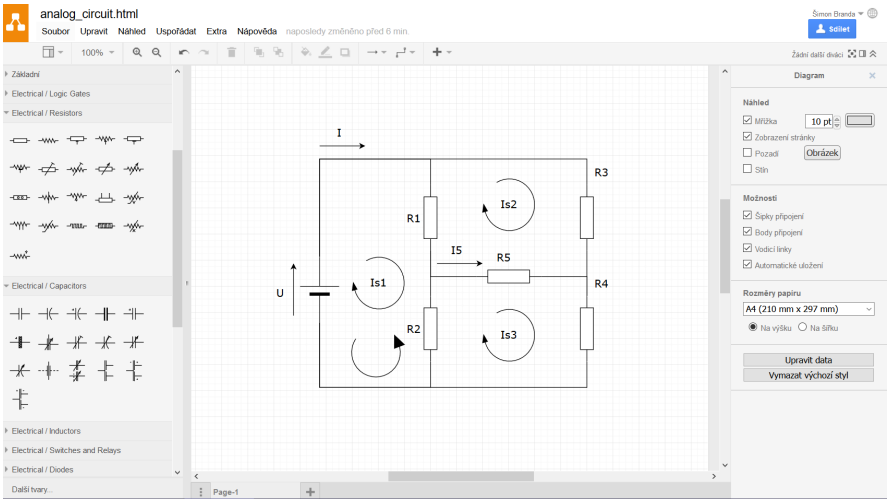
Na začátku této práce bylo sepsáno několik podmínek, které výsledný program musí splňovat. Mezi ně patří:

1. Umožnění změny součástky bez překreslení celého obvodu.
2. Vygenerování obvodových rovnic pro všechny uzly a pro každou součástku.
3. Automatické generování obvodu na základě zadaných parametrů.

Většina dostupných programů splňovala vždy jeden z výše tří zmíněných bodů, bohužel se ale nepodařilo najít software, který by splnil všechny podmínky. Pokusím se zde popsat výhody a nevýhody tří programů, které si vedly nejlépe, ale přesto se nevybraly.

Draw.io

Webová stránka `draw.io` je především určená pro tvorbu rozmanitých diagramů. Obsahuje velice intuitivní uživatelské prostředí, které napomáhá rychlé a snadné tvorbě. Podle uživatelského rozhraní (obrázek 1.1) můžeme poznat, že je celý program napojený na cloudový úložný systém od společnosti Google, což výrazně zjednodušuje práci se soubory pro širší veřejnost, která tento systém zná a používá. Díky širokému pojetí této aplikace se mezi vytvořené ikony dostala i schémata elektrických obvodů, díky nimž je umožněné navrhovat a vykreslovat elektrické obvody obsahující i diody, transistory nebo logická hradla.



Obrázek 1.1: Uživatelské prostředí aplikace `draw.io`.[1]

Velká nabídka funkcí tohoto programu je sice pro většinu uživatelů výhodná, ale pro účely výuky BI-ČAO je to spíše nevýhodné. V grafickém prostředí je možné měnit pozici jedné součástky nezávisle na zbytku obvodu, ale bohužel automatické generování celého obvodu neobsahuje. To je však stěžejní pro účely předmětu. Kvůli specializaci na kreslení více druhů diagramů a grafů je zřejmé, že také vytvoření obvodových rovnic v tomto programu není možné.

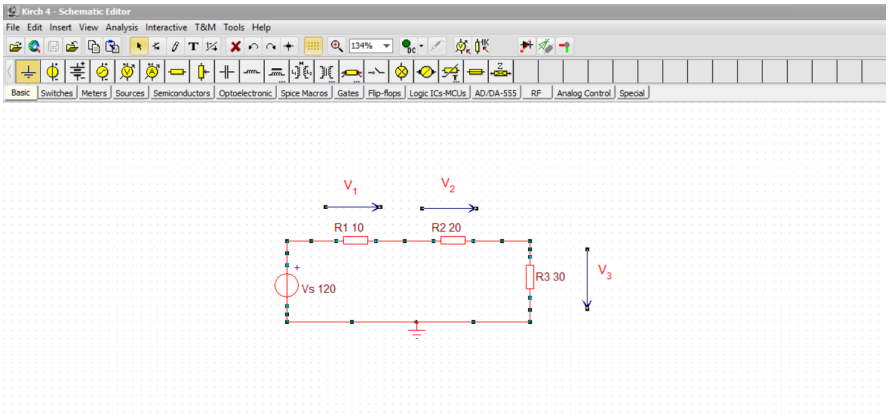
TINA Design Suite

Tento licencovaný software je od základu rozdílný od předchozí aplikace. Pochází od firmy DesignSoft a je určený pro analýzu, návrh a testování elektrických obvodů, ať už analogových nebo digitálních. Je velice obsáhlý, nabízí spoustu funkcí, jako například návrh rozložení obvodu na plošném spoji nebo otestování aplikací před nahráním do mikrokontroléru.

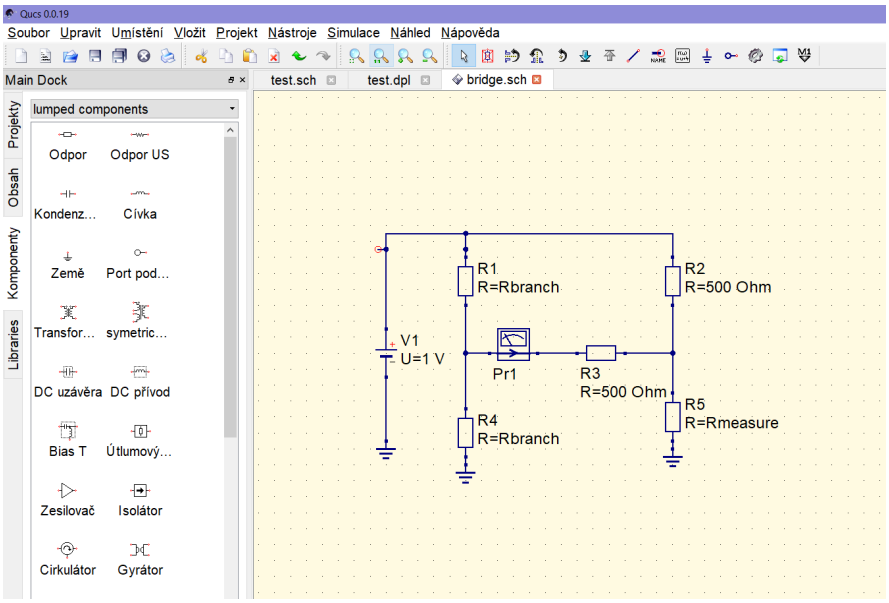
Uživatelské prostředí nepatří k těm přehlednějším (viz obrázek 1.2) a kvůli obsáhlosti tohoto programu je ovládání pro nováčka obtížné a trvá nějakou dobu, než se s ním sžije. Obsahuje spoustu funkcí pro práci s elektrickými obvody, dají se měnit jednotlivé součástky, jejich hodnoty a také se můžou vypočítat jednotlivá napětí a proudy na všech součástkách. Přestože tento program dokáže vygenerovat a spočítat obvodové rovnice pro zadané obvody, chybí i této aplikaci možnost automatického vygenerování náhodného obvodu.

Qucs

Poslední software, který je zde zmíněný, je program Quite Universal Circuit Simulator (zkráceně Qucs). Je to ještě nedokončený open-source program, který slouží k simulacím okruhů při určitém signálu. Umožňuje různé analýzy jak stejnosměrným, tak střídavým proudem a také například analýzu šumu.



Obrázek 1.2: Uživatelské prostředí programu TINA Design Suite.[2]



Obrázek 1.3: Uživatelské prostředí programu Qucs.[3]

Z uživatelského hlediska není tak přímočarý a jednoduchý jako prvně zmíněný `draw.io`, ale není až tak složitý jako software TINA (obrázek 1.3).

I přesto, že je zaměřený přímo na simulaci elektrických obvodů, pro naše účely mu chybí dvě zásadní věci. Tak jako žádný dostupný software, ani tento neumí při zadaných parametrech samostatně vygenerovat náhodný obvod. Program zvládne správně nakreslenému obvodu vypočítat napětí a proudy na všech součástkách, ale bohužel už nezvládne vytvořit obvodové rovnice pro dané obvody, tudíž je pro nás nepoužitelný.

Kapitola 2

Analýza a návrh

V této kapitole jsou analyzovány implementační možnosti nejdůležitějších cílů, které má výsledný program splňovat. Jsou rozebrána různá řešení zadaných problémů, a poté je popsána vybraná varianta, která byla implementovaná.

Po domluvě s vedoucím práce byl jako vývojové prostředí vybrán software Visual Studio od společnosti Microsoft.[4] Tento software je velmi rozšířený a před zadáním této práce v něm už pracoval jak student, tak i vedoucí práce, což jen utvrdilo výběr tohoto vývojového prostředí.

Pro účely této práce byl jako programovací jazyk vybrán jazyk C++ s přidanou grafickou knihovnou Qt.[5] C++ byl vybrán díky jeho univerzálnosti a také díky možnosti snadného pokračování v rozšiřování práce dalším studentem. Na FIT ČVUT se tento jazyk vyučuje povinně. Grafická knihovna Qt byla přidána pro vykreslení grafů a pro snazší tvorbu uživatelského rozhraní.

2.1 Generování tvaru obvodu

Způsob generování grafů a jejich vykreslování nepatří k jednoduchým úlohám. Pro tuto práci byly z počátku zamýšlené tři způsoby vykreslení jednotlivých obvodů. Mezi prvními myšlenkami bylo řečeno, že by se mohl vždy vygenerovat náhodný planární graf („*Graf nazveme planární (rovinný), jestliže existuje jeho nakreslení v rovině (diagram grafu) takové, že žádné dvě hrany se v něm neprotínají.*“[6]), který by se rozložil na určenou plochu a na něj by se poté osázely vybrané komponenty.

Po prozkoumání tohoto řešení bylo rozhodnuto, že to není to nejpřímočařejší a že se na tuto úlohu musí jít jinak. Přišly dva jiné návrhy. První návrh spočíval v předem určené šachovnici, která by měla předem stanovený počet hran. Uživatel by poté mohl vybrat, kterou hranu chce vykreslit, na které hrany chce umístit jednotlivé součástky, a jak má vůbec graf vypadat.

Druhý návrh spočíval v jednotlivém navazování volných hran. Před vygenerováním vždy musí uživatel zadat, kolik hran bude zvolený obvod mít. Jako první se vygeneruje základní graf, který bude pevně daný počáteční velikostí

celého obvodu. Tento graf obsahuje právě čtyři hrany, které se od počátečního počtu odečtou. Pokud by zbývalo k vykreslení tři a více hran, pseudonáhodný generátor by určil, zda by se předchozí obvod zmenšil vertikálně nebo horizontálně a k němu by se z boku, respektive zespodu, přidal tvar složený ze tří hran, doplňující obvod na původní velikost. Takto by se pokračovalo do té doby, dokud by k vykreslení vždy zbývaly více než tři hrany. Jestliže by zbývaly dvě nebo jen jedna hrana, naposledy přidaný útvar by se rozdělil v jeho polovině a tam by se přidala předposlední, nebo v druhém případě poslední hrana. Pokud by zbývala ještě jedna hrana, tak ta by se přidala kolmo na poslední přidanou hranu.

Oba návrhy byly prozkoumány a k implementaci byl určen ten druhý. Nevýhodou u postupného vytváření obvodu je možná nepřesnost v počtu zadaných a vytvořených hran. Pokud uživatel zadá jiný počet hran než dělitelný třemi (po odečtení základních čtyř hran), vždy vznikne o dvě hrany, respektive o čtyři hrany (pokud je zbytek po dělení roven dvěma) větší graf. Tato nepřesnost bude vždy při vytvoření grafu jasně dána uživateli najevo, aby věděl, že výsledný graf může zobrazit více součástek než mohl předpokládat. První návrh sice tuto nevýhodu nemá, ale rozcházel se s hlavní myšlenkou této práce v tom, že by k vytvoření grafu byl vždy potřeba velký zásah uživatele, aby přesně vykreslil graf, který zamýšlel.

2.2 Generování rovnic

Existuje spousta návodů a praktik jak z výsledného obvodu vygenerovat příslušné rovnice. Na FIT ČVUT je vyučována metoda uzlových napětí. Opírá se o důsledky Kirchhoffových zákonů, přesněji o jeho první zákon, který říká, že součet všech vtékajících i vytékajících proudů do jednotlivých uzlů se rovná nule.[7]

Pro účely zjednodušení tohoto tématu studentům byl sepsán algoritmus na tvoření daných obvodových rovnic. Nejedná se sice o nejefektivnější postup, kdy vzniká co nejméně neznámých proměnných a rovnic, ale díky používání softwaru Mathematica nám nevadí, že počet rovnic není minimální. Tento postup se skládá ze sedmi kroků:

1. Uzly se označí a očíslují. Jako referenční uzel je nejlepší vybrat ten největší, v našem případě to je ten nejspodnější.

2. Označí se uzlová napětí vůči referenčnímu uzlu.

3. Označí se jednotlivé proudy u daných součástek.

4. Pro každý uzel se napíše rovnice zmíněného Kirchhoffova zákona, tedy $\sum I_{inNode} = \sum I_{outNode}$.

5. Pro každou součástku se napíše její rovnice. Tato rovnice vyjadřuje její napětí mezi svorkami dané součástky. Pro rezistor má rovnice tvar

$$u_R(t) = R * i_R(t),$$

pro kondenzátor má tvar

$$i_C(t) = C * u'_C(t)$$

a pro cívku má tvar

$$u_L(t) = L * i'_L(t),$$

kde $u_S(t), i_S(t)$, znamená napětí nebo proud součástky S v čase t, značky R, L, C značí hodnotu dané součástky a značka $'$ znamená derivaci dané proměnné.

6. Pokud se v rovnicích vyskytuje derivace, musí se napsat počáteční podmínka pro danou proměnnou.

7. Pro kontrolu správného provedení všech bodů poslouží jednoduchý součet. Pokud se rovná součet všech neznámých a jejich derivací s počtem rovnic, tak se nikde nevyskytla chyba a rovnice by měly být správné. [8]

Jak zde můžeme vidět, tento postup je jednoznačný a lehce algoritmizovatelný. Protože je tento návod vyučovaný přímo v předmětu BI-ČAO, pro který je tato aplikace vytvářena, bude implementován tento přesný postup.

2.3 Export/import do souboru

Ukládání výsledných obvodů je velice důležitá vlastnost výsledného programu, protože pro účely testování studentů je třeba je na nějakou dobu uchovat. Poté může vyučující znovu načíst ten, který zadal do určitého zápočtového nebo zkouškového testu a porovnat referenční výsledky s výsledky studentů.

Pro ukládání dat z programu existuje veliké množství formátů. Po konzultaci a debatě byl vybrán formát XML (extensible markup language). Tento formát je jednoduchý na přečtení jak libovolnou aplikací, tak i lidským okem v textovém editoru. Je také snadné z tohoto formátu získat veškeré potřebné informace. V grafické knihovně Qt je možnost využít třídy QXmlStreamReader a QXmlStreamWriter, které práci s tímto typem souborů ještě více ulehčují.

Pro tuto aplikaci bylo tedy jen zapotřebí vymyslet odpovídající strukturu elementů tohoto formátu. Kvůli ulehčení případného vizuálního náhledu vyučujícího do uloženého souboru, jsou v souboru uloženy informace o celkovém počtu hran a počtu všech součástek obsažených v uloženém obvodu. Po těchto informacích jsou v XML souboru uložené jednotlivé hrany, ze kterých se graf skládá. Jak můžeme vidět na obrázku 2.1, každá hrana obsahuje údaje o své

```xml
<?xml version="1.0"?>
<circuit>
 <numOfEdges>5</numOfEdges>
 <numOfResistors>2</numOfResistors>
 <numOfCondensers>1</numOfCondensers>
 <numOfCoils>1</numOfCoils>
 <numOfCurrents>0</numOfCurrents>
 <numOfPowers>1</numOfPowers>
 <edges>
 <edge x1="0" y1="0" x2="0" y2="400" value="ampl * Sin [Pi * f * t]" direction="0">U</edge>
 <edge x1="0" y1="400" x2="300" y2="400" value="" direction="1">Edge</edge>
 <edge x1="300" y1="400" x2="600" y2="400" value="" direction="1">Edge</edge>
 <edge x1="0" y1="0" x2="300" y2="0" value="100" direction="0">R2</edge>
 <edge x1="300" y1="0" x2="600" y2="0" value="1 n" direction="0">C</edge>
 <edge x1="600" y1="0" x2="600" y2="400" value="100" direction="1">R1</edge>
 <edge x1="300" y1="0" x2="300" y2="400" value="1 u" direction="1">L</edge>
 </edges>
 </circuit>
```

Obrázek 2.1: Příklad výsledné struktury souboru XML.

pozici (souřadnice prvního a druhého bodu), název součástky, která se na ní nachází (v případě žádné součástky je zde uvedeno „Edge“). Pokud tato hrana není prázdná, obsahuje údaje o hodnotě dané součástky a o směru proudu, který jí protéká.

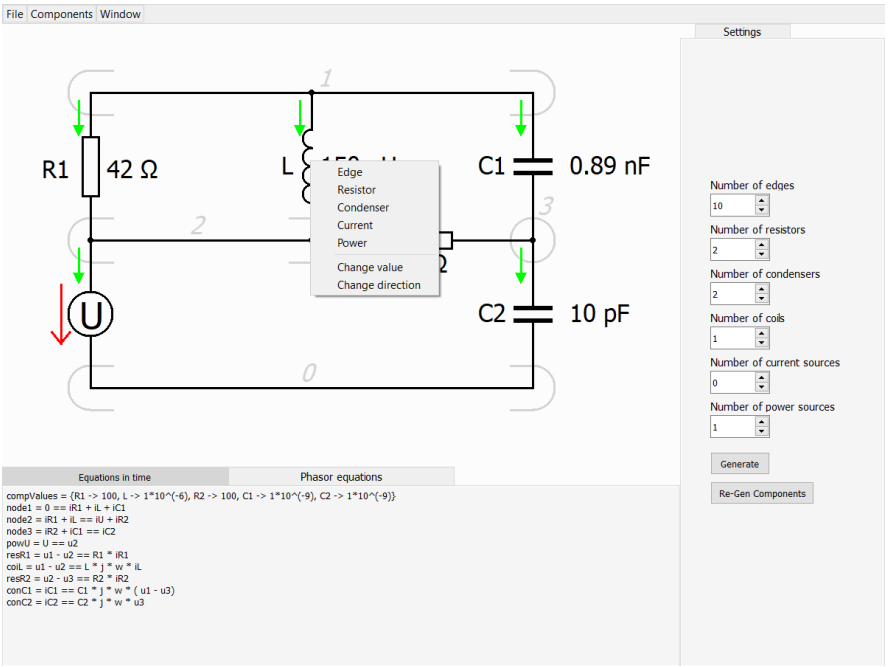
2.4 Uživatelské rozhraní

Ze všech implementačních úkolů byla tvorba uživatelského rozhraní jednoznačně nejdelší. Protože prvotní návrh aplikace nebyl jasně daný a nevědělo se přesně, které funkcionality budou v programu obsaženy, tak ani samotné uživatelské rozhraní nebylo dopředu přesně dáno.

Díky znalosti zadání bylo jen jasné, které základní prvky bude muset uživatelské rozhraní obsahovat. Bylo nutné dát uživateli možnost zvolit počet hran a počet jednotlivých komponentů k vytvoření nového obvodu. Protože to jsou jediné povinné údaje od uživatele, musely být umístěny na pevně dané, snadno přístupné a dobře viditelné místo. Pro správný chod programu bylo potřebné správně umístit výsledný graf a jeho vygenerované rovnice. To jsou nejdůležitější věci, které uživatel potřebuje vidět hned po vytvoření grafu, aby mohl srovnat výsledný obvod s jeho představami.

Drobné změny, jako změna proudu jednotlivých součástek, změna hodnoty daných komponentů nebo zapnutí a vypnutí zobrazení uzlů, nepatří k pravidelným a opakovaným činnostem. Proto nebyly umístěny na hlavní plochu programu, aby nemátly uživatele a ten viděl jen to nejpodstatnější. Tyto vedlejší funkcionality byly umístěny do horní lišty nebo do kontextového menu, které se otevře stisknutím pravého tlačítka u vybraného komponentu.

Po analýze těchto potřebných ovládacích a zobrazovacích prvků se navrhl finální vzhled uživatelského prostředí. Velikost aplikace se dopředu určila a tím pádem jsou všechny prvky statické a vždy na stejném místě. Největší část plochy (jak můžeme vidět na obrázku 2.2), přibližně dvě třetiny, zabírá prostor na vykreslení požadovaného obvodu. Při spuštění je tato plocha vždy prázdná.



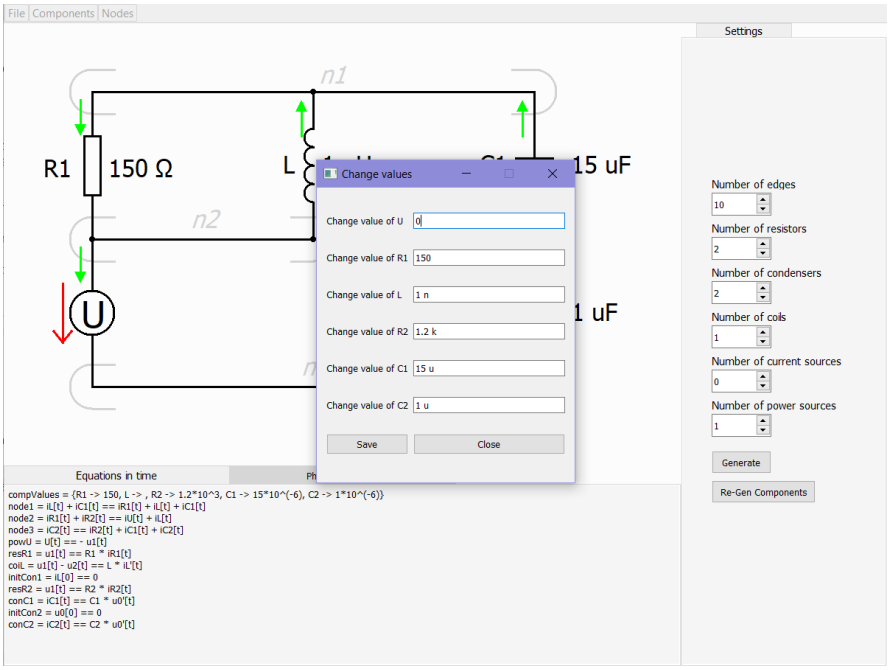
Obrázek 2.2: Výsledný vzhled uživatelského rozhraní.

Napravo od vykreslovací plochy se nachází panel na nastavení generování obvodu. Tento panel zabírá zbývající třetinu šířky aplikace. Zde uživatel zadává počet hran a počet jednotlivých komponentů. Kvůli problému popsaném v kapitole 2.1, se zde také objevuje upozornění v případě, když by se vygenerovalo více hran, než uživatel zadal.

Ve spodní části se pak nachází panel, který obsahuje dvě záložky. Je vysoký přibližně třetinu celkové výšky a rozpíná se až k vertikálnímu panelu s nastavením. Při každém vygenerování grafu se zde vypíší rovnice pro výpočtení obvodového napětí. Jedna záložka obsahuje rovnice v časové oblasti, v druhé jsou rovnice vypsané pomocí fázorů.

Pro vybrání a vyměnění jedné součástky za jiný druh, slouží pravé tlačítko myši. Jak lze vidět na obrázku 2.2, při této situaci se po kliknutí pravým tlačítkem objeví nové kontextové menu. Tato nabídka obsahuje výčet všech možností, na co se může daná součástka změnit. Vždy tedy obsahuje všechny součástky bez zrovna vybraného typu (Na obrázku 2.2 je zrovna pravým tlačítkem vybraná cívka. Tudíž v kontextovém menu tato součástka chybí.). Pod vizuálním oddělovačem se nachází tlačítka pro změnu proudu vybraného komponentu a pro změnu hodnoty té součástky.

Pokud by uživatel chtěl změnit více hodnot součástek najednou, je to také možné. Po stisknutí tlačítek „Components → Change Values“ umístěných v horní liště se objeví uživateli tabulka, kde bude mít zobrazené všechny součástky zadaného obvodu a aktuální hodnoty (viz obrázek 2.3). Po vyplnění



Obrázek 2.3: Hromadná změna parametrů součástek umístěných na obvodu.

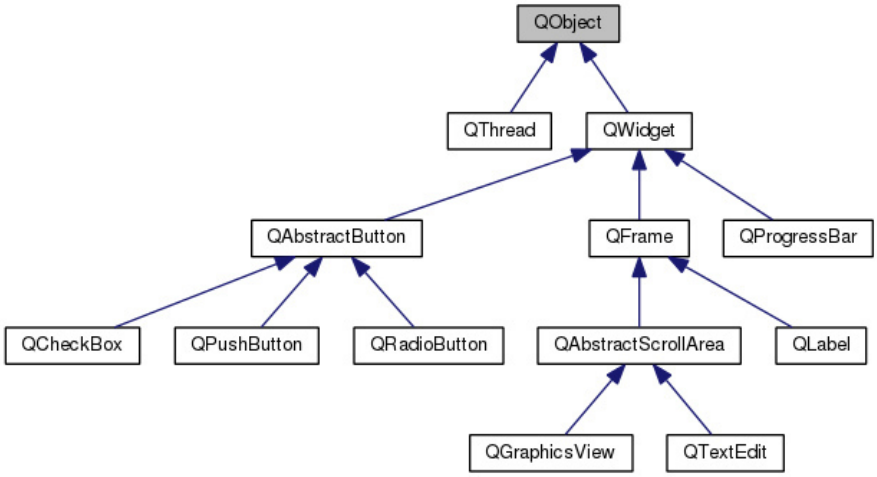
nových požadovaných hodnot se při zmáčknutí tlačítka „Save“ zkontrolují zadané hodnoty a, pokud odpovídají normě, uloží se.

Kapitola 3

Realizace

Jak již bylo zmíněno, tento program byl realizován za pomoci jazyka C++ a grafické knihovny Qt. Tato knihovna ulehčuje práci při tvorbě grafických aplikací, které pracují nejen s textovým vstupem a výstupem. Qt obsahuje spoustu tříd a metod, které urychlují návrh programů a zjednodušují tvorbu nových aplikací.

Jako základní kámen této knihovny se určuje třída `QObject`, která se stará o správu všech objektů.[9] Z ní postupně dědí další třídy, mezi které patří třeba `QWidget` (viz obrázek 3.1), která se používá pro uchování pozice jejích prvků, pro spojení s jinými třídami a také pro nastavení její velikosti. Z této třídy dědí spousta dalších tříd, které už jsou více specializované, jako například `QMainWindow` nebo `QSpinBox`. Třída `QMainWindow` se stará o zobrazení celého programu a všech prvků uvnitř sebe. Třída `QSpinBox` zobrazuje číselník, ve kterém může uživatel vyplnit nebo nalistovat číslo, které si přeje vložit.



Obrázek 3.1: Ukázka dědění tříd v knihovně Qt.[9]

Nespornou výhodou pro nezkušené tvůrce je i pomocný nástroj Qt Designer. Tento vestavěný software funguje na principu WYSIWIG („*What you see is what you get.*“[10] – To, co člověk vidí na obrazovce, je i to, co se vytvoří.) a pomůže programátorovi si představit výslednou aplikaci.

Pomocí tohoto nástroje bylo vytvořené uživatelské prostředí, které se řídilo návrhem uvedeném v kapitole 2.4. Na implementaci uživatelského prostředí (viz obrázek 2.2) byly zapotřebí tyto třídy z knihovny Qt:

- `QMainWindow`,
- `QWidget`,
- `QTabWidget`,
- `QLabel`,
- `QSpinBox`,
- `QPushButton`,
- `QPlainTextEdit`,
- `QMenu`,
- `QAction`.

Pro spojení uživatelského rozhraní se zbytkem programu, který funguje pod ním, byla vytvořena základní třída `circuitGenerator`. Ta dědí vlastnosti od třídy `QMainWidget`, takže obsahuje všechny vlastnosti pro vytvoření hlavního programu. V této třídě se nachází veškeré spojení signálů (zpráva od daného objektu, který signálem informuje, že se změnil jeho stav[9]) a slotů (funkce zodpovědná za příjem signálů a reakcí na ně[9]) pro interakci s uživatelem. Ve třídě `circuitGenerator` jsou také základní funkce celého programu, jako například vykreslení obvodu na určené místo nebo také vygenerování daného obvodu, jak je z názvu třídy patrné.

Pro další funkcionality programu jsou vytvořeny další pomocné třídy. Například pro generování rovnic slouží třída `equations` a pro uložení obvodů slouží třída `fileWork`. Pro případné změny hran, nebo změny hodnoty vybraných součástek slouží třída `changeAttributes`.

Implementace této aplikace probíhala v postupných vlnách. Jednotlivé vlny se nepřekrývaly a v každé jednotlivé vlně se vždy řešil jeden problém. Jakmile se zadaný problém vyřešil, zdokumentoval se a uchoval. Tím se předešlo programování více funkcionalit najednou a zbytečným chybám při implementaci.

Prvním krokem bylo naprogramování samostatného generování grafu bez jakýchkoliv komponentů. Poté se přidalo generování komponentů, následováno řešením ukládání výsledků do souborů a jejich zpětné načítání. V neposlední

řadě se řešila správnost generování obvodových rovnic a na závěr se implementovala možnost změny jednotlivých komponentů a změna jejich parametrů.

V následujících sekcích jsou popsané jednotlivé programovací vlny. Každá obsahuje stručný popis použitých algoritmů a postupů, případně obsahuje i ukázku vytvořeného kódu nebo použitých funkcí a metod.

3.1 Generování nového obvodu

První úkol k vytvoření generátoru obvodů byl právě samotný generátor. Signál pro vygenerování nového obvodu je vždy předán z tlačítka „Generate“, které je umístěné v panelu nastavení.

3.1.1 Generování hran

Signál z tlačítka „Generate“ přijme slot `handleGenerate`. Jako první se volá funkce na vytvoření tvaru grafu. Tato funkce naplní kontejner `std::vector` informacemi o každé hraně. Algoritmus pro toto vytvoření, byl popsán v kapitole 2.1. Zkráceně se jedná o založení základního útvaru, který má rozměr určený parametry pro největší možný graf a poté se náhodně přidávají obdélníky z pravé nebo dolní strany, dokud je dostatek hran.

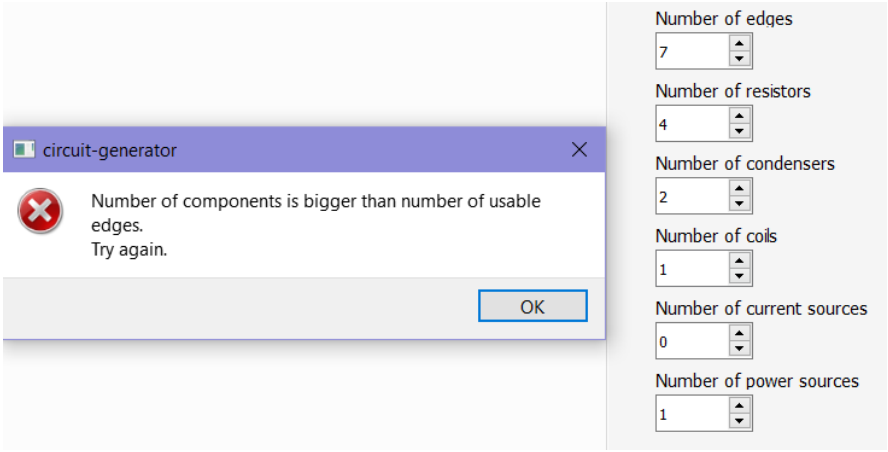
Pro tento algoritmus platí jeden podstatný předpoklad. Je důležité mít kvalitní a – pokud je to možné – náhodný (nebo aspoň pseudo-náhodný) generátor čísel. Byla možnost použít základní generátor náhodných čísel, který je implementovaný v knihovně `stdlib.h` (standardní knihovna pro obecné funkce, jako je například správa paměti nebo aritmetika s celými čísly[11]). Ale bohužel tento generátor není dostatečně kvalitní. Jeho problém spočívá v tom, že po inicializaci opakuje vždy ta stejná čísla, tudíž jeho náhodnost je nulová a po každém zapnutí aplikace by výsledné grafy byly stejné.

Proto se po domluvě s vedoucím bakalářské práce vybral jiný zdroj náhodnosti. Jako vhodný zdroj se vybral 256 stavový celulární automat [12] (dynamický matematický model, kde se v diskrétním čase prvek automatu vyvíjí podle jeho hodnoty a hodnoty jeho sousedů[13]). Tento generátor stačí nainicializovat při zapnutí aplikace, pro náš případ tedy v konstruktoru třídy `circuitGenerator`, náhodným číslem (pro inicializaci nám postačí knihovna `random.h`) a poté se vždy jen dotázat na nový náhodný prvek.

3.1.2 Vygenerování komponentů

Po vygenerování tvaru obvodu se program ujistí, zda uživatel nezadal víc komponentů k vykreslení než je volných hran. S vedoucím práce bylo dohodnuto, že na nejspodnější hranu se nebude umisťovat žádná součástka. Proto se v podmínce kontroluje to, zda není zadáno více součástek než je volných hran. Výsledná nerovnice pro kontrolu této podmínky tedy vypadá:

$$\sum Components \leq \sum UpperEdges,$$



Obrázek 3.2: Příklad chybně zadaného počtu hran a komponentů.

kde *Components* značí součástky k vykreslení a *UpperEdges* značí hrany, které se nenacházejí úplně nejníže v horizontálně poloze.

Pokud je tato nerovnice splněna, program projde všechny součástky, které chce uživatel vykreslit, a pro každou z nich náhodně vygeneruje pozici v grafu. I tato funkce používá ke generování náhodného čísla celulární automat[12], aby se stejné rozmístění součástek opakovalo co nejméně.

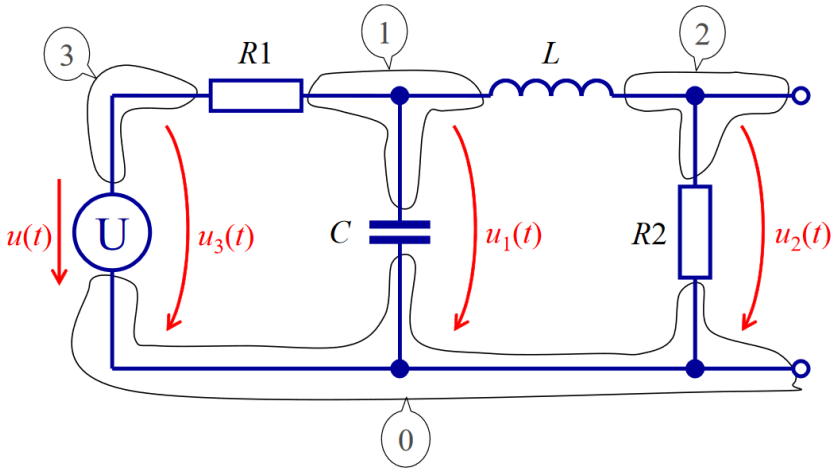
Po dokončení této funkce je vytvořen nový obvod, který má téměř všechny potřebné údaje k tomu, aby byl vykreslen. Jediné co mu chybí, je údaj o všech uzlech, které se v grafu nachází. Tuto skutečnost ošetřuje následující funkce, která zavolá třídu `equations`, aby našla v tomto obvodu všechny uzly a vypsala všechny nutné rovnice. Detailnější popis těchto funkcí je sepsán v kapitole 3.4.

3.1.3 Vykreslení obvodu

Po roztřídění všech hran do jednotlivých uzlů je konečně možné vše vykreslit. V knihovně Qt je možné vykreslovat cokoli za pomocí funkce `paintEvent`, která je součástí třídy `QWidget`. Protože je deklarována i v rodičovské třídě, tak je v této třídě zděděná a deklarovaná jako „override“. To znamená, že pokud se zavolá tato funkce, přehlíží se její implementace v rodičovské třídě a vykonává se implementace pouze z třídy potomka.

K nastavení vlastností vykreslovaného grafu slouží třída `QPen`. V této třídě lze nastavit například šířku tvořené čáry, styl čáry (jestli je plná, tečkovaná nebo například čárkovaná) nebo barvu. Obdobně slouží třída `QFont`, ale jak už z názvu vypovídá, ta ovlivňuje psaný text.

Pro správné vykreslení celého obvodu je nutné ošetřit jednu věc. Je nutné zjistit, jak velké mají být součástky. Pro to nám slouží dvě funkce, které se volají na začátku funkce `paintEvent`.



Obrázek 3.3: Názorný příklad jak označit jednotlivé uzly v obvodu.[8]

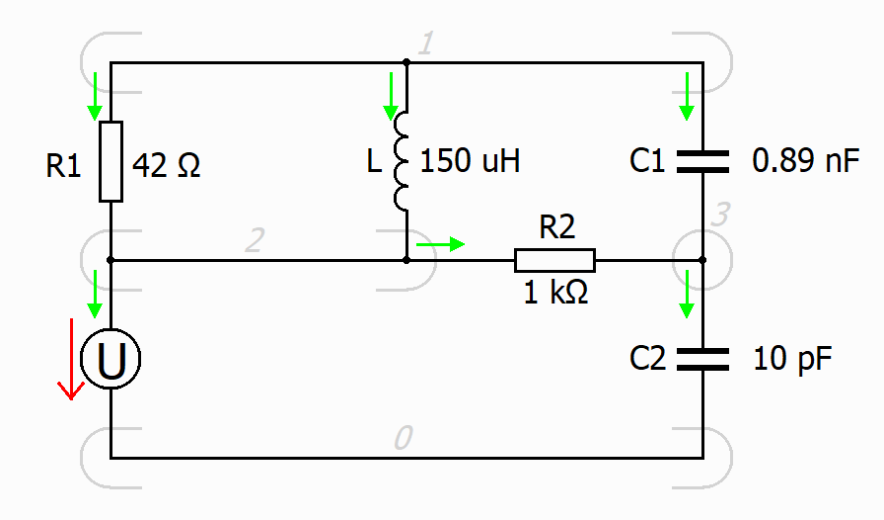
První funkce zjistí nejmenší hranu, na které je položena součástka a podle té vypočítá velikost všech typů součástek. Pro rezistor například platí, že samotná součástka zabírá čtyři desetiny délky hrany. Na rozdíl od kondenzátoru, kterému se mění pouze šířka součástky, se u rezistoru změnou velikosti mění vždy jak šířka, tak i výška.

Poté, co se vypočítají velikosti všech komponentů, je možné všechny součástky předělat do tvarů, které dokáže vygenerovat knihovna Qt. Pro vykreslení slouží dva pomocné kontejnery. Pokud hrana neobsahuje žádnou součástku, automaticky je uložena do prvního kontejneru typu `std::vector`. Druhý kontejner je stejného typu, ale slouží pro hrany, na kterých je posazena jakákoli součástka. Všechny části součástek, které se dají vykreslit přímkou, jsou uloženy do druhého kontejneru s počátečním bodem a koncovým bodem dané přímky. Pokud je tvar komponentu složitější, uloží se jen rozsah, odkud a kam až má být část komponentu vykreslena, a jaký typ komponentu to je.

Nakreslení uzlů

K vykreslení uzlů je nutné vědět, co to uzel je a jak takový uzel vypadá. Neformálně řečeno, jeden uzel je kus vodiče, který je ohraničený připojenými součástkami. Ukázku označení uzlů v obvodu můžeme vidět na obrázku 3.3. Uložení jednotlivých uzlů je implementováno kontejnerem `QSet`. Tento kontejner ukládá své prvky neuspořádaně, což znamená, že nejsou přístupné v tom pořadí, v jakém byly uloženy. Výhodou tohoto kontejneru je to, že se v něm rychle vyhledává a každá hodnota v něm může být pouze jednou, což je ideální pro ukládání uzlů, kde každý vrchol může být v uzlu také pouze jednou.

Všechny uzly jsou uloženy v kontejneru `QVector`. Na rozdíl od `QSet`, tento kontejner ukládá prvky v pořadí, ve kterém se vkládaly. To nám usnadní



Obrázek 3.4: Ukázka obvodu se všemi typy uzlů.

číslování obvodů. První uzel je vždy referenční, v našem případě tedy ten uzel, který se nachází okolo nejspodnější hrany. Dále jsou uzly číslovány zleva doprava a od vrchních uzlů po ty nejnižší.

Pro vykreslení stačí projít kontejner `QVector`, který má uložené všechny uzly v obvodu. Pokud je uzel tvořen jediným bodem, je okolo něho pouze vykreslen kruh a jeho číslo je napsáno nad kruh a je lehce posunuto doprava. V případě uzlu, který je tvořený dvěma body, se vytvoří v každém bodě půlkruh, který má na svých stranách přímky. Ty zvýrazní příslušnost k danému uzlu (viz obrázek 3.4). Číslo uzlu se poté vypisuje mezi tyto dva body.

Pro uzly, které mají více bodů, je to složitější. V zadaném uzlu se musí porovnat každý bod s každým dalším bodem. Pokud má nějaký z nich pouze jednoho souseda, vytvoří se půlkruh, který je stejný jako v případě uzlů se dvěma body, mířící směrem k jeho sousedovi. V celém procházení bodů si program pamatuje bod s největším počtem sousedů. Právě k tomuto bodu se po konci hledání vypíše číslo uzlu. Na obrázku 3.4 je vidět například uzel číslo 1, který obsahuje právě 3 body.

Nakreslení součástek

Pro vykreslení součástek už máme připravené dva kontejnery. První spravuje volné hrany a druhý má uložené komponenty, které jsou rozebrané na jednotlivé části. Nejprve se tedy prolistuje první kontejner a pro každý záznam, který se najde, se vykreslí daná čára. Tím se vykreslí všechny neobsazené hrany. U druhého kontejneru může nastat více možností. Pokud je zde uložená přímka, jen se vykreslí, stejně tak, jako to probíhalo v předchozím případě.

Pokud je zde ale uloženo něco složitějšího, program to detekuje a vykreslí správný tvar. Mezi tyto tvary patří cívka a pak také zdroje proudu a napětí. Protože se piktogram cívky značí jako čtyři navazující půlkruhy a piktogram zdrojů je celý kruh, musí se k nim přistupovat jinak.

V případě cívky program vysází od startovacího bodu 4 půlkruhy pomocí funkce `drawArc`, která má šest parametrů. Potřebuje vědět střed vykreslovaného útvaru (souřadnice x a y), poloměry vertikální a horizontální osy a poté úhel, ve kterém má začít vykreslovat a úhel, jak dlouho má vykreslovat. Pro vykreslení kruhu jsou potřeba jen 4 parametry. Jsou to ty stejné, které se používají u `drawArc`, jen zde není třeba zadávat informace o úhlu.

Vypsání popisků

Po zobrazení všech komponentů a hran je třeba je všechny popsat. Pokud uživatel zvolí, je třeba vypsat i jejich hodnoty. V této části stačí projít kontejner s uloženými hranami a pro každou součástku vypsat její název. Pokud je komponent v horizontální pozici, píše se název nad něj. V případě komponentu ve vertikální pozici se píše na levou stranu. Hodnota dané součástky se píše na opačnou stranu. Pro horizontální polohu se tedy píše pod součástku, jinak se píše na pravou stranu.

3.2 Přegenerování pozic součástek

Při tvorbě obvodu se může stát, že se uživateli zalíbí výsledný tvar nového obvodu, ale nebude se mu líbit počet nebo rozmístění jednotlivých součástek. Pro tuto situaci se nachází v uživatelském prostředí tlačítko „Re-Generate Components“. Toto tlačítko zachová tvar vygenerovaného obvodu, ale smaže všechny nanesené součástky a poté je znovu náhodně zasadí do grafu.

Signál, který se vygeneruje stisknutím tohoto tlačítka, zachytává funkce `handleReGenerate`. Tato funkce je obdobná jako zmíněná funkce na vygenerování nového obvodu, ale chybí jí vygenerování nového tvaru. Tudíž implementačně obsahuje ty samé části, jaké jsou zmíněné v kapitolách 3.1.2 a 3.1.3.

3.3 Změna vybraných součástek

Při navrhování nového obvodu může vzniknout situace, že se sice výsledný tvar uživateli líbí, ale není na 100 % spokojený s tím, jak vypadá poskládání komponentů na něm. V tom případě je možné využít změny jednotlivých součástek za jiné. Uživateli stačí, aby najel myší nad součástku, kterou chce změnit a stisknout pravé tlačítko.

Spojení pravého tlačítka s novým menu

```
this->setContextMenuPolicy(Qt::CustomContextMenu);
connect(this,SIGNAL(customContextMenuRequested(const QPoint&)),
        this,SLOT(ShowContextMenu(const QPoint&)));
```

Malá ukázka kódu ukazuje, jak se vytváří nové menu pro pravé tlačítko myši. První řádek nastavuje pro aktuální třídu (`circuitGenerator`) to, že při vzniknutí signálu pro vytvoření kontextového menu se bude volat menu, které je vytvořeno uživatelem. Na druhém řádku probíhá spojení popsaného signálu se slotem dané třídy. Funkce `connect` má 4 argumenty. První argument je ukazatel na třídu odesílatele signálu, další je ukazatel na daný signál, třetí je ukazatel na třídu příjemce a poslední je ukazatel na slot, který provádí určené reakce na signál.

V našem případě je odesílatel i příjemce stejná třída a to dokonce stejná jako ta, ze které se tato funkce volá, proto se používá symbol `this`. Slot `ShowContextMenu(const QPoint &)` je vlastní funkce, která ovládá otevření nového menu. Její vstupní parametr předává informace o tom, kde se myš nacházela v době, když byla stisknutá.

Funkce `ShowContextMenu` nejdříve nalezne součástku, na které uživatel zmáčkl pravé tlačítko. Funkce má nastavené krajní případy, aby uživatel nemusel klikat přímo na daný pixel, ale mohl kliknout i o kousek vedle. Pokud se nenalezne komponent shodný s pozicí myši, nic se neprovede a funkce končí. V opačném případě se vytvoří nové kontextové menu. Menu se skládá z tlačítek pro změnu typu součástky a pod oddělovačem se nachází tlačítka pro změnu hodnoty a poté pro změnu směru.

Pokud uživatel klikne na prázdnou hranu, na výběr se ukáží pouze typy součástek. Hodnota ani směr prázdné součástky se nedají měnit. Podobné pravidlo platí pro základní zdroj napětí. Po domluvě s vedoucím bakalářské práce bylo dohodnuto, že jediný povinný komponent celého obvodu je jeden zdroj napětí. Je vertikálně umístěný na levou hranu a pokud je těchto hran vlevo více, tak na tu nejspodnější. Tato součástka se nedá měnit, fixně je nastavená jako zdroj napětí. Ale stále se u ní dá měnit hodnota nebo také směr generovaného napětí.

3.3.1 Změna typu

Po kliknutí na změnu součástky na jiný typ se zavolá slot `hanChangeToXXX`, kde `XXX` je název součástky, na kterou jí chce uživatel změnit. Tento slot vytvoří novou třídu `changeAttributes` a zavolá její funkci pro změnu součástky. Všechny funkce pro změnu součástek vypadají až na lehké drobnosti stejně, proto v této kapitole budu pro příklad ukazovat změnu libovolné součástky na rezistor:

Slot pro změnu komponentu na rezistor

```
void circuitGenerator::hanChangeToRes() {
  changeAttributes chA;
  std::vector <int> vecComp = getValuesIntoVector();
  chA.changeToRes(&pEdges, &vecComp, foundEdge, foundType);
  setValuesFromVector(vecComp);
  buildCircuit(false);
}
```

V ukázce se vyskytuje kontejner `std::vector`, který uchovává počty všech součástek ve vygenerovaném obvodu. Tyto informace by se daly předat jednotlivě, ale pro přehlednost se raději předává tento kontejner. Funkce `changeToRes` dostává 4 parametry. První je kontejner s uloženými hranami celého obvodu, druhý je dočasný kontejner s počty jednotlivých součástek, a poté následuje index nalezené hrany a jako poslední je předáván její typ.

První věc, která se provádí ve funkci `changeToRes`, je přejmenování stávající součástky na novou. Na začátku se inkrementuje proměnná držící hodnotu o počtu rezistorů. Pokud je nová hodnota rovna jedné, tak to znamená, že se rezistor v grafu ještě neobjevil a toto je první výskyt. Tím pádem stačí jako název tohoto komponentu dát jeho typ bez očíslování (pokud se v grafu nachází jediný zástupce nějakého typu, obvykle se u něj neuvádí číslo, jako například na obrázku 3.4 u cívky). Pokud už v grafu byly dva a více zástupců od této součástky, tak je úkol znovu lehký a stačí jako název nové součástky nastavit její typ a počet všech zástupců dané součástky. Komplikovanější případ nastává, pokud byl v grafu před přidáním nové součástky právě jeden jedinec stejného typu. V tom případě se tento jedinec musí najít a musí se přidat k jeho názvu i pořadové číslo.

Po přejmenování tohoto komponentu už stačí jen zmenšit počet výskytů u bývalé součástky. Pro tuto činnost jsou implementované funkce pro každou součástku, pojmenované `xxxxMinusOne`, kde hodnota `xxxx` značí název součástky. Pro ukázku předpokládejme, že uživatel chtěl změnit cívku na rezistor a proto je zde ukázka snížení počtu cívek:

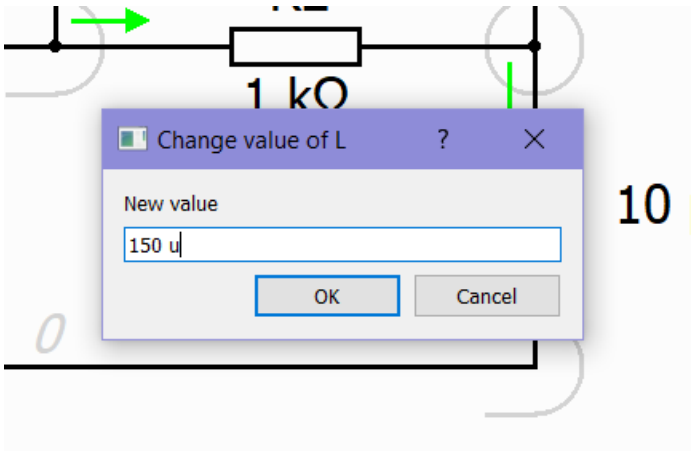
Funkce pro snížení počtu cívek

```
int changeAttributes::coilMinusOne(std::vector<Edges> *pEdges,
                                   int pNumOfCoils,
                                   std::string foundType) {

  int drawedCoi = pNumOfCoils;
  pNumOfCoils--;

  if (pNumOfCoils) { //if some coil is still remaining
    if (stoi(foundType.substr(1)) == drawedCoi) {
                //removed coil with biggest number
      if (pNumOfCoils == 1) { //find last coil and change it
```

```
        for(auto it=pEdges->begin();it!=pEdges->end();++it){
          if (it->t_type.find("L") == 0) {
            it->t_type = "L";
            break;
          }
        }
      }
    }
    else {          //it's needed to change all numbers
      drawedCoi = 1;
      for(auto it=pEdges->begin();it!=pEdges->end();++it){
        if (it->t_type.find("L") == 0) {
          if (pNumOfCoils == 1) {   //only one coil remains
            it->t_type = "L";
            break;
          }
          else                  //rename all remaining coils
            it->t_type = "L" + std::to_string(drawedCoi++);
          if (drawedCoi == pNumOfCoils + 1)
            break;
        }
      }
    }
  }
  return pNumOfCoils;
}
```

V ukázce lze vidět, jak probíhá snižování počtu součástek. Pokud stále zbývá více než jedna součástka a byla odebrána součástka s nejvyšším číslem, není třeba nic měnit. Pokud byla odebrána součástka s nejvyšším číslem a už zbývá pouze jedna, tak je třeba vyhledat tu poslední zbývající a změnit jí název na samostatný typ součástky. Pokud se ale neodebrala poslední součástka, je nutné projít všechny stejné součástky a postupně je znovu očíslovat, aby nevznikaly mezi čísly mezery.

Po návratu z funkce `changeToRes` je třeba uložit změněné počty komponentů. Po uložení už stačí pouze zaktualizovat vykreslený obvod, aby se změna součástky objevila i uživateli. K tomuto kroku slouží funkce `buildCircuit`. Její jediný parametr slouží k určení, zda má být v novém obvodu změněno nastavení směru proudu jednotlivých součástek. Změna směru všech součástek může nastat ve dvou případech. Pokud se generuje nový graf nebo se generují všechny nové součástky, pak je tato hodnota nastavena na `true`. V ostatních případech se se směry proudů nic neděje.



Obrázek 3.5: Ukázka dialogu ze třídy `QInputDialog`.

3.3.2 Změna hodnoty

Pokud uživatel stiskne tlačítko pro změnu hodnoty komponentu, zavolá se slot `hanChangeValue`. Tento slot vytvoří novou instanci třídy `changeAttributes`, ze které se zavolá funkce `changeValue`. Volaná funkce přijímá tři parametry a vrací právě jednu hodnotu. První přijímaný parametr je ukazatel na aktuální třídu, druhý je název zvolené součástky a třetí je aktuální hodnota součástky, která se bude měnit. První parametr umožňuje zobrazit nové okno pro změnu součástky, zbylé dva jsou důležité pro samotnou změnu. Návratovou hodnotou této funkce je nová hodnota, kterou zadá uživatel.

Funkce `changeValue` vytvoří po svém zavolání nový dialog, který je instancí třídy `QInputDialog` (ukázka na obrázku 3.5). Tento dialog potřebuje ke svému vytvoření ukazatel na třídu, ve které se může otevřít. Právě k tomuto účelu nám slouží první parametr, který se předal z výchozího slotu. Po otevření dialogu je uživatel vyzván zadat novou hodnotu pro změnu součástky. Pro uložení nové hodnoty uživatel musí zmáčknout tlačítko „Ok“. Po stisknutí tohoto tlačítka proběhne kontrola zadaného vstupu.

Pro kontrolu správné hodnoty komponentu slouží samostatná funkce. Uživatel má možnost zadat číslo typu „double“ (hodnoty můžou vypadat například: $15, 0.002$ nebo $15e2$) nebo může zadat číslo i s pomocnou předponou. Nejmenší možnou zadanou předponou je „femto“ (násobek 10^{-15}), největší je pak „peta“ (násobek 10^{15}). Pokud uživatel zadal jakékoli jiné znaky nebo nezadal nic, funkce toto detekuje a zobrazí uživateli zprávu, kde ho upozorní na nesprávně zadanou hodnotu. Uživatel má poté možnost zadat hodnotu znovu. Pokud je hodnota správná, aplikace hodnotu uloží a zavře dialogové okno.

Po zavření dialogového okna funkce pouze vrátí hodnotu, kterou uživatel zadal. Po vrácení do hlavního programu následuje uložení návratové hodnoty k dané součástce a poté vykreslení grafu s novou hodnotou součástky.

3.3.3 Změna směru

Poslední možností, kterou si uživatel může z kontextového menu vybrat, je změna směru součástky. Toto tlačítko je spojeno se slotem `hanChangeDir`. Tento slot přijme signál vyvolaný stisknutím tlačítka a protože je směr definovaný pouze jako hodnota `bool`, tak ho pouze zneguje. Poté už stačí pouze obvod překreslit, aby byla tato změna vidět. Proto se zavolá funkce `buildCircuit` s parametrem `false`.

3.4 Generování rovnic

Jedním z cílů této práce bylo vygenerování příslušných obvodových rovnic, které se bez úprav dají zkopírovat do softwaru Mathematica.[14] Po otestování syntaxe programu Mathematica bylo zjištěno, že pro tyto účely stačí vhodně zvolené klasické textové prostředí. Všechny potřebné symboly, které jsou potřeba pro popis rovnic, se pro přijmutí Mathematicou dají napsat klasickými znaky.

O generování rovnic a o všechno, co je pro jejich generování potřeba, se stará třída `equations`. Jak již bylo zmíněno v kapitole 3.1.2, tato třída je volaná ve funkcích předcházejících samotné vykreslení obvodu, protože některé informace z generování rovnic jsou potřebné, například pro vykreslení uzlů. Samostatná třída `equations` obsahuje čtyři funkce, dvě z nich jsou veřejné, volané z hlavní třídy, a dvě jsou soukromé. Třída také obsahuje jeden pomocný kontejner, kde je uložený každý bod z grafu.

3.4.1 Uložení všech bodů

Jako první se vždy zavolá funkce `getVertexes`. Tato funkce má dva parametry. Prvním parametrem je hodnota `setDirection`, která určuje, zda má mít nový obvod stejný směr proudů součástek jako doposud, nebo jestli je potřeba projít celý kontejner a změnit směry proudů dle základního zdroje. Druhým parametrem je kontejner, ve kterém jsou uložené všechny hrany v obvodu. Tato funkce tedy projde každou hranu a z každé si uloží jak počáteční, tak i koncový bod. K uložení těchto informací se využívá kontejner `QSet`, takže je zajištěna jedinečnost záznamů, protože se do tohoto kontejneru neuloží záznam, který už tam vložený někdy byl.

Pokud byl první vstupní argument pravdivý, tak se pro každou hranu nastaví proud tak, aby korespondoval s dohodnutými směry proudy. Proud ve vertikálních součástkách má směr zeshora dolů, v horizontálních součástkách je směr zleva doprava.

3.4.2 Vypsání rovnic

Po dokončení předchozí funkce se zavolá funkce `generateEquations`. Ta má pět vstupních parametrů. První parametr obsahuje kontejner se všemi hranami obvodu. Druhý a třetí parametr je reference na kontejnery spravující uložení uzlů a jejich hran. Tyto kontejnery jsou zatím prázdné, ale protože informace tvořené v právě volané funkci jsou potřeba i pro běh hlavní třídy, jsou předány referencí, aby se změny projevily i v hlavní třídě. Čtvrtý a pátý parametr předává funkci ukazatele na plochy, kde se vypisují rovnice.

Výpis součástek

Pro správné a korektní vypočtení uzlových rovnic je potřeba znát i hodnoty součástek. S vedoucím práce bylo domluveno, že budou mezi rovnice umístěné i pravidla pro dosazení hodnot součástek do jejich proměnných. Pro vypsání těchto parametrů je vytvořena soukromá funkce. Jejím úkolem je projít všechny hrany a v případě, že narazí na součástku s platnou hodnotou, vypíše ji do obou ploch zároveň. Každá z ploch obstarává jiný typ rovnic, proto je potřeba vypsat tyto informace na obě plochy. Před jednotlivým vypsáním hodnot se vždy zavolá funkce na změnu hodnoty. Tato funkce zaručí správnou syntaxi výsledných hodnot vzhledem k jazyku systému Mathematica. Pokud se hodnota skládá pouze z čísla, nic se nemění a funkce vrací nezměněné číslo. Pokud ale za číslem následuje jednotková předpona, funkce předponu změní na ekvivalentní násobek exponentu 10 (například z hodnoty 5 nF se stane $5 * 10^{-9}$ F).

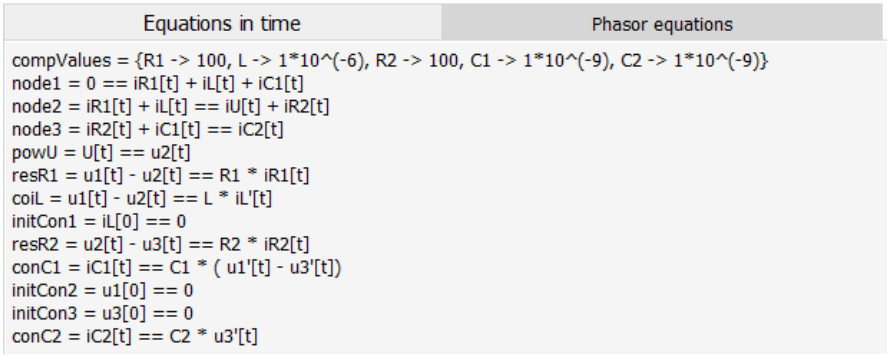
Uzlové rovnice

Jak bylo popsáno v kapitole 2.2, každému uzlu náleží jedna rovnice. Tvar této rovnice vypadá následovně:

$$\sum I_{inNode} = \sum I_{outNode}.$$

Protože v tuto chvíli ještě nejsou žádné uzly uložené, budeme vypisovat rovnice při hledání a tvorbě uzlů. Hledání uzlů je velice podobné jako hledání souvislých komponent v grafu („*Souvislé komponenty jsou různé „místnosti“, mezi nimiž nelze přejít (alespoň bez prokopání nějaké zdi)*“[15]). V našem případě nemáme zdi a místnosti, ale místnosti zde představují body a volné hrany okolo nich a zdi jsou zde nahrazené součástkami.

Pro hledání uzlů potřebujeme jisté struktury, abychom je mohli korektně vyhledávat. Je potřeba mít kontejner se všemi uloženými body. Tento kontejner už jsme naplnili ve funkci `getVertexes`. Poté potřebujeme mít vytvořenou frontu, kam vkládáme nalezené sousedy vybraného bodu. Pro vypsání vtékajících a vytékajících proudů je také potřeba uchovávat si v průběhu hledání uzlů součástky, které s daným uzlem hraničí. Proto se tyto součástky ukládají do dvou kontejnerů, jeden pro vtékající, druhý pro vytékající proudy součástek.



Obrázek 3.6: Ukázka časových rovnic pro obvod ukázaný na obrázku 3.4.

První hledaný uzel je vždy referenční. Podle konvence, která je určena na předmětu BI-ČAO, je tento uzel ten nejspodnější v celém obvodu. Vybere se bod, který se nachází úplně nejníže na levé straně, přímo pod základním zdrojem napětí. Ten se vždy bude nacházet v referenčním uzlu. Tento bod se přidá do fronty jako první a zároveň se smaže z kontejneru se všemi volnými body, aby se už znovu nemohl vybrat (každý bod náleží pouze jednomu uzlu). Poté se prochází hrany, které mají jeden z konců umístěný v tomto bodě. Pokud je hrana obsazená, uloží se název její součástky do kontejneru pro vtékající, respektive vytékající proudy. V druhém případě, kdy je hrana volná, se druhý konec této hrany uloží do fronty, vyřadí se z volných bodů a vloží se k ostatním bodům hledaného uzlu. Po projití všech hran se z fronty odstraní první prvek, ke kterému se hledali sousedé. Takto probíhá hledání do té doby, dokud existuje prvek ve frontě.

Jakmile toto hledání skončí a fronta je prázdná, stačí vše uložit a vypsat. Kontejner se všemi body, které patří do jednoho uzlu, stačí uložit do kontejneru `nodes`, který spravuje všechny uzly v obvodu. Ukázkové vypsání uzlových rovnic můžeme vidět na obrázcích 3.6 a 3.7. Na obrázcích lze vidět, že se vypisují všechny uzly, kromě uzlu referenčního. To je dáno tím, že rovnice pro referenční uzel je lineárně závislá na zbývajících uzlech a proto je zbytečné jí explicitně vypisovat.[8]

Rovnice začíná názvem uzlu s jeho číslem. Poté se vypíší vtékající proudy, které se rovnají vytékajícím proudům. V případě, že do uzlu žádný proud nevtéká nebo nevytéká, napíše se místo proudů 0. Vypsaný název uzlu je důležitý pro software Mathematica, kde pojmenování rovnic usnadňuje manipulaci s více rovnicemi. Časové rovnice u sebe obsahují parametr `[t]`, který značí, že měření probíhá vždy diskrétně, tedy v jednom časovém okamžiku. Fázory pracují jiným systémem, kdy všechna napětí po odeznění počátečního přechodného děje dostanou stejnou frekvenci jako zdroj a proto není nutné časovou konstantu psát.[8]

Po vypsání jednoho uzlu začíná hledání dalšího. Jako první se zvolí bod,

Equations in time	Phasor equations

```
compValues = {R1 -> 100, L -> 1*10^(-6), R2 -> 100, C1 -> 1*10^(-9), C2 -> 1*10^(-9)}
node1 = 0 == iR1 + iL + iC1
node2 = iR1 + iL == iU + iR2
node3 = iR2 + iC1 == iC2
powU = U == u2
resR1 = u1 - u2 == R1 * iR1
coiL = u1 - u2 == L * j * w * iL
resR2 = u2 - u3 == R2 * iR2
conC1 = iC1 == C1 * j * w * ( u1 - u3)
conC2 = iC2 == C2 * j * w * u3
```

Obrázek 3.7: Ukázka rovnic pomocí fázorů pro obvod ukázaný na obrázku 3.4.

který ještě nebyl použit a nachází se co nejvíce nahoře a vlevo. Tím se zaručí seřazenost uzlů zleva doprava a ze shora dolů. Toto hledání uzlů a vypisování jejich proudů probíhá do té doby, dokud existuje nějaký nepokrytý bod v obvodu.

Rovnice součástek

Po vypsání všech uzlových rovnic zbývá jen vypsat rovnice všech součástek, které jsou přítomné ve vykresleném obvodu. Tvary rovnic pro jednotlivé součástky jsou přesně dané, a to takto:

- Pro rezistor platí rovnice: $u_R(t) = R * i_R(t)$.
- Pro kondenzátor platí rovnice: $i_C(t) = C * u'_C(t)$.
- Pro cívku platí: $u_L(t) = L * i'_L(t)$.

Proud i a hodnoty součástek R, L, C jsou vázané k dané součástce a tím pádem se nemění. Problém nastává u napětí. Každý uzel má totiž jiné napětí vůči referenčnímu uzlu. Při nalezení součástky je tedy třeba, aby program zjistil, z jakého uzlu součástka vede a také do jakého uzlu teče její proud.

Hledání uzlu obsahující bod (x1, y1)

```
int nodeVS;
for ( nodeVS = 0; nodeVS < nodes->size(); nodeVS++ ) {
  if (*(nodes->begin() + nodeVS)->find(qMakePair(x1, y1)) !=
      *(nodes->begin() + nodeVS)->end()) {
    break;
  }
}
```

V tomto jednoduchém cyklu můžeme vidět vyhledání uzlu, ve kterém leží bod (x1, y1). Hlavní cyklus iteruje přes proměnnou `nodeVS` (zkratka pro označení nodeVertexStart), která nabývá stejné velikosti jako je celkový počet uzlů v obvodu. V každém cyklu se program ptá, zda v tomto vybraném uzlu se

nenachází vybraný bod. Pokud se nenachází, počítadlo se inkrementuje a tím pádem se cyklus ptá dalšího uzlu. Funkce `find` vrací při úspěšném nalezení prvku ukazatel na daný prvek v kontejneru a v případě neúspěchu vrací ukazatel na konec kontejneru. Proto se v podmínce uvnitř cyklu kontroluje, jestli není vrácený ukazatel jiný než poslední prvek kontejneru. Pokud ano, tato hodnota se uloží a cyklus se opustí.

Tento cyklus se vykoná i pro konečný bod součástky, tedy pro bod (x2, y2). Po vyhledání těchto dvou bodů je důležité se ujistit, že proud součástky opravdu teče od počátečního bodu ke koncovému. Pokud je tomu naopak, program body otočí. Po této kontrole už program vyhledá správné rovnice určené pro dané součástky a vypíše je. Pokud se součástka jednou stranou nachází v referenčním uzlu, jeho napětí se nevypisuje (můžeme vidět na obrázku 3.6 u kondenzátoru C2.).

3.5 Práce se soubory

V rámci cílů této práce bylo také nutné dokázat vygenerovaný obvod uložit a zpětně načíst. V kapitole 2.3 byl popsán postup uložení a formát souboru XML, do kterého se výsledný obvod ukládá. O práci se soubory se v tomto programu stará třída `fileWork`. Tato třída obsahuje dvě veřejné funkce. Jedna funkce obstarává uložení obvodů, ta druhá má za úkol načíst nový obvod z vybraného souboru.

3.5.1 Uložení obvodu

Pokud se uživatel rozhodl uložit vygenerovaný obvod, musí nejdříve vybrat soubor, do kterého se obvod bude ukládat. V horní záložce s názvem „File“ se nachází položka „Save circuit“. Po vyvolání signálu z této položky se zavolá slot `handleExport`, který je s ní spojený. Tento slot vytvoří novou instanci třídy `fileWork` a zavolá její veřejnou funkci `exportData`. Volaná funkce má 3 vstupní parametry. První je ukazatel na třídu, ze které se tato funkce volá. Dalším parametrem je kontejner s uloženými počty hran a součástek. Jako poslední je předávaný kontejner se všemi hranami. Tato funkce vrací svému předchůdci hodnotu, zda se uložení do souboru povedlo či nikoli.

Po zavolání této funkce je potřeba otevřít nové dialogové okno pro vybrání názvu souboru. K tomuto účelu slouží třída `QFileDialog`, která otevře dialogové okno a vrátí název vybraného souboru. Poté se otevře tento soubor jako výstupní a uloží se do něj data. Nejprve se uloží vlastnosti o formátu souboru a poté se zapíše startovní značka kořenového elementu (Kořenový element je právě jeden element, který se nevyskytuje v žádném jiném elementu a všechny ostatní elementy jsou potomky právě tohoto kořenového elementu.[16]). Na každý další řádek se zapíše informace o počtu hran a jednotlivých součástek ve tvaru: `<numOfEdges>X</numOfEdges>`, kde X je počet daných hran nebo součástek. Po vypsání všech číselných parametrů se vypíše startovací značka

`<edges>` a do souboru se uloží všechny hrany. U každé z nich se jako parametry uloží počáteční i cílový bod, hodnota dané součástky a její směr. Typ součástky je uložený mezi startovací a koncovou značkou jako obsah elementu. Po vypsání všech hran je potřeba uzavřít tento výpis koncovou značkou `</edges>` a poté i ukončit zápis celého obvodu koncovou značkou kořenového elementu `</circuit>`.

Po dokončení zápisu se ještě vytvoří stejnojmenný soubor s příponou PNG (portable network graphics) a do něj se uloží vygenerované schéma grafu k možnému zobrazení i mimo tento program. Nakonec se soubory uloží a funkce vrátí hodnotu pro úspěšné zapsání. Pokud se do volající funkce vrátí hodnota úspěchu, zavolá se třída `QMessageBox` pro informování o úspěšném uložení.

3.5.2 Načtení obvodu

Pro načtení obvodu je práce uživatele podobná jako pro jeho uložení. V záložce „File“ se nachází druhá položka „Load circuit“, která slouží pro načtení existujícího obvodu. Po stisknutí tohoto tlačítka se signál spojí se slotem `handleImport`. Podobně jako v případě uložení, tento slot také vytvoří novou instanci třídy pro správu souborů, ale poté zavolá její funkci `importData`. Tato funkce má pouze dva vstupní parametry. Ukazatel na třídu, ze které se tato funkce volá a ukazatel na kontejner pro správu všech hran. Výstupním parametrem této funkce je kontejner s uloženými počty jednotlivých součástek a hran.

Po zavolání této funkce se otevře dialog třídy `QFileDialog`, kde uživatel vybere, ze kterého souboru chce nový obvod načíst. K jednoduššímu čtení ze souborů formátu XML existuje třída `QXmlStreamReader`. Za pomoci vstupního souboru se vytvoří její instance, ze které se budou číst všechny informace. Informace se čtou do té doby, dokud tato třída nenarazí na konec souboru nebo nedojde k chybě. Poté se začnou kontrolovat jednotlivé informace. Pokud obvod obsahuje startovní značku kořenového elementu, začne tato třída číst jednotlivé elementy souboru a ukládat počty součástek do pomocného kontejneru. Po dočtení těchto informací narazí čtecí třída na startovací značku `<edges>`. To znamená, že za ní se už nacházejí pouze informace o každé hraně. Proto vždy, když čtecí třída narazí na startovní značku nového elementu, uloží veškeré jeho parametry i s jeho hodnotou do nového záznamu kontejneru. Tím se naplní tento kontejner údaji o všech hranách.

Po dokončení čtení všech informací ze souboru se tato funkce zavře s návratovým kontejnerem, který obsahuje počty všech součástek. Pokud je tento kontejner prázdný, tak čtení nebylo úspěšné a nový obvod se nevytvoří. Pokud čtení bylo úspěšné, tak slot `handleImport` aktualizuje parametry obvodu podle nových informací a vygeneruje načtený obvod.

Kapitola 4

Testování

Výslednou hotovou aplikaci bylo potřeba řádně otestovat. To bylo provedeno metodou „test case“. Tato metoda je založená na předpokladu, že se na software nahlíží jako na „black box“ (uživatel nemá žádné poznatky o implementaci a pouze ví, jak software má fungovat[17]). K tomuto testu byli vybráni tři studenti FIT ČVUT, kteří byli nejvíce podobní cílové skupině. Protože výsledný software je určený vyučujícím předmětu BI-ČAO vyučovaném právě na této fakultě, tak by měli mít znalosti o tomto předmětu i o této fakultě. Účastníkům testu bylo vysvětleno, v jakém prostředí bude tato aplikace používána a k čemu by měla sloužit. Poté dostali sadu úkolů, které měli splnit. U úkolů byly vždy sepsané očekávané výsledky aplikace. Po splnění úkolu účastník vyplnil, zda dosáhl stejného výsledku jako byl očekávaný výsledek v zadání. Pokud při testování probíhalo něco jinak, než se účastníkovi zdálo, sepsal daný problém a předal ho potom s výsledky.

