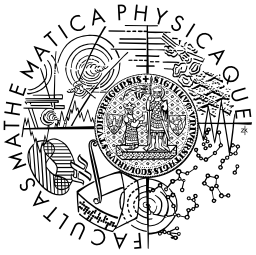


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Jan Horák

Webový editor a simulátor hradlových sítí

Katedra softwarového inženýrství

Vedoucí bakalářské práce:	RNDr. Martin Kruliš, Ph.D.
Studijní program:	Informatika
Studijní obor:	Softwarové a datové inženýrství

Praha 2018

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Děkuji svému vedoucímu práce RNDr. Martinu Krulišovi, Ph.D. za cenné rady, připomínky a čas, které mi během psaní bakalářské práce věnoval.

Název práce: Webový editor a simulátor hradlových sítí

Autor: Jan Horák

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Martin Kruliš, Ph.D., Katedra softwarového inženýrství

Abstrakt: Jedním z témat vyučovaných v rámci informatiky jsou principy hradlové logiky. Pomůckou k lepšímu pochopení této látky může být aplikace, která studentům umožní sestavovat a zkoumat hradlové sítě. Měla by být jednoduše použitelná a její instalace by měla být snadná nezávisle na vybavení počítače. Podařilo se nám vytvořit prototyp webové aplikace, která tyto požadavky splňuje a umožňuje sestavování logických obvodů ze všech základních typů hradel, a která názorně zobrazuje hodnotu na každém vodiči. Implementovali jsme i další funkce, které zpříjemní použití aplikace — neomezenou editační plochu, úvodní tutoriál, možnost importu a exportu sítí do souboru či knihovnu logických obvodů, které lze importovat v podobě sítě či samostatné komponenty. Aplikace je díky objektovému přístupu a rozdělení jednotlivých částí do modulů snadno rozšiřitelná a její zdrojový kód je uvolněn pod open-source licencí.

Klíčová slova: hradla editor simulátor webová aplikace

Title: Web Editor and Simulator of Logic Gate Networks

Author: Jan Horák

Department: Department of Software Engineering

Supervisor: RNDr. Martin Kruliš, Ph.D., Department of Software Engineering

Abstract: One of the topics taught in computer science is the principles of logic gates. An application that allows students to experiment with logic gates and gate networks can be used as a tool to provide better understanding of the topic. However, in order for the application to be usable, it should be available independently of the software on the user's computer. As a part of this thesis, we were able to implement such application. It provides users with the functionality of constructing circuits from logic gates interconnected with wires. The application is able to simulate the network and display logic values on each of the wire. Additional functionality has been provided to make the application more enjoyable — unlimited canvas for circuit construction, a simple tutorial introducing new users to the basics of the application, import and export functionality and a library of logic circuits that can be imported onto the editing canvas in the form of gate networks or as single components. The application is easily expandable and the source code is available under an open-source license.

Keywords: logic gates editor simulator web application

Obsah

Úvod

Součástí výuky středoškolské informatiky a některých informatických vysokoškolských předmětů je i seznámení studentů s logickými obvody. Jako podpůrný materiál lze u výuky použít schémata logických obvodů či nějakou aplikaci, která umožňuje hradlové sítě sestavovat, upravovat a simulovat.

Aby však taková aplikace byla skutečně pomůckou při výuce a přispívala snadnějšímu pochopení látky, musí být dostatečně intuitivní — měla by tedy používat ustálené označení jednotlivých logických prvků (hradel) a názorně zobrazovat logický stav jednotlivých vstupních a výstupních pinů.

Zároveň musí aplikace poskytnout uživateli při tvorbě sítě dostatečnou volnost, aby omezení aplikace nevedla k horší přehlednosti zobrazované sítě. Uživatel by měl mít tedy možnost hradla po přidání na síť posouvat, otáčet jimi, případně zobrazení celé sítě libovolně přiblížit, oddálit či přesunout tak, jak mu to vyhovuje. Toho může využít například vyučující k vysvětlení funkce nějaké konkrétní části složitější sítě — může ji třeba přiblížit a vysvětlit fungování jednotlivých komponent.

Kromě toho by měla být aplikace uživatelsky přívětivá i po stránce instalace. Uživateli by pro spuštění aplikace měly stačit znalosti práce s počítačem na uživatelské úrovni. Jednou z možností, jak jednoduché instalace dosáhnout, je vytvořit webovou aplikaci, která by ke svému fungování využívala jen standardizovaného aplikačního rozhraní webového prohlížeče. Díky tomuto přístupu pak koncový uživatel aplikace nemusí nic instalovat do svého počítače a pro spuštění mu stačí navštívit správnou webovou adresu.

Existuje mnoho nástrojů, které slouží k simulaci hradlových sítí, nezřídka jsou však zaměřeny především na složitější integrované obvody (a neumožňují tedy práci přímo s hradly), vyžadují složitější instalaci, nebo postrádají některou jinou z výše uvedených vlastností, potřebných pro to, aby byly příjemně použitelné ve výuce.

Nejdůležