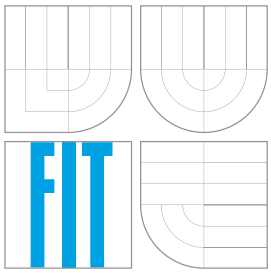
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

GRAFICKÉ ROZHRANÍ PRO MANIPULACI S CHROMOZOMY GENETICKÉHO PROGRAMOVÁNÍ V JAVĚ

GUI FOR HANDLING GENETIC PROGRAMMING CHROMOSOME

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JANA STAUROVSKÁ
AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ JAROŠ
SUPERVISOR

BRNO 2010

Abstrakt

Cílem této práce je vytvořit program pro manipulaci s chromozomy genetického programování, který by měl umožňovat export do vektorového formátu, posouvání hradel, jejich zabarvení a další grafické operace, který funguje na různých operačních systémech (hlavně Microsoft Windows a Linux). Pro lepší pochopení problematiky je v teoretické části popsán základní princip kartézského genetického programování.

Abstract

The main goal of this thesis is to create a program for manipulation with genetic programming chromosomes, which should allow export to a vector graphics format, moving of gates, their colouring and other graphical operations, and will work on different operating systems (mainly Microsoft Windows and Linux). For better understanding, the basic principles of cartesian genetic programming are described in theoretical part.

Klíčová slova

genetika, genetické programování, kartézské genetické programování, hradlo, chromozom, grafické uživatelské rozhraní

Keywords

genetics, genetic programming, cartesian genetic programming, gate, chromosome, graphical user interface

Citace

Jana Staurovská: Grafické rozhraní pro manipulaci s chromozomy genetického programování v Javě, bakalářská práce, Brno, FIT VUT v Brně, 2010

Grafické rozhraní pro manipulaci s chromozomy genetického programování v Javě

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Jiřího Jaroše. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Jana Staurovská
14. května 2010

Poděkování

Chcela by som poďakovať Jiřímu Jarošovi, Ing. za ochotu, trpezlivosť a odborné vedenie bakalárskej práce a môjmu priateľovi Františkovi Skálovi za neustálu podporu a užívateľský pohľad na tento program.

Obsah

Kapitola 1

Úvod

V súčasnej dobe sa genetické programovanie (ďalej iba GP) čoraz viac rozširuje. Je veľmi efektívne napríklad pre automatický návrh jednoduchých kombinačných obvodov a tiež pre riešenie rôznych optimalizačných a ďalších problémov. Vychádza z toho, čo môžeme vidieť v živej prírode — evolúcie. Efektivita GP je nižšia pri návrhu zložitejších obvodov kvôli vyššej časovej náročnosti a tak nemôže byť použité priamo, ale je nutné použiť nejakú z ďalších techník, napr. inkrementálnu evolúciu, evolúciu na úrovni funkčných blokov alebo začleniť embryonálnu fázu do evolučného algoritmu.[8]

Evolučný prístup poskytuje netradičný pohľad na riešenie širokého spektra problémov. Na rozdiel od konvenčných prístupov je schopný využiť aj také vlastnosti prehľadávaného priestoru, ktoré ostávajú ľudskému návrhárovi skryté. Konvenčné metódy návrhu fungujú na rôznych matematických modeloch a tak sú ich výsledky vždy rovnaké a fungujú na rovnakých princípoch. V prípade evolučného prístupu nejakým spôsobom (napr. pravdivostnou tabuľkou) špecifikujeme, ako má daný obvod fungovať, zvolíme rekonfigurovateľný obvod, v ktorom sa má riešenie hľadať a evolučná metóda ho nájde. Tento prístup často produkuje lepšie výsledky než klasické konvenčné metódy čo sa týka množstva použitých hradiel alebo oneskorenia obvodu. Z toho sa dá usúdiť, že poskytuje riešenia, ku ktorým sa klasickými metódami nedopracujeme a teda je množina riešení väčšia.

Cieľom tejto práce je vytvoriť grafické rozhranie pre všetkých, ktorí využívajú CGP (Cartesian Genetic Programming) na zakódovanie obvodu do chromozómu. Výsledný program by mal dokázať zobraziť chromozóm ako kombinačný obvod, mal by dokázať farebne odlíšiť rôzne typy hradiel a pohybovať nimi. Taktiež by mal zvládnuť priblížiť a oddialiť zobrazený chromozóm pre väčšiu prehľadnosť a prípadný výrez byť schopný uložiť buď do vlastného formátu pre následné pokračovanie v práci s ním, alebo do vektorového grafického formátu pre pohodlné prehliadanie. To isté by samozrejme mal zvládnuť aj s celým chromozómom.

Prvou kapitolou je tento úvod. V druhej kapitole bude popísané čo to kartézske genetické programovanie je, ako vyzerá chromozóm a na čo sa používa. V kapitole tretej bude vysvetlená vektorová grafika a jej formáty. Keďže táto práca vyžaduje aj určité grafické úpravy, tieto budú popísané v ďalšej kapitole. Pred samotným programovaním je potrebné urobiť analýzu zadania a návrh riešenia, čo je obsahom piatej kapitoly. Ako vyzerá a funguje výsledný program je popísané v predposlednej kapitole. Poslednou kapitolou je záver, kde sú zhrnuté dosiahnuté výsledky, prínos práce a nápady do budúcna.

Kapitola 2

Kartézske genetické programovanie

CGP predstavil najprv J. Miller vo svojom článku Cartesian Genetic Programming [7] v roku 1999 a potom spoločne s P. Thomsonom v roku 2000 [8]. CGP sa od klasického GP líši reprezentáciou problému, používa totiž graf s pevným počtom uzlov (GP používa strom), ktorý sa môže nazývať maticou. V nej nie je povolená spätná väzba. Z toho vyplýva, že teoreticky môžeme napojiť akýkoľvek vstup na akýkoľvek výstup bez príliš veľkej zložitosti (nemusí sa vytvárať nový podstrom). Prakticky sa táto charakteristika obmedzuje pomocou l-back parametru. Tento parameter musí byť z intervalu $\langle 1, m\rangle$ a určuje o koľko stĺpcov vpred môžeme pripojiť výstup na ďalší vstup. Ak je tento parameter rovný jednej, tak výstupy z jedného stĺpca môžu byť pripojené len na vstupy stĺpca hneď vedľa neho. Takýto obvod je potom jednoduché previesť do zreťazeného tvaru (pipeline). Naopak, ak je tento parameter rovný počtu stĺpcov, tj. maximu, tak zaisťuje maximálnu konektivitu a tým slúži na hľadanie veľmi komplexných obvodov.

2.1 Kódovanie prepojenia elementov (chromozóm)

Zakódované kandidátne riešenia problému sú v našom prípade chromozómy. Jeden chromozóm tvorí množina $m \times n \times$ *(počet vstupov elementu + 1) + počet výstupov celého obvodu* k-tic celočíselných hodnôt. Pre jednoduchú orientáciu majú všetky výstupy elementov a vstupy kombinačného obvodu svoje čísla. Čísluje sa po stĺpcoch zľava doprava — najprv vstupy kombinačného obvodu, potom výstupy jednotlivých elementov, až po výstupy kombinačného obvodu.

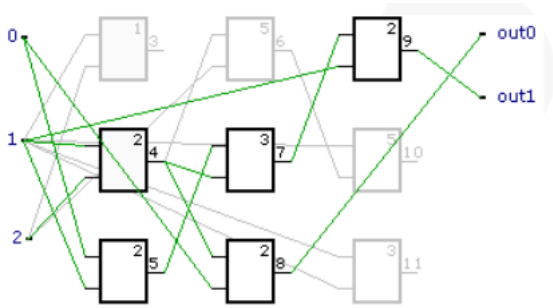
Hodnoty v každej k-tici udávajú čísla výstupov privedených na vstupy konkrétneho elementu, posledné číslo špecifikuje logickú funkciu, ktorú element realizuje (napr. AND, OR, XOR...). Na konci chromozómu je ešte jedna k-tica — pre každý výstup kombinačného obvodu určuje, výstup ktorého elementu je naňho napojený.

Príklad 2.1.1. Ukážkový chromozóm 3×3 (prevzatý z [8]):
(1,2,1)(1,2,2)(0,1,2)(4,2,5)(5,4,3)(4,0,2)(7,1,2)(1,6,5)(1,1,3)(8,9)

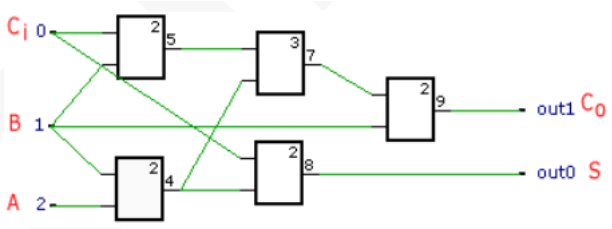
V príklade 2.1.1 je zápis ukážkového chromozómu, ktorý tvoria elementy s dvoma vstupmi a s dvoma celkovými výstupmi kombinačného obvodu. Tento chromozóm je tvorený maticou 3×3, má tri vstupy kombinačného obvodu a teda index výstupu prvého elementu je 3. Elementy, ktoré sa nepodieľajú na riešení sú vykreslené šedou farbou na obrázku 2.1. Sú to elementy, ktorých výstupy sa nepripájajú na iné vstupy.

Na obrázku 2.2 je rovnaké prepojenie, avšak hradlá sú inak umiestnené — sú vynechané tie, ktoré boli šedo vykreslené v predchádzajúcom obrázku a sú dodané významy vstupov a

výstupov. Tu už je vidieť, že ide o štandardné zapojenie jednobitovej úplnej sčítačky, ktorá využíva štyri hradlá typu XOR (funkcia 2) a jedno hradlo typu AND (funkcia 3).



Obrázek 2.1: Dekódovaný chromozóm [8]



Obrázek 2.2: Prekreslené schéma zapojenia [8]

2.2 Evolučný algoritmus

Algoritmus CGP je založený na evolučnej stratégii (ES), konkrétne na variante $(1 + \lambda)$. Pri tomto spôsobe tvorí novú populáciu λ novovzniknutých mutantov najlepšieho rodiča aj ich rodič. Veľkosť populácie je teda $\lambda + 1$ a na základe skúseností sa λ volí okolo 4 [8]. Ako operátor CGP sa používa iba mutácia, ktorá pracuje tak, že náhodne zmení hodnotu génu na inú. Samozrejme s ohľadom na to, aké hodnoty sú v tomto géne prípustné. Tento operátor je riadený parametrom, ktorý udáva percento mutovaných génov, v našom prípade je jeden gén jedna celočíselná hodnota.

Celý algoritmus sa dá popísať pomocou niekoľkých krokov [7] :

1. Náhodné vygenerovanie inicializačnej populácie
2. Ohodnotenie všetkých jedincov populácie pomocou fitness funkcie (viď. kapitola 2.3
3. Výber najlepšie ohodnoteného jedinca do novej populácie
4. Vygenerovanie λ potomkov mutácií nájdeného najlepšieho jedinca
5. Pokiaľ nie je splnená podmienka pre ukončenie, pokračuje sa krokom 2

Tretí krok je kritický. Je potrebné vybrať vždy rôzneho jedinca od toho pôvodného, aby nedochádzalo k degradácii a spomaleniu evolúcie, pretože by v populácii nebol prítomný žiaden nový genetický materiál.

2.3 Fitness funkcia

Vzhľadom na to, že počas evolúcie je potrebné určiť ako dobré je určité zapojenie (chromozóm), je potrebná funkcia, ktorá dokáže spravodlivo ohodnotiť vzniknutých jedincov. Táto funkcia sa volá **fitness funkcia**.

Výpočet fitness hodnoty sa robí postupným nastavovaním všetkých možných vstupných kombinácií podľa pravdivostnej tabuľky. Pre tieto kombinácie sa potom podľa zapojenia vypočítajú výstupy, tj. simuluje sa činnosť kombinačného obvodu a porovnávajú sa očakávané výsledky so simulovanými. V prípade úplnej definície sa jedná o 2^n kombinácií, kde n je počet vstupov [8].

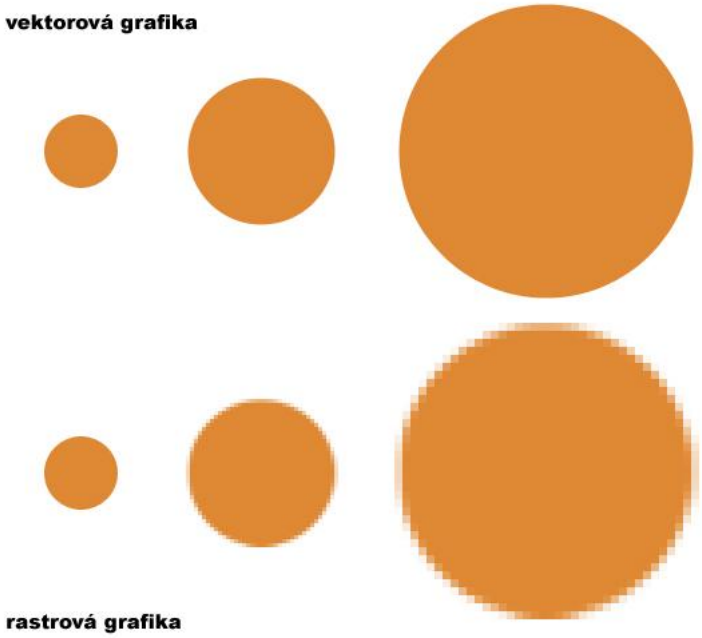
Hodnota fitness je potom rovná počtu zhôd simulovaných výsledkov s výsledkami očakávanými. Ak máme napríklad 4 vstupy a 2 výstupy, tak počet vstupných kombinácií je $2^4 = 16$. Maximálny fitness teda môže byť $16 * 2 = 32$ zhôd [8].

Toto samozrejme nie je jediná možnosť fitness funkcie. Môže sa porovnávať počet použitých elementov, spotreba, oneskorenie a pod.

Kapitola 3

Vektorová grafika

Existujú dva spôsoby vykresľovania objektov: rastrovo a vektorovo. Základom rastrového vykresľovania (tj. rastrovej grafiky) je pixel — jeden bod na obrazovke. Pomocou pixelov sa vyskladá celý obraz. Oproti tomu vektorová grafika je založená na matematickom popise jednotlivých objektov — úsečka, kruh, obdĺžnik a pod. Každý tento objekt môže mať rôzne vlastnosti — veľkosť, obrys, výplň a pod. Vektorový obraz je vo svojej podstate tvorený vrstvami, pričom v každej vrstve je jeden objekt, čo umožňuje jednoduchú manipuláciu s časťami obrazu (tj. objektami). Hlavnou výhodou vektorovej grafiky je možnosť neobmedzene zväčšovať výsledný obraz bez straty kvality, čo je veľmi výhodné pri rôznych logách, plagátoch a pod. Na obrázku je názorná ukážka zväčšovania objektu v rastrovej a vo vektorovej grafike.



Obrázek 3.1: Rozdiel medzi vektorovou a rastrovou grafikou pri zväčšovaní obrázka. [3]

Pre túto prácu je vektorová grafika veľmi výhodná tým, že obraz v nej je možné zväščovať v podstate do nekonečna, čo je pri veľkých chromozómoch veľmi užitočné. Zobrazovanie je

síce rastrové, pretože samotná obrazovka počítača sa skladá z konkrétnych bodov, ale pri ukladaní je reprezentácia vektorová, takže si zachováva všetky detaily.

3.1 Vektorové formáty

Formáty vektorovej grafiky nemusia obsahovať iba matematicky popísané objekty, môžu obsahovať aj rastrové dáta, stratí sa tým však ich primárna výhoda. Raster sa využíva aj pri záverečnom vykreslení na digitálne obrazovky, ktoré sa skladajú z bodov (pixelov). Formátov, do ktorých sa dajú vektorové dáta uložiť, je viacero, avšak najpoužívanejší je **SVG (Scalable Vector Graphics)**. SVG je značkovací jazyk z rodiny XML (eXtensible Markup Language,) ktorý je určený na opis dvojrozmernej, statickej alebo animovanej vektorovej grafiky. [2]. SVG je otvorený štandard vytvorený konzorciom World Wide Web[1]. Jeho prednosťou je práve to, že je značkovacím jazykom — je veľmi jednoduché upravovať súbory, vyhľadávať v nich, dokáže definovať statický obrázok tak ako aj animáciu, dajú sa definovať akcie napríklad pri pohybe myšou a podobne, čím si tento formát nájde využitie napríklad na webe. Výhodou je určite aj existencia mobilných profilov SVG1.1: SVG Tiny (SVGT) a SVG Basic (SVGB) pre user agentov s obmedzenými možnosťami zobrazovania, napr. mobilné telefóny, PDA a podobne. Tieto profily sú plne kompatibilné podmnožiny plného štandardu [2].

Ďalším bežne podporovaným formátom je **CGM (Computer Graphics Metafile)**. Je to formát pre 2D vektorovú a rastrovú grafiku a text. Je definovaný pomocou ISO/IEC 8632. Všetky grafické elementy sú definované v textovom súbore, ktorý je možné skompilovať do binárnej podoby alebo do jednej z dvoch textových reprezentácií. Pri rekonštrukcii obrázku z dát využíva objektový prístup. Nie je síce veľmi používaný, aj keď stále vie nájsť svoje využitie. W3C (World Wide Web Consorcium) vytvorilo WebCGM pre CGM použiteľné v internetových stránkach.

Iné formáty sú využívané oveľa menej, prípadne majú len špeciálne použitie. Sú to napríklad: ODG (OpenDocument Graphics, otvorený formát), EPS (Encapsulated PostScript), PDF (Portable Document Format), SWF (shockwave Flash), WMF / EMF (Windows Metafile / Enhanced Metafile) alebo XPS (XML Paper Specification).

[1] Viac na http://www.w3c.org/

Kapitola 4

Grafické úpravy

Nie je v možnostiach tejto práce napísať všetky grafické úpravy, ktoré existujú a preto je táto kapitola zameraná iba na tie, ktoré sú relevantné pre túto prácu. Je to metóda antialiasing a využitie transformačnej matice.

4.1 Antialiasing

Aby bolo možné vysvetliť antialiasing, je potrebné vysvetliť, čo je to alias. Najčastejšie je to nechcený detail, artefakt v obraze. Vzniká nízkofrekvenčným vzorkovaním signálu s relatívne vysokou frekvenciou. Môžeme ho tiež definovať ako presakovanie (zobrazenie) vysokých frekvencií do nízkych v dôsledku podvzorkovania [5].

Za aliasing sú často nesprávne považované zubaté hrany (jagged edges, alebo skrátene jaggies). Jedná sa o detail vnesený do obrazu spôsobom vzorkovania. Zubaté hrany vznikajú v obraze na miestach s vysoko kontrastnými hranami [5].

Aliasing sa prejavuje v grafike vo forme javu moiré, čo je rušivý efekt vyskytujúci sa v obrazoch obsahujúcich nejaký pravidelný vzor o vysokej frekvencii. Ak budeme tento obraz podvzorkovávať, môže dôjsť k interferencii medzi touto frekvenciou a vzorkovacou frekvenciou. Výsledkom sú rôzne nepravidelné obrazce v obraze, viď obrázok 4.1.

Obrázek 4.1: Príklad moiré. [1]

Už z názvu antialiasing vyplýva, že je to obecne nejaká metóda zaoberajúca sa potlačovaním javov spôsobených aliasingom. Týchto metód je mnoho, napr. supersampling,

multisampling alebo filter typu dolná priepustnosť. Pre túto prácu je dôležité, že antialiasing vyhladzuje hrany — v prípade rasterizácie vektorového obrazu teda všetko.

4.2 Transformácie

Pre túto prácu je dôležitá iba transformačná matica pre zmenu zoomu (merítka, zväčšenia). Pre lepšie pochopenie je však potrebné poznať základ geometrických transformácií. Pri príprave tejto kapitoly bola použitá [6].

4.2.1 Geometrická transformácia

Geometrickú transformáciu môžeme chápať ako zmenu pozície vrcholov v momentálnom súradnicovom systéme alebo ako zmenu súradnicového systému. Sú to hlavne operácie: posunutie, otočenie, zmenšenie/zväčšenie, skosenie. Všetky tieto základné geometrické transformácie sú lineárne. To znamená, že zobrazenie z jedného vektorového priestoru do druhého zachováva lineárne kombinácie, tj. všetky body pôvodného priestoru sa vynásobia vektorom transformácie, okrem posunutia, ktoré využíva súčet. Vzhľadom na túto skutočnosť nie je možné pracovať so všetkými transformáciami jednotne. Preto je snaha previesť všetky transformácie na jednotnú formu. Riešením sú homogénne súradnice. Umožňujú nám vyjadriť všetky druhy základných transformácií jednou transformačnou maticou a využiť tak násobenie matíc a vektorov (aj pri posunutí, takže nie je potrebný súčet).

Definícia 4.2.1. Homogénne súradnice bodu v 3D s kartézskymi súradnicami $[x, y, z]$ je usporiadaná štvorica $[X, Y, Z, w]$ pre ktorú platí $x = X/w, y = Y/w, z = Z/w$. Bod je svojimi homogénnymi súradnicami určený jednoznačne. Súradnicu w nazývame váhou bodu. Hodnota váhy je väčšinou $w = 1$, v prípade lineárnych transformácií.

4.2.2 Transformačná matica pre zmenu zoomu

Zmena zoomu (merítka) bodu v rovine s homogénnymi súradnicami $P(x, y, 1)$ a faktormi zmeny merítka S_x a S_y je daná vzťahom 4.1. Zápis násobenia matíc je vo vzťahoch 4.2 (skrátene) a 4.3 (rozvedene) . Tvar samotnej transformačnej matice je vo vzťahu 4.4. Bod P' je pôvodný bod po transformácii.

Ak je faktor zmeny merítka $S_{x,y} > 1$, dochádza k zväčšeniu. Ak je $0 < S_{x,y} < 1$, dochádza k zmenšeniu. Ak je $S_{x,y} < 0$, dochádza k prevráteniu.

$$x' = x \cdot S_x, \quad y' = y \cdot S_y \tag{4.1}$$

$$P' = P \cdot S \tag{4.2}$$

$$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

Kapitola 5

Analýza a návrh riešenia

Na úvod tejto kapitoly si ujasníme požiadavky na výsledný program a porovnáme ich s funkciami už existujúceho programu cgpviewer[1], ktorý je však len pre OS[2] Windows. Prvou požiadavkou je teda prenositeľnosť aj na iné OS (napr. distribúcie Linuxu). Táto požiadavka je riešiteľná použitím vhodného programovacieho jazyka a používaním takých konštrukcií, ktoré pracujú rovnako na viacerých OS — vyžadovalo by to však kompilovať program vždy pre viacero OS alebo sprístupniť kód, čo nie je veľmi užívateľsky priateľské riešenie. Vybrala som si možnosť použiť programovací jazyk, ktorý bez úprav funguje na väčšine OS, konkrétne teda programovací jazyk Java. Kód v Jave sa prekladá do medzikódu, ktorý potom spracuje Java Virtual Machine (JVM) a vykoná ho. JVM je špecifická pre každý OS a práve ona je prispôsobená fungovaniu konkrétneho OS. V súčasnosti už má väčšina užívateľov JVM nainštalovanú, prípadne nie je problém ju doinštalovať. Vďaka tomuto postupu je Java kód ľahko prenositeľný.

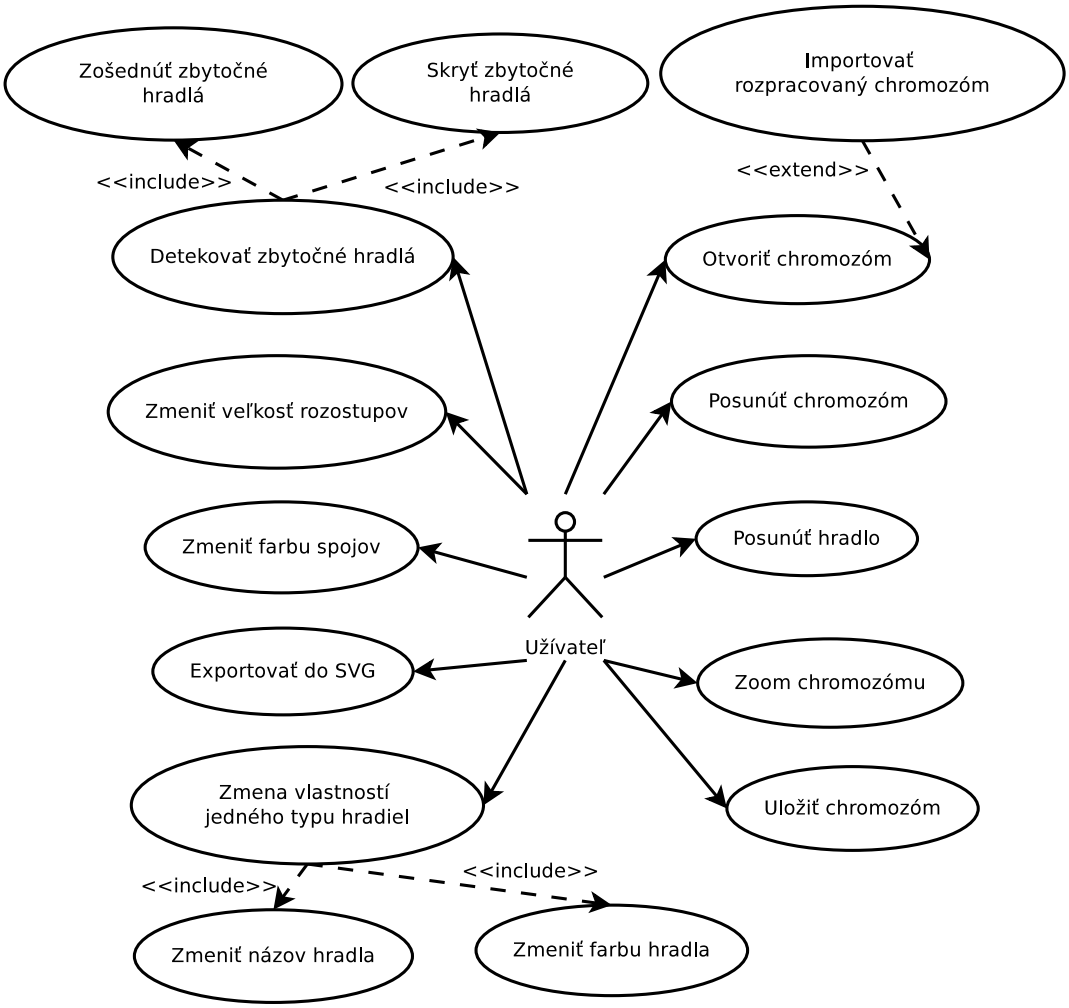
Ďalšie požiadavky sa týkajú vlastného fungovania programu. Pri zobrazovaní chromozómu by malo byť možné zafarbiť jednotlivé hradlá podľa ich typu kvôli prehľadnosti. Hradlá, ktoré sa na výsledku nepodieľajú by nemali byť výrazné, ideálne je na nich použiť šedú (a teda nevýraznú) farbu. Mala by byť tiež možnosť nepotrebné hradlá nezobraziť. Aby mal užívateľ dostatočný prehľad, je vhodné aby sa dali hradlá aj spoje presúvať tak, aby sa spoje príliš nekrížili, bolo jasnejšie ako bude konkrétny obvod fungovať a podobne. Pri veľkých chromozómoch je vhodné použiť zoom (lupu), tj. mať možnosť si daný chromozóm zväčšiť, zamerať sa iba na jeho určitú časť. Aby sa užívateľove zmeny nestratili, je potrebné ich uložiť. Buď do nejakého vnútorného formátu, ktorý bude obsahovať popis všetkých hradiel, spojov medzi nimi, farieb, prípadné posunutie hradiel a pod. Ďalšou možnosťou je ukladanie do vektorového formátu ako ľubovolne zväčšiteľný obrázok. Do vektorového formátu by sa mal dať uložiť nielen celý chromozóm, ale aj jeho výrez.

Z týchto požiadavok som vytvorila diagram prípadov užitia, viď. obrázok 5.1.

Pre porovnanie bude popísaný už existujúci program cgpviewer (verzia 1.5 z roku 2008). Má možnosť skryť nefunkčné bloky, čo v tomto prípade znamená, že zošednú. Má však aj možnosť automatického rozmiestnenia, ktoré všetky prebytočné bloky schová a zobrazí len tie, ktoré sa podieľajú na výsledku. Je to optimalizácia kvoli prehľadnosti. Spoje medzi hradlami sa správajú adekvátne k súčasnému zobrazeniu, tj. môžu tiež zošednúť alebo sa schovať. Jednotlivým blokom je možné priradiť funkcie (napr. AND, NOR, XOR) a vďaka tomu ich potom nahradiť schématickou značkou vybraného hradla. Čo sa týka možností

[1] Na stiahnutie tu: http://www.fit.vutbr.cz/~vasicek/cgp/bin/viewer.zip

[2] operačný systém



Obrázek 5.1: Diagram prípadov užitia, vytvorený programom Dia.

exportu, tento program dokáže vygenerovať VHDL a DOT, uložiť obrázok do formátu BMP a počas práce s programom je schopný si zapamätať jedno rozmiestnenie hradiel (napr. ak chcete vyskúšať nejaké iné zobrazenie a radi by ste sa potom prípadne vrátili k pôvodnému).

5.1 Formát chromozómu (výstup z programu cgp)

Tento program som si vybrala pre vytváranie chromozómov kvôli informáciám, ktoré výsledný súbor obsahuje. Tieto informácie zjednodušia niektoré konštrukcie, pretože sa z nich dá napríklad zistiť veľkosť chromozómu a podobne. Tiež z toho však vyplýva, že dosť podstatnou časťou výsledného programu bude parsovanie textu na zistenie potrebných informácií.

Program cgp využíva evolúciu na generovanie chromozómov. Výstupom tohoto programu je súbor s dvoma časťami, viď príklad 5.1.1. Riadky začínajúce sa mriežkou (#) tvoria prvú časť a riadok bez nej, v ktorom je samotný chromozóm, tvorí druhú časť. V prvom riadku prvej časti je popis chromozómu. V druhom riadku sú názvy vstupov a v treťom názvy výstupov. Jednotlivé prvky v druhej časti sú uzavrené v jednoduchých zátvorkách, výnimku tvorí prvý prvok tohto riadka, ktorý je uzavrený v zložených zátvorkách {}. Nachá-

dzajú sa v ňom informácie o celom chromozóme a to v poradí uvedenom v tomto zozname:

1. počet vstupov obvodu
2. počet výstupov obvodu
3. počet stĺpcov
4. počet riadkov
5. počet vstupov hradla
6. l-back parameter
7. počet použitých blokov.

Tieto informácie sú veľmi užitočné pre zobrazenie chromozómu. Ostatné prvky v tomto riadku popisujú jednotlivé elementy — v hranatých zátvorkách je uvedené číslo výstupu tohto elementu, za ním sú uvedené čísla výstupov elementov, ktoré sú privedené na vstupy tohto elementu a posledné je číselné označenie funkcie elementu. Posledným prvkom je zoznam hradiel, ktoré sú pripojené k výstupom. Tieto hradlá sú od seba oddelené čiarkami.

Príklad 5.1.1. Ukážka chromozómu:
median z 5b 5 vstupy, 1 vystup
#%i i4,i3,i2,i1,i0
#%o out
{5, 1, 5, 5, 2, 1, 11} ([5]3,4,0) ([6]3,4,2) ([7]2,4,0) ([8]4,3,1) ([9]0,2,2) ([10]3,9,2) ([11]4,3,1) ([12]2,0,1) ([13]6,9,1) ([14]3,1,2) ([15]1,13,1)([16]13,1,2) ([17]3,3,3) ([18]2,2,1) ([19]12,11,2) ([20]16,1,0) ([21]16,0,0) ([22]2,17,2) ([23]15,19,1) ([24]15,19,2) ([25]24,21,0) ([26]24,4,1) ([27]21,24,1) ([28]0,0,2) ([29]2,24,1) (27)

5.2 Vnútorná reprezentácia chromozómu

Java je objektovo orientovaný programovací jazyk, z čoho vyplýva, že celý program bude založený na tomto prístupe. Chromozóm bude reprezentovaný jedným z objektov. Základom chromozómu bude matica o veľkosti $m \times n$ (m — stĺpce, n — riadky). Jednotlivé prvky bude tvoriť vnútorná reprezentácia elementu, tj. štruktúra. Jej obsahom budú 3 položky: čísla vstupov, číslo výstupu, číslo funkcie. Pomocou tejto štruktúry sa správne vykreslí chromozóm. Nápomocné budú samozrejme údaje o veľkosti chromozómu, o počte vstupov obvodu, o počte výstupov obvodu, názvy vstupov a výstupov obvodu.

5.3 Export do vektorového grafického formátu

V požiadavkách bol export do vektorového grafického formátu. Snáď najrozšírenejším formátom je SVG (Scalable Vector Graphics) a tak bude tento program exportovať práve do tohto formátu. Na internete sa dá nájsť zopár knižníc, ktoré slúžia na prácu s formátom SVG. Jednou z nich je aj Batik SVG Toolkit [3], ktorý som sa rozhodla vo svojej práci použiť. Je veľmi jednoduchý na použitie a využíva výhody formátu SVG — vytvára štruktúru DOM (viac v [4]) založenú na XML popise, čo umožňuje veľmi ľahko upravovať jednotlivé časti

[3] http://xmlgraphics.apache.org/batik/

obrazu a zároveň je DOM štruktúra ľahko uložiteľná do súboru. Je možné „kresliť“ (vpisovať grafické objekty) priamo do DOM-u a potom s nimi pracovať. To umožňuje príjemnú a jednotnú manipuláciu s celým obrazom i s jeho časťami.

5.4 Zobrazenie jednotlivých hradiel a spojov

Kvôli názornosti a prehľadnosti som sa rozhodla zobrazovať hradlá ako jednoduché obdĺžniky, ktorých funkciu budú naznačovať iba dovnútra vpísané čísla alebo symboly. Typu hradla bude možné priradiť jeho logickú funkciu. Podľa toho bude možné jednotlivé hradlá zafarbiť a tak ich rozlišovať. Aby to však bolo užívateľsky prívetivé a zároveň dostatočne upraviteľné, niektoré základné farby budú prednastavené, avšak bude možnosť ich zmeniť — napríklad prepísaním hodnoty alebo výberom zo vzorkovníka. Spoje medzi hradlami budú obyčajné úsečky, ktorým bude možné meniť farbu — iba celkovo, nie jednotlivo, pričom simulácia tvorí výnimku.

5.5 Rozloženie prvkov v grafickom rozhraní

Pre užívateľa výsledného programu je dôležitá prehľadnosť, takže riadiace prvky musia byť nenápadné, avšak umiestnené intuitívne. Možností je na to samozrejme viacero, s prihliadnutím na cgpviewer by bolo vhodné ich umiestniť do spodnej časti pod kresliacu plochu a zoskupiť ich podľa funkcie. Podľa môjho názoru však zaberajú príliš veľa miesta a väčšina užívateľov je zvyknutá ovládať program pomocou menu umiestneného tesne pod dekoráciou okna (tj. nad kresliacou plochou), takže ovládacie prvky budú prirodzenejšie tam. Kresliaca plocha by mala byť dostatočne veľká, aj keď je potrebné počítať aj s užívateľmi s menšími monitormi. Preto by sa veľkosť hradiel mala prispôsobovať veľkosti okna.

5.6 Posúvanie jednotlivých hradiel

Posúvanie, inak zvané drag'n'drop, je veľmi dôležitou časťou programu. Umožňuje užívateľovi usporiadať hradlá na ploche programu podľa jeho potrieb. Z užívateľovho pohľadu by to malo fungovať na princípe „chytím hradlo a posuniem“. „Chytenie“ hradla (alebo akéhokoľvek objektu) obvykle prebieha tak, že užívateľ klikne niekam dovnútra. Je preto potrebné zisťovať kam užívateľ klikol. Takisto je potrebné vedieť, kde uvoľnil kliknutie. Rozdiel medzi týmito dvoma súradnicami je posun a podľa neho sa prepočítajú súradnice hradla.

5.7 Zoom

Približovanie a oddaľovanie obrazu sa dá robiť rôznymi spôsobmi. Dá sa to urobiť ručným prepočítavaním súradníc alebo pomocou transformačnej matice. Ručné prepočítavanie by bolo zbytočne komplikované, kód by bol neprehľadný a ľahko by sa vyskytla chyba. Preto je najvhodnejšie použiť transformačnú maticu, ktorá bola popísaná v predchádzajúcej časti tejto práce.

5.8 Simulácia zobrazovaného chromozómu

Simulácia by v tomto prípade mala fungovať na základe užívateľovho zadania hodnôt, ktoré majú byť na vstupoch (tj. kde bude nula a kde jednička), a program potom pre každé hradlo vypočíta jeho hodnotu na výstupe. Takýmto spôsobom sa program dostane až k výstupom obvodu, ktoré tiež nastaví na hodnotu, ktorú uvidí užívateľ. Je to dobré pre kontrolu obvodu a tiež pre riešenie jednoduchších úloh.

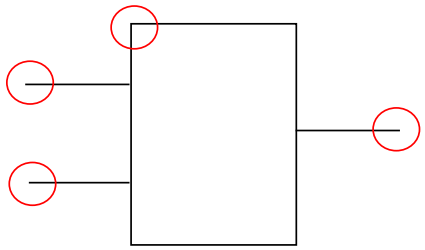
Kapitola 6

Implementácia

V tejto časti budú popísané jednotlivé časti implementácie. Program bol implementovaný v Jave, ako je spomínané vyššie. Java okrem prenositeľnosti poskytuje mnoho nástrojov na tvorbu grafických rozhraní, ktoré sú jednoduché na použitie a spĺňajú požiadavky kladené touto prácou. Samotná platforma Java však neobsahuje možnosť exportovať obrazové dáta do formátu SVG. Túto funkciu však poskytuje knižnica Batik SVG Toolkit, ktorá je priložená na CD so samotným programom.

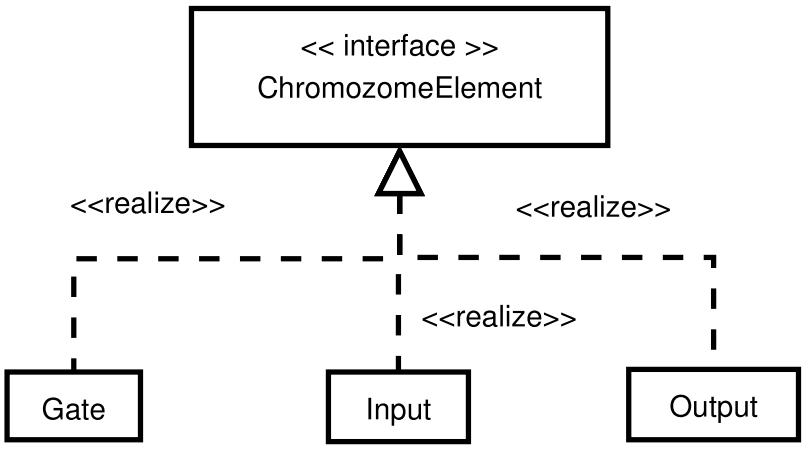
6.1 Chromozóm

Najdôležitejšou časťou tejto práce je chromozóm. S touto filozofiou som pristúpila k implementácii. Po uvedomení si, že matica $m \times n$ nie je vhodná, ak vopred nevieme jej veľkost, som ako základný objekt vybrala zoznam. V tomto zozname sa nachádzajú jednotlivé hradlá, ktoré sú taktiež samostatnými objektami. Každé hradlo obsahuje informácie o sebe samom — svoju funkciu, vstupy, výstup a pre ďalšie použitie aj súradnice ľavého horného rohu a koncových bodov vstupov a výstupu, ktoré si hradlo pri posúvaní dynamicky prepočítava.



Obrázek 6.1: Zakrúžkované sú ukladané súradnice.

Ďalšími objektami sú vstupy a výstupy. Tie majú vždy svoje poradové číslo, svoj názov a takisto súradnice. Vstupy tvoria vlastný zoznam, rovnako ako aj výstupy. Pre prehľadnosť pri niektorých operáciach je k dispozícii aj zoznam všetkých elementov (tj. vstupov, výstupov aj hradiel), ktorý obsahuje odkazy na objekty elementov (je to možné vďaka spoločnému rozhraniu, viď. obrázok 6.2). Všetky tieto objekty sa nachádzajú v objekte chromozóm. Ten sa vytvorí po načítaní súboru s textovou reprezentáciou chromozómu.



Obrázek 6.2: Diagram tried pre triedy vstupu obvodu, hradla a výstupu obvodu.

6.2 Vykresľovanie

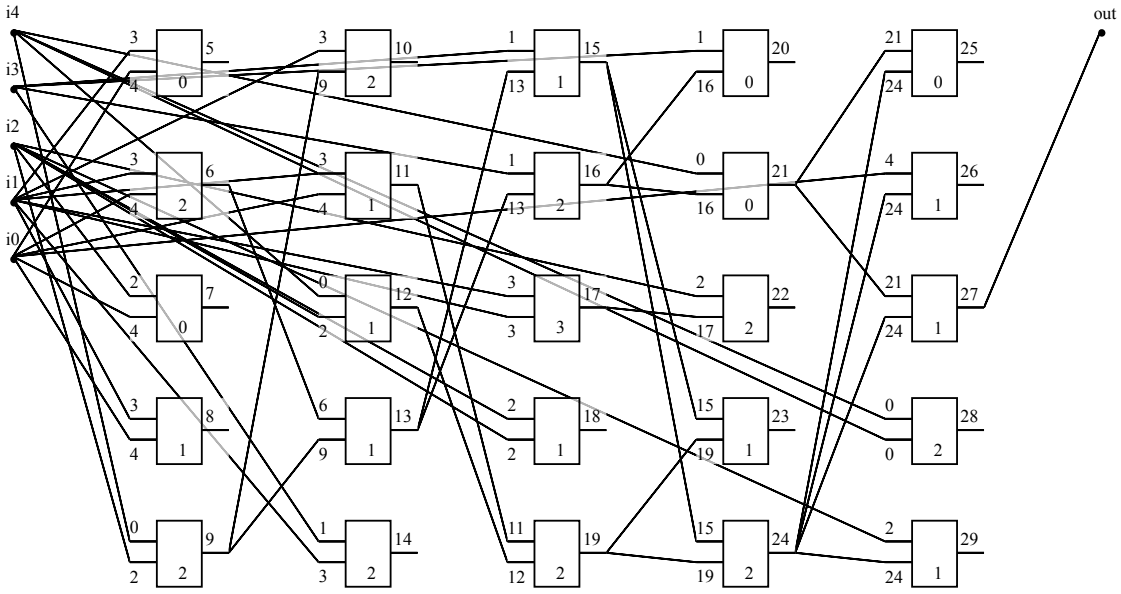
Keď existuje objekt chromozóm, je možné ho vykresliť. Avšak na to potrebujeme poznať súradnice hradiel. Tie sa počítajú v objekte chromozómu. Na začiatku sa počítajú tak, aby vstupy boli vykreslené vľavo, s rovnakými rozostupmi od seba, vedľa nich by mali byť hradlá usporiadané do matice a úplne vpravo sa vykreslia výstupy, ktoré majú tiež rovnaké rozostupy. Výsledný obrázok je tak symetrický a prehľadný. Aby však bol kompletný, je potrebné vykresliť aj spoje medzi hradlami. Tie sa vďaka už vypočítaným a uloženým súradniciam vykreslia veľmi jednoducho — pospájajú sa správne vstupy s výstupmi. Je však potrebné počítať aj so zmenou zobrazenia — zošednutie hradiel nepodieľajúcich sa na výsledku, prípadne ich úplné vymazanie. Táto funkcionalita je riešená pomocou jednoduchého prepínača, ktorý buď vyradí zbytočné hradlá z ďalšieho spracovania alebo zmení iba farbu. V prípade, že hradlo nezošedne, použije sa buď predom definovaná farba, alebo farba, ktorú si vybral užívateľ v nastaveniach. Tieto nastavenia sa vymažú pri reštarte programu, avšak uložený rozpracovaný chromozóm si ich pamätá.

Pri načítaní chromozómu, sa po vypočítaní súradníc jednotlivých hradiel, vstupov a výstupov, vypočíta aj posun a zoom, aby sa výsledný obraz vykreslil do stredu kresliaceho plátna. Posun sa vypočíta ako rozdiel stredu plátna a grafického stredu obvodu. Stred obvodu sa nájde vyhľadaním krajných súradnic, ktoré sa od seba odčítajú, čím zistíme veľkosť strán grafickej reprezentácie obvodu. Sú to dve čísla, výška a šírka. Obe sa vydelia dvoma, takže nájdeme polovice týchto rozmerov. Tieto čísla sa potom pripočítajú k súradniciam ľavého horného rohu (kvôli prípadnému posunutiu) a výsledkom je grafický stred obvodu.

Oproti tomu zoom sa vypočíta ako pomer výšky plátna k výške chromozómu a šírky plátna k šírke chromozómu. Na vykreslenie sa použije menšie číslo z týchto dvoch. V prípade, že by sa použilo väčšie, obvod by sa na plátno zmestil iba tým jedným rozmerom (napríklad šírkou), ktorý by sme použili, a druhý rozmer by sa na plátno nezmestil celý a niektoré časti obvodu by neboli viditeľné.

V tejto časti by som rada spomenula vykresľovanie polopriesvitného bieleho obdĺžnika pod hradlom. Má slúžiť na lepšiu čitateľnosť a viditeľnosť hradla v prípade, že cez neho prechádza veľa spojov. Spoje tak budú stále viditeľné, avšak nebudú príliš rušivé.

Výsledok vykresľovania a úprav s ním spojených je vidieť na obrázkoch[1] 6.3 a 6.4.



Obrázek 6.3: Zobrazenie chromozómu hneď po otvorení.

6.2.1 Zošednutie hradiel

Pri nastavení prepínača viditeľnosti na `GREY` (jedna z hodnôt vlastného vymenovaného typu pre viditeľnosť elementov), hradlá zošednú, viď obrázok 6.5. Konkrétne sa to deje pri vykresľovaní, keď sa nastavuje farba, ktorou sa budú objekty kresliť, táto farba sa nastaví na svetlú šedú.

6.2.2 Zmiznutie hradiel

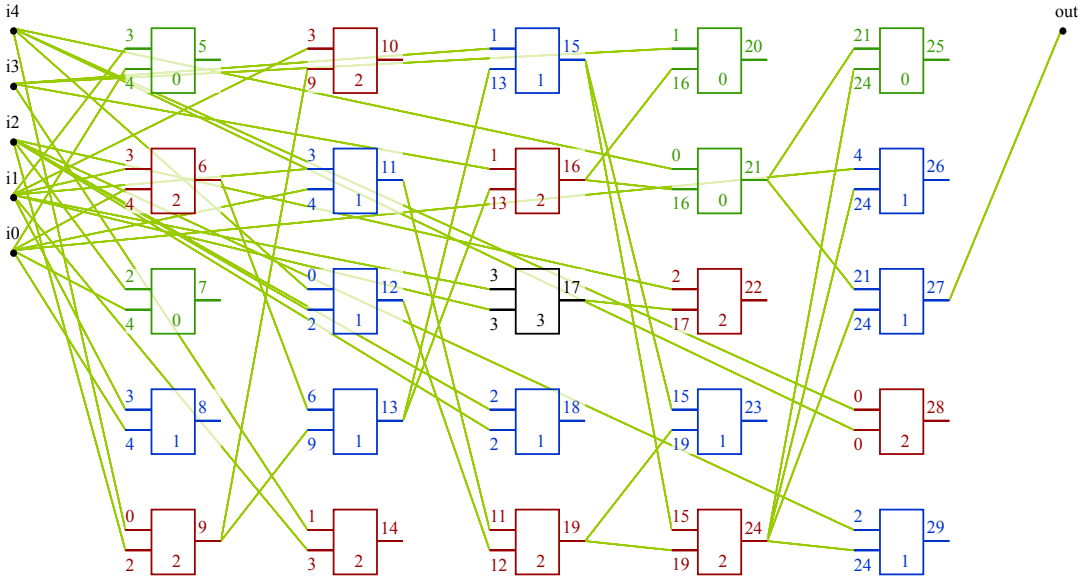
V tomto prípade sa prepínač viditeľnosti nastaví na `INVISIBLE`. Pre úplnosť, ak chce užívateľ vidieť všetky hradlá, prepínač je nastavený na `VISIBLE`. Nastavenie na `INVISIBLE` znamená, že sa tie hradlá, ktoré sa zobraziť nemajú, pri vykreslení úplne vynechajú. Tým pádom sa vykreslia iba tie správne hradlá. Viď obrázky 6.6 a 6.7.

Pri tomto vykreslení vzniká menší problém, pretože síce sa zbytočné hradlá nevykreslia, avšak stále majú súradnice, nachádzajú sa v zozname hradiel aj elementov. Riešením je pridanie podmienky, aby sa v prípade, že je prepínač viditeľnosti nastavený na `INVISIBLE`, nielen nevykreslili, ale sa s nimi ani nepočítalo a nedetekovalo sa kliknutie na nich.

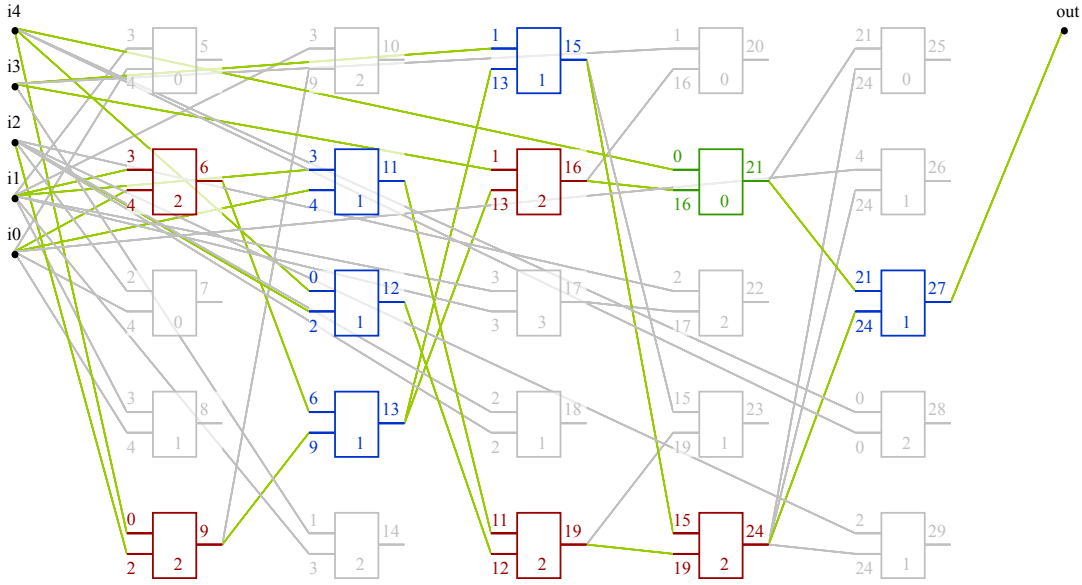
6.2.3 Detekcia hradiel nepodieľajúcich sa na výsledku

Pre miznutie aj zošednutie hradiel je potrebná detekcia hradiel nepodieľajúcich sa na výslednom obvode. Po načítaní chromozómu majú všetky hradlá nastavenú viditeľnosť na

[1] Screenshoty v tejto práci používajú kvôli jednotnosti ako označenie typu hradla čísla, program však umožňuje nastaviť symbolické znaky na označovanie hradiel.

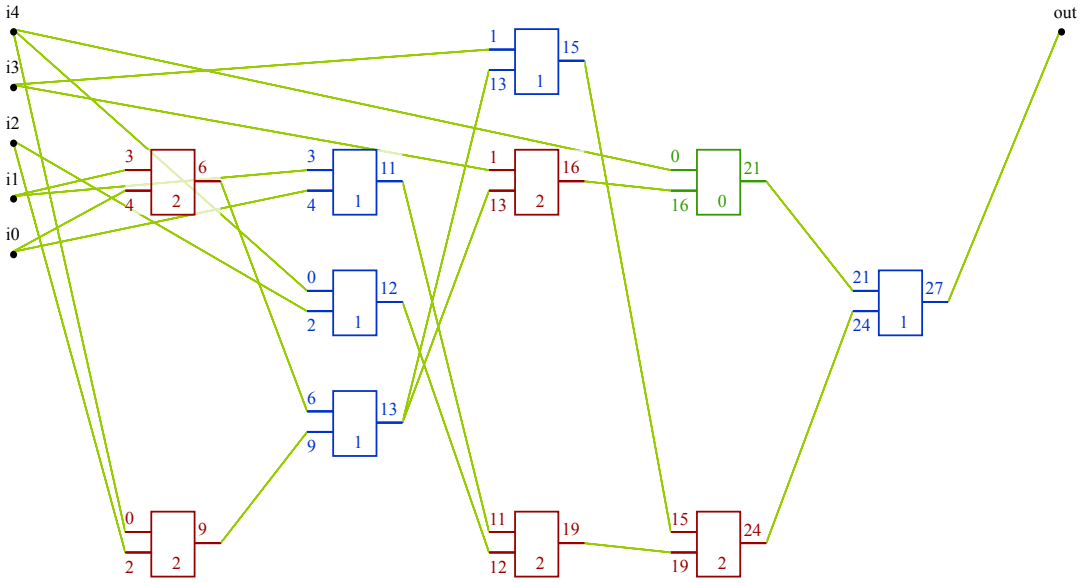


Obrázek 6.4: Zobrazenie chromozómu hneď po otvorení a zafarbení.

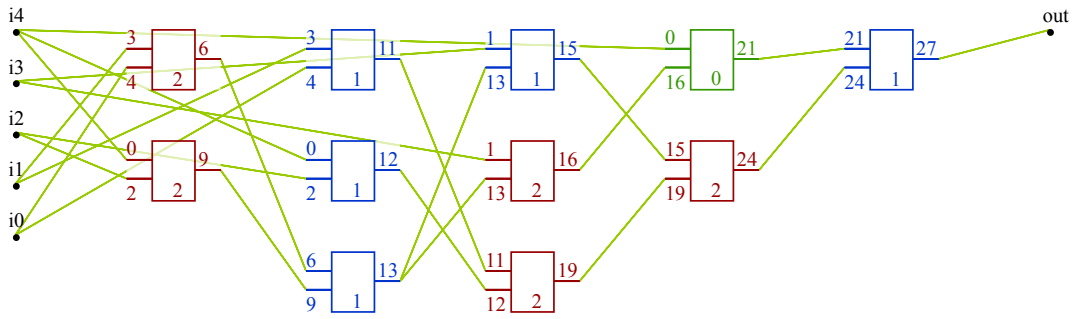


Obrázek 6.5: Zobrazenie chromozómu po zošednutí zbytočných hradiel.

`VISIBLE`. Pre zjednodušenie detekcie sa všetky nastavia na `GREY` a celý obvod sa prechádza od výstupov, pričom podieľajúcim sa hradlám sa viditeľnosť mení na `VISIBLE`. Prechádza-



Obrázek 6.6: Zobrazenie chromozómu po tom, čo užívateľ nechal zmiznúť hradlá.



Obrázek 6.7: Zobrazenie chromozómu bez zbytočných hradiel zarovnané na mriežku.

nie obvodu využíva rekurziu — vždy sa metóda na detekciu zavolá pre konkrétne hradlo s číslami jeho vstupov, ktoré vlastne označujú čísla výstupov iných hradiel. Tesne pred zavolaním seba samej však podľa čísel vstupov vyhľadá ďalšie hradlá a zmení im viditeľnosť. Špeciálnym prípadom sú výstupy. Tie majú vstup iba jeden, takže potrebovali vlastnú metódu, ktorá prejde ceý zoznam výstupov a vždy zavolá metódu na detekciu zbytočných hradiel. Algoritmy 6.2.1 a 6.2.2 sú pre popis týchto metód názornejšie.

Algoritmus 6.2.1.

```
public void findUnnecessary() {
        // vymazať všetky hradlá
        for (int i = 0; i < listOfGates.size(); i++) {
                listOfGates.get(i).visible = INVISIBLE;
        }
        // skontrolovať každý výstup
        for (int i = 0; i < number_of_out; i++) {
                // volanie rekurzívnej metódy
                int number = listOfOuts.get(i).number;
                listOfGates.get(number).visible = VISIBLE;
                setVisible(listOfGates.get(number).in1,
                 listOfGates.get(number).in2);
        }
}
```

V algoritme 6.2.1 je metóda pre prechádzanie výstupov chromozómu. Ako je vidieť, vždy sa zavolá metóda z algoritmu 6.2.2, ktorá vykonáva rekurziu.

Algoritmus 6.2.2.

```
private void setVisible(int x, int y) {
        if (x >= number_of_in) {
                listOfGates.get(x).visible = VISIBLE;
                setVisible(listOfGates.get(x).in1,
                 listOfGates.get(x).in2);
        }
        if (y >= number_of_in) {
                listOfGates.get(y).visible = VISIBLE;
                setVisible(listOfGates.get(y).in1,
                 listOfGates.get(y).in2);
        }
}
```

6.2.4 Presúvanie hradiel

Túto časť obstaráva objekt `canvas` (v Jave je to trieda `java.awt.Canvas`), čo je v preklade kresliace plátno. Je to objekt, ktorý obsahuje metódu `paint()`, ktorá sa volá automaticky a v nej je treba zariadiť vykreslenie všetkého, čo chceme mať vykreslené (napríklad čiary, nastavenie antialiasingu a pod.). Vďaka nej prebieha všetko vykresľovanie. Tento objekt však má aj metódy na zachytenie pohybu a stlačení myši. Vďaka tomu dokáže zistiť kam užívateľ klikol a kde zase tlačítko myši pustil. Presne takto totiž funguje presúvanie hradiel. Podľa súradníc, na ktoré užívateľ klikol sa vyhľadá správne hradlo, prípadne vstup alebo výstup, ktoré obsahuje tieto súradnice, objekt si ho zapamätá a potom podľa súradníc, kde užívateľ uvoľnil stlačenie, ho posunie. Posunutie je teda rozdiel medzi súradnicami stlačenia a uvoľnenia myši.

Pri presúvaní hradiel sa využíva metóda `MouseDragged()`, ktorá spôsobuje, že presúvané hradlo sa stále vykresľuje a tak je vždy vidno jeho aktuálnu polohu. Bohužiaľ to má nevhodný vedľajší efekt — obraz „bliká“. Neustálym prekresľovaním sa vždy vymaže aktuálny obraz, vypočíta nový a potom sa vykreslí. Tomuto efektu sa dá vyhnúť použitím dvoch

zásobníkov (tzv. double buffering). Vždy jeden je ten zobrazený a do druhého sa predpočíta nový obraz. Potom sa tieto dva zásobníky vzájomne vymenia, takže predpočítaný sa zobrazí. Počas počítania jedného zásobníku obsahuje ten druhý vždy dáta, ktoré sa používajú pre zobrazenie až do okamihu výmeny. Pre urýchlenie vykresľovania sa pri posúvaní hradla vypne antialiasing a polopriesvitný obdĺžnik, ktoré sú výpočetne náročnejšie a zapnú sa až po ukončení posúvania, čo je dôležité pre plynulé posúvanie s pohybom myši.

Pri zaškrtnutí možnosti **Prichytávať k mriežke** sa pod chromozómom vykreslí svetlošedá mriežka, ku ktorej sa pri posúvaní budú prichytávať hradlá. Funguje to hlavne na zaokrúhľovaní súradníc. Kvôli viditeľnosti hradla pri posúvaní sa opakovane volá `MousePressed` a `MouseReleased`, v ktorých sa vždy prepočítajú a uložia nové súradnice. Ak by sa hneď po zaokrúhlení uložili súradnice a použili by sa pri ďalšom zaokrúhľovaní, tak by sa hradlo nikam neposunulo, pretože by rozdiel bol tak malý, že súradnice by sa vždy zaokrúhlili na to isté číslo. Preto sa používajú dvoje súradnice, pričom jedny sú zaokrúhlené a druhé nie. Zaokrúhlenie sa počíta vždy z tých nezaokrúhlených, ktoré sa menia pri pohybe myši. Zaokrúhlené súradnice sa potom využijú pri výpočte všetkých súradníc hradla a tým pádom pri vykreslení. Takto je možné aby po vypnutí prichytávania k mriežke ostali hradlá na svojich miestach.

6.2.5 Posúvanie celou kresliacou plochou

Celá kresliaca plocha sa posúva veľmi podobne ako hradlá, avšak detekcia je presne opačná. V prípade, že sa nenájde žiadny element, ktorý by obsahoval kliknuté súradnice, posunie sa celá kresliaca plocha. Toto funguje ako, v súčasnosti veľmi obľúbená, náhrada skrolovania. Tento princíp sa využíva hlavne u dotykových displejov, ale aj v rôznych hrách. Je to veľmi intuitívne a pohodlné pre užívateľa, čo bolo hlavným dôvodom pre môj výber. Taktiež to uľahčilo orientáciu v kóde a ďalšiu prácu s posúvaním.

Pri posúvaní celou plochou bolo potrebné zohľadniť už vyššie spomínané prichytávanie k mriežke. Preto bolo potrebné si zapamätať, kde mriežka „začína“, tj. kde má svoj pôvodný bod $[0, 0]$, ktorý sa pri posunutí celou plochou tiež posunul. Tieto súradnice sa pred zaokrúhlením súradníc hradla od týchto súradníc odpočítajú, čím sa dostaneme na pôvodné nulové súradnice mriežky a po zaokrúhlení sa naspäť pričítajú, aby sa hradlo vykreslilo na správnom mieste. Takto sa zachováva relatívnosť súradníc voči pôvodnému začiatku, ktorý sa posúva spolu s chromozómom.

6.2.6 Zarovnanie všetkých zobrazených hradiel na mriežku

Pre zobrazenie hradiel na mriežku, tj. do „pôvodného“ stavu, je potrebné opätovne prepočítať všetky súradnice s ohľadom na aktuálne zobrazenie — znamená to brať do úvahy či sa zobrazujú všetky hradlá, alebo len niektoré. Na to samozrejme slúži už spomínaný prepínač viditeľnosti. Po prepočítaní stačí už len prekresliť kresliacu plochu metódou `repaint()`.

6.2.7 Vykreslenie všetkých hradiel po úpravách

V prípade, že užívateľ chce vynulovať všetky zmeny, ktoré na chromozóme urobil, je najprv potrebné všetkým hradlám nastaviť viditeľnosť na `VISIBLE`, aby sa mohli súradnice prepočítať pre všetky hradlá, a aby malo každé hradlo svoju farbu, tj. žiadne nezošedlo. Potom sa teda súradnice prepočítajú, znovu dostanú všetky elementy svoje pôvodné rozmiestnenie. Podľa toho sa vykreslia.

6.2.8 Zoom

Zoom funguje za pomoci transformačnej matice. Je to veľmi pohodlné a prehľadné riešenie, podporované všetkými triedami odvodenými od a používajucími objekt `Graphics`, tj. objekt `canvas` a export do SVG. Taktiež je veľmi elegantné z hľadiska grafiky. V Jave na to postačuje jeden príkaz, ktorý sa postará o výsledok.

```
g.setTransform(new AffineTransform(zoom, 0, 0, zoom, 0, 0));
```

Premenná zoom sa počíta vždy pri stlačení tlačidla na priblíženie alebo oddialenie, alebo pri automatickom roztiahnutí chromozómu na veľkosť kresliacej plochy. Tento príkaz sa volá vždy pri vykresľovaní chromozómu a vykresľuje sa pri akejkoľvek zmene obrazu.

6.2.9 Antialiasing

Každý užívateľ sa radšej pozerá na pekne vyhladené čiary, písmená, aj iné objekty. A presne to má na starosti antialiasing. Je možné ho prepínať ručne, pomocou prepínača v menu, avšak prepína sa aj sám a to konkrétne pri pohybovaní elementami alebo celou kresliacou plochou. Vypne sa akonáhle užívateľ klikne niekam na ploche (vrátane vnútra elementov) a zapne sa pri „pustení“ objektu alebo plochy. Samozrejmé je rešpektovanie voľby užívateľa, takže v prípade, že antialiasing bol vypnutý už pred posúvaním, po ňom sa nezapne.

6.2.10 Nastavenie farieb a vlastností

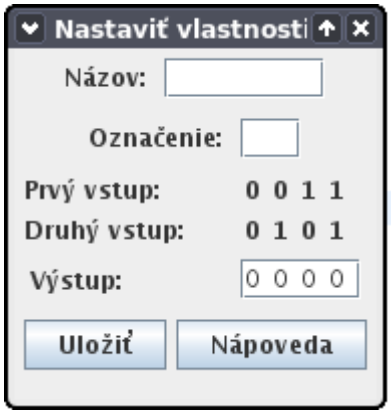
Farby hradiel a spojov sa nastavujú vždy po spustení na čiernu. Čierna je neutrálna farba, na vykreslenie na bielom pozadí najlepšia, avšak nie je veľmi prehľadná pri zobrazení väčšieho množstva hradiel a spojov. Preto má užívateľ možnosť zmeniť všetky farby. K dispozícii má položku **Vlastnosti** v menu, ktorá obsahuje okno s výpisom funkcií a farieb (obrázok 6.9). Farby sa dajú meniť v užívateľsky prívetivom vzorkovníku, ktorý sa otvorí pri kliknutí na tlačidlo **Zmeniť farbu**. Po vybratí farby a potvrdení voľby sa táto farba automaticky uloží.

V tomto okne je tiež možné nastaviť šírku a výšku rozostupov medzi elementami chromozómu pre prípad, že by užívateľ chcel upravené zobrazenie. Aby sa zmena tohto nastavenia prejavila, je potrebné znovu prepočítať súradnice. Aby sa obraz opäť vykreslil do stredu plátna, je potrebné tiež vypočítať posun a potrebný zoom. Toto všetko nastane po kliknutí v menu na položku **Vyresetovanie zobrazenia**.

Po kliknutí na **Nastaviť funkciu** sa zobrazí okno (obrázok 6.8), v ktorom je možné modifikovať názov funkcie, symbolický znak na označenie funkcie namiesto čísla a samotné výsledky funkcie, ktoré používa simulácia. Ako presne nastaviť výsledky funkcie a čo ktoré čísla znamenajú je uvedené v kapitole o simulácii, prípadne v nápovede v okne nastavení funkcií.

6.3 Export do SVG formátu

O tento export sa stará knižnica Batik SVG Toolkit (spomínaná v kapitole 5.3). Je to knižnica pre manipuláciu s obrázkami v SVG. Najprv je potrebné vytvoriť dokument so štruktúrou DOM, do ktorého sa bude kresliť. Kreslenie prebieha rovnako ako do `Graphics` objektu, ktorý sa používa napríklad na kreslenie do canvasu (kresliaca plocha v Jave). Takto je možné kresliť jednoduché geometrické útvary, využívať farby, písmo atď. Knižnica



Obrázek 6.8: Okno nastaviteľných vlastností funkcií.

sa postará o ich zápis do formátu SVG. S obrázkom vo formáte SVG sa dá robiť viacero akcií — uložiť, vykresliť alebo akokoľvek ďalej manipulovať. Na knižnici je v súčasnosti ešte práce dosť, avšak pre účely tejto práce plne postačuje. Vzhľadom na to, že obrázok vo formáte SVG je možné veľmi jednoducho vykresliť priamo na kresliacu plochu programu, bola možnosť mať obrázok už od začiatku v tomto formáte. Keďže sa nedá počítať s tým, že užívateľ bude chcieť vždy vyexportovať obrázok do tohto formátu, rozhodla som sa túto možnosť nevyužiť a vykresľovať radšej do objektu `Graphics`. Export do SVG sa deje teda nezávisle na objekte `Graphics`, avšak do SVG sa uloží vždy presne to, čo užívateľ vidí, vďaka ukladaniu súradníc a zoomu.

6.4 Export do interného formátu (rozpracovaný chromozóm)

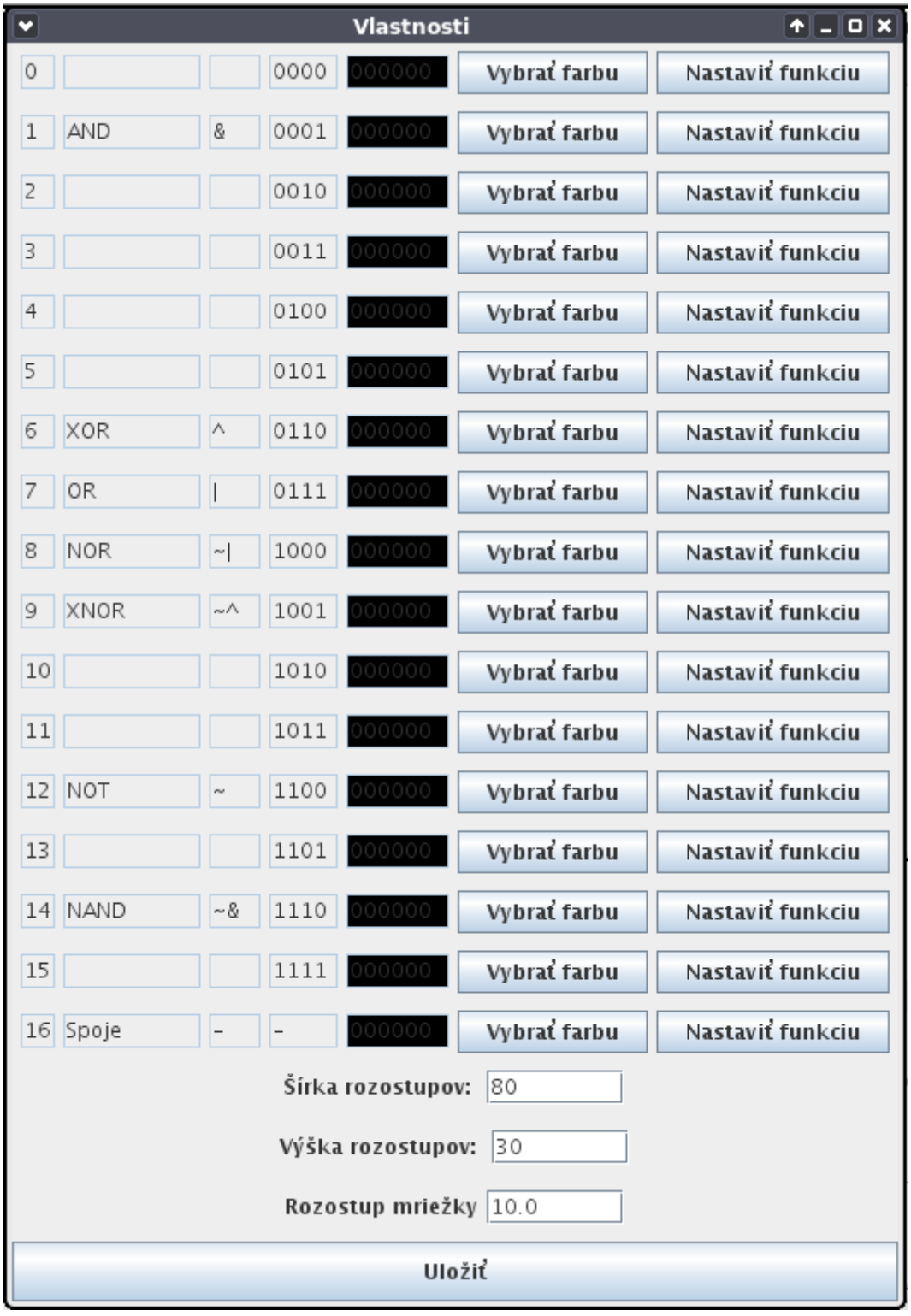
Java má možnosť serializácie objektov. To znamená, že sa stav daného objektu prevedie do binárnej podoby. Z tejto podoby je možné jeho stav opätovne obnoviť. Interným formátom je binárny formát serializácie v Jave, ktorý je v tejto práci dobre použiteľný. Všetky informácie potrebné pre vykreslenie chromozómu (vstupy, výstupy, hradlá, farby...) sú uložené v objekte `Chromozome`. Tento objekt implementuje rozhranie `Serializable`, čo znamená, že je možné ho serializovať. Aby sa však uložili aj vstupy, výstupy a hradlá, ktoré majú svoje vlastné triedy (programátorom vytvorené, tj. Java ich priamo neobsahuje), bolo potrebné pridať implementáciu rozhrania `Serializable` aj do nich.

Po serializácii objektu je potrebné ho uložiť do súboru. Tento súbor má príponu `.rch`, čo sa dá chápať ako „Rozpracovaný CHromozóm“, alebo ako prehodenie písmen prípony súboru s chromozómom `.chr`.

6.5 Simulácia chromozómu

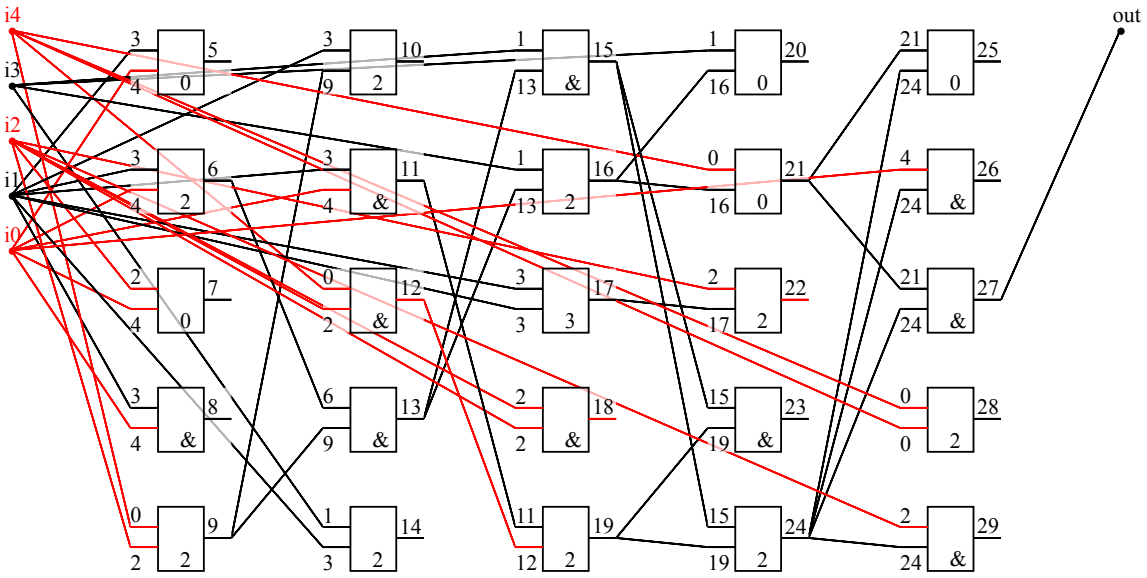
Na spustenie tejto funkcie slúži tlačidlo **Prepnúť do režimu simulácie**. V tomto režime nie je možné posúvať hradlami ani celou plochou, avšak všetky ostatné manipulácie fungujú (antialiasing, zoom, zošednutie/vymazanie zbytočných hradiel a pod.).

Pri zobrazení simulácie sa využíva v maximálnej možnej miere grafické znázornenie. Ak je na elemente (v tomto prípade je ním vstup obvodu, výstup obvodu, spoj alebo vstupy a výstup hradla) nula, tak je ten element čierny. Ak je tam jednička, element zčervená (tj. akoby svietil, je dostatočne výrazný).



Obrázek 6.9: Okno nastaviteľných vlastností.

Samotná simulácia je realizovaná pomocou bitových operácií. Pre výpočet je potrebné vedieť, aká hodnota je na príslušnej pozícii. Tá sa zisťuje pomocou jednotky posunutej o



Obrázek 6.10: Simulácia. Červenou sú spoje, na ktorých je hodnota práve jedna. Čierne obsahujú nulu.

n bitov. Zisťuje sa to pomocou algoritmu 6.5.1. Premenná $in1val$ je hodnota na prvom vstupe, analogicky k nej je $in2val$ hodnota na druhom vstupe. Posúvať bude potrebné o 0, 1, 2 a 3 bity, pretože máme 4 možné hodnoty. K tomu nám pomôže negácia ($\sim$). Tabuľka vstupov (6.1) totiž po negácii dáva vhodné výsledky ak negované hodnoty sčítame, viď. tabuľka 6.2. Ak by sme však na negované hodnoty nepoužili masku pomocou AND (&), z čísla 1 by vzniklo číslo, ktoré by v binárnej sústave malo na začiatku samé jednotky a posledná by bola nula, čo by k výsledku nijak nepomohlo. Po použití masky ostane iba posledná číslica, tj. tá ktorá je potrebná pre výsledok.

Algoritmus 6.5.1.

```
n = (((~in1val) & 1) << 1) + ((~in2val) & 1);
```

prvý vstup	0	0	1	1
druhý vstup	0	1	0	1

Tabulka 6.1: Tabuľka vstupov

`((~in1val) & 1) << 1`	10	10	00	00
`(~in2val) & 1`	01	00	01	00
súčet predchádzajúcich dvoch riadkov	11	10	01	00

Tabulka 6.2: Tabuľka vstupov po negácii a bitovom posune a tretí riadok je výsledok po sčítaní (pre krajší vzhľad sú nuly doplnené v druhom riadku)

V tejto chvíli už existuje hodnota, ktorá je potrebná pre simuláciu. Simulácia je popísaná algoritmom 6.5.2. Premenná *value* je výsledná hodnota hradla, *func* je binárna reprezentácia funkcie hradla. Takúto binárnu reprezentáciu vidíme na treťom riadku v tabuľke 6.3, tj. je to binárne číslo 0001, ktoré sa v premennej uchováva v podobe integeru (celého čísla), takže sa na ňom bez veľkých problémov dajú použiť bitové operácie. Najprv sa teda číslo jedna posunie o predtým vypočítaný počet bitov n, čím sa určí poloha hľadaného výsledku v binárnej reprezentácii funkcie hradla. Potom sa táto maska použije pomocou operácie AND na binárnu reprezentáciu funkcie hradla a v prípade, že výsledkom tejto operácie bude nula, je aj celkovým výsledkom nula. Ak je to akékoľvek iné číslo, výslednou hodnotou bude jedna.

Algoritmus 6.5.2.

```
value = (func & (1 << n)) == 0 ? 0 : 1;
```

Na tomto mieste by som chcela upozorniť na operáciu NOT. Tá totiž využíva iba jeden vstup. V stave v akom je definovaná v programe, bude fungovať pre hodnoty prvého vstupu, tj. pre poradie hodnôt prvého vstupu, ktoré je uvedené v tabuľkách vyššie, budú jeho výsledky 1100. Na druhom vstupe teda nezáleží.

prvý vstup	0	0	1	1
druhý vstup	0	1	0	1
výstup	0	0	0	1

Tabulka 6.3: Prvé dva riadky sú vstupy, posledný riadok sú výstupy po operácii AND

Kapitola 7

Záver

Cieľom tejto bakalárskej práce bolo hlavne uľahčiť a sprehľadniť zobrazovanie chromozómov na rôznych OS. Dôležitá bola prenositeľnosť na iné OS, rovnako ako export do vektorového formátu, jednoduchá simulácia a možnosť manipulácie s chromozómom (posúvanie hradiel, zoom, farbenie hradiel a spojov atď). Tento cieľ bol splnený.

Základom bolo implementovanie exportu do vektorového formátu pre ďalšiu prácu s chromozómom, avšak už tento program sa správa ako jednoduchý editor. Je možné posúvať jednotlivé hradlá, približovať a odďaľovať obraz, posúvať celý obraz, odstraňovať hradlá, ktoré sa nepodieľajú na výsledku obvodu. Tiež je možné si hradlá rôzne zafarbiť kvôli prehľadnosti. Všetky tieto vlastnosti pomôžu programátorom CGP k lepšej orientácii a prehľadnosti pri ich práci.

Nespornou výhodou je aj možnosť jednoduchej simulácie v tomto programe. Umožňuje programátorom vyskúšať si obvod okamžite a podľa výsledkov ho prípadne upraviť a znovu zobraziť. Taktiež umožňuje kontrolu správnosti zadania logických funkcií jednotlivým hradlám.

Tento projekt je síce založený na podobnom programe, avšak oproti nemu tento projekt je možné využiť na rôznych operačných systémoch. Taktiež je možné ďalej pracovať s grafickou reprezentáciou chromozómu, upravovať ho a uskutočniť jednoduchú simuláciu.

Veľmi dôležité pre genetických programátorov je samotné vytváranie chromozómov. Určite by bolo veľmi pohodlné a praktické, ak by táto funkcionalita bola niekedy v budúcnosti súčasťou programu. V tomto prípade by program mal obsahovať implementáciu jedného z genetických algoritmov (môže ich mať samozrejme aj viac) a k tomu odpovedajúce grafické rozhranie pre zadávanie parametrov algoritmu.

Literatura

[1] Obrázok Moiré. online, 27. apríla 2010.
URL http://www.freedumb.net/img/corpicon.jpg

[2] Scalable Vector Graphics. online, 27. apríla 2010.
URL http://sk.wikipedia.org/wiki/Scalable_Vector_Graphics

[3] Úvod do vektorovej grafiky. online, 27. apríla 2010.
URL http://grafika.sk/clanok/vektorova-grafika-uvod/

[4] Burget, R.: Internetové aplikace (WAP) II. 2007.

[5] Hrádek, J.: Aliasing & Antialiasing. online, 27. apríla 2010.
URL
http://herakles.zcu.cz/education/apg_2002_2003/hradek/html/Aliasing.html

[6] Kršek, P.: Základy počítačové grafiky. 2005.

[7] Miller, J. F.; Thomson, P.: *Cartesian Genetic Programming*. 2000.

[8] Vašíček, Z.; Sekanina, L.: *Evoluční návrh kombinačních obvodů*. Electrorevue, 43/2004.

Dodatek A

Obsah CD

- Elektronická verzia technickej správy vo forme PDF a jej zdrojový text pre LaTeX
- Zdrojový kód programu v Jave
- Ant súbor pre vytvorenie .jar archívu
- Spustiteľný .jar archív
- Ukážkový chromozóm