Jako výsledek testování se zde uvádí pár zadaných úkolů, výsledné ohodnocení od účastníků a zhodnocení jejich odezvy.

4.1 Vytvoření obvodu a uložení

Tento úkol byl úvodním testem tohoto softwaru, pro seznámení s uživatelským prostředím a základními funkcemi. Uživatelé měli za úkol vytvořit základní obvod složený ze sedmi hran a čtyř součástek. Výsledný obvod poté měli uložit do domovského adresáře.

4.1.1 Zhodnocení účastníků

Tento úkol byl základní a proto všichni účastníci splnili tento úkol bez problémů. Žádný z nich neměl potíže s nalezením nastavení parametrů obvodu ani s jeho následným uložením. Uživatelské rozhraní bylo dostatečně intuitivní pro pochopení tohoto úkolu.

4.2 Načtení existujícího obvodu

V tomto úkolu bylo nejdůležitější načíst obvod ze souboru a poté s ním dále pracovat. Uživatelé měli načtený obvod přegenerovat a změnit vybrané součástky za jiné. Poté měli upravit hodnoty všech součástek na jimi vybrané hodnoty. Celý úkol byl završen zkopírováním časových rovnic do softwaru Mathematica.

4.2.1 Zhodnocení účastníků

Tento úkol byl také splněn všemi účastníky testování. Většina neměla problém ani s přegenerováním obvodu a změnou součástek. Jeden z účastníků měl problém se změnou typu součástky. Po vysvětlení funkce pravého tlačítka a s tím spojeného rozbalovacího menu byl problém vyřešen a účastník dokončil úkol bez sebemenších problémů. Kopírování rovnic nečinilo účastníkům také žádné problémy, vše jim připadalo na svém místě.

4.3 Komplexní práce s programem

Toto byl závěrečný úkol celého testování, který měl prověřit funkčnost celého programu a jeho základních vlastností. Uživatelé měli za úkol následující body:

1. Načíst obvod z přiloženého souboru, skládající se z 13 hran .
2. Snížit počet hran na devět a složit obvod pouze z rezistorů a cívek.
3. Uložit výsledný obvod se všemi vizuálními zobrazeními.
4. Vygenerovat nový obvod z 16 hran a změnit ho do podoby obvodu, který byl uveden v zadání.
5. Skrýt zobrazení uzlů, hodnot a směru proudů všech součástek.
6. Uložit nový obvod pod jiným názvem.

4.3.1 Zhodnocení účastníků

Úkol všichni účastníci provedli až do konce a všichni ho zvládli tak, jak bylo uvedeno v zadání. V průběhu testování nastala jedna chyba. Jeden z účastníků v bodě číslo dva snížil počet hran v panelu nastavení, ale nesnížil počet součástek, které byly vygenerované v minulém obvodu. Tím došlo k chybě, protože se snažil vygenerovat více součástek než bylo volných hran. Program zobrazil chybu, ale poslední obvod se mu smazal a on musel začít od začátku znovu. Poté už upravil i počet součástek a chybu obešel. Problém byl při odevzdávání výsledků nahlášen a zapsán.

Druhý účastník vznesl připomínku, že funkce pro skrytí různých vizuálních částí jsou nepřehledné a dostává se k nim složitě. Tato připomínka také byla přijata při odevzdání testování.

4.4 Vyhodnocení testování

Testování proběhlo v pořádku. Všichni účastníci splnili všechna zadání a dokázali vše vytvořit. V průběhu testování si nestěžovali na žádné komplikace, kromě dvou zaznamenaných problémů. Práce byla plynulá a účastníci věděli, kde danou funkcionalitu hledat.

Problém se špatným zadáním obvodu a smazáním jeho předchůdce byl zaznamenán a vyšetřen. Při krokování tohoto problému se přišlo na chybu v implementaci aplikace. Ve funkcích `handleGenerate` a `handleReGenerate`, které spravují zmíněné generování obvodu, bylo špatné pořadí vyhodnocovaných operací. Nejdříve se minulý obvod smazal, vytvořil nový a poté se teprve kontrolovalo, zda je vůbec možné počet součástek umístit na nový obvod. Po změně pořadí těchto operací, kde se nejdříve zjistí, zda je možné obvod bez problémů vygenerovat a až poté se minulý obvod smaže, se problém vyřešil a tato závada byla odstraněna.

Druhý problém nastal v nejasnosti rozmístění prvků pro nastavení zobrazení grafu. Chyba v tomto případě nastala při postupné implementaci těchto prvků. Protože nebyly původně v plánu, každý jednotlivý prvek byl postupně dodělán a byl umístěn na jiné místo. Řešením tohoto problému bylo přesunutí těchto prvků na jedno místo. V horním panelu byla vytvořena nová záložka „Window“, které se stará o nastavení zobrazení výsledného obvodu. Pouze v této záložce se vyskytují tlačítka pro zobrazení nebo skrytí vykreslovaných prvků.

Závěr

Cílem této práce bylo vytvořit automatický generátor elektrických obvodů, který dle zvoleného počtu hran a součástek vykreslí výsledný obvod, umožní jeho lehkou změnu, aniž by se celý překresloval. Také udělá export nebo import obvodu z nebo do programu. Výsledný program automaticky generuje analogové obvody, které odpovídají všem bodům zadání této práce. Je možné vždy obvod vygenerovat celý od začátku. Lze také vygenerovat jen rozložení součástek položených na daném obvodu, když by se uživateli nelíbilo jejich předchozí rozložení. Je možné vyměnit jen jednotlivé součástky za jiné, které si uživatel vybere. Při exportu obvodů se vždy uloží data do formátu XML.Ten se používá i pro opětovné načtení. K exportu se připojí soubor ve formátu PNG, ve kterém bude uložený obrázek obvodu.Ten lze použít k vložení do testů, které budou dostávat studenti.

Po otestování aplikace se v budoucnu očekává zapojení programu do výuky předmětu BI-ČAO, kde by mohl částečně vypomoci vyučujícím při tvorbě a kontrole testů a zkoušek. Tento generátor by se určitě dal využít při úplné automatizaci testů, kde by se jeho rovnice mohly porovnávat se studentskými odpověďmi. Pro tento účel by se muselo unifikovat názvosloví používané studenty v testech, aby správnou odpověď neoznačil za špatnou jen kvůli jinému označení komponentů.

Tato bakalářská práce také otevírá možnosti dalším tématům závěrečných prací, ať už zmíněnou automatizaci celého testování v předmětu BI-ČAO a nebo jen rozšíření stávajícího programu. Program se může rozšířit například o používání dalších součástek, jako jsou například transistory, nebo o jiné tvarování výsledných obvodů, než jen do stávajících obdélníků nebo čtverců.

Bibliografie

1. *draw.io* [online] [cit. 2018-05-03]. Dostupné z: https://github.com/jgraph/drawio.
2. *TINA Design Suite* [online]. DesignSoft [cit. 2018-05-03]. Dostupné z: https://www.tina.com.
3. *Qucs: Quite Universal Circuit Simulator* [online] [cit. 2018-05-03]. Dostupné z: https://github.com/jgraph/drawio.
4. *Visual Studio 2017* [online]. Microsoft Corporation [cit. 2018-05-01]. Dostupné z: https://www.visualstudio.com.
5. *Qt Documentation* [online]. Qt Company [cit. 2018-05-05]. Dostupné z: http://doc.qt.io/.
6. VEČERKA, Arnošt. Grafy a grafové algoritmy. *Katedra informatiky, přírodovědecká fakulta, Univerzita Palackého, Olomouc*. 2007.
7. KUPHALDT, Tony R. Lessons In Electric Circuits, Volume I–DC. *Vol. Fifth Edition. Open Book Project*. 2006.
8. KYNCL, Jan; NOVOTNÝ, Martin. *Číslicové a analogové obvody* [online]. České Vysoké učení technické, 2016 [cit. 2018-04-17]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-CAO/_media/textbook/cao-skripta.kindle.pdf.
9. *Qt for Beginners: Qt class hierarchy* [online]. Qt Company [cit. 2018-05-05]. Dostupné z: https://wiki.qt.io/Qt_for_Beginners.
10. *Collins English Dictionary - Complete & Unabridged 10th Edition* [online] [cit. 2018-05-04]. Dostupné z: http://www.dictionary.com/browse/wysiwyg.
11. *Reference for cstdlib* [online]. 2000 - 2017 [cit. 2018-05-05]. Dostupné z: http://www.cplusplus.com/reference/cstdlib/.

12. PASQUALONI, Anthony. *Random number generation using a 256-state cellular automaton* [online]. 2006 [cit. 2018-05-05]. Dostupné z: `http://www.pub22.net/ca-rng/report.html`. School of Graduate Studies at Southern Connecticut State University.
13. WOLFRAM, Stephen. *Cellular automata and complexity: collected papers*. CRC Press, 2018.
14. *Mathematica* [online]. Wolfram Research [cit. 2018-05-07]. Dostupné z: `www.wolfram.com/mathematica/`.
15. MAREŠ, Martin; VALLA, Tomáš. In: *Průvodce labyrintem algoritmů*. CZ. NIC, zspo, 2017, s. 107.
16. *Extensible Markup Language* [online]. 2. vyd. World Wide Web Consortium, 2006 [cit. 2018-05-11]. Dostupné z: `https://www.w3.org/TR/xml11/`.
17. PATTON, Ron. *Software Testing*. 2. vyd. Sams Publishing, 2005.

Příloha **A**

Seznam použitých zkratek

BI/E-ČAO Bakalářský předmět (vyučovaný v češtině a angličtině) Číslicové a analogové obvody

FIT ČVUT Fakulta Informačních Technologií Českého vysokého učení technického v Praze

PNG Portable network graphics

Qucs Quite Universal Circuit Simulator

TINA Toolkit for Interactive Network Analysis

WYSIWIG What You See Is What You Get

XML Extensible markup language

Příloha B

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `exe` ..... adresář se všemi knihovnami pro spuštění programu
  - `circuit-generator` ..... adresář se spustitelným souborem
  - `examples` ..... adresář s příkladnými obvody
    - `circuit1.xml`
    - `circuit2.xml`
- `src`
  - `implementation` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce ve formátu PDF
  - `brandsim-BP.pdf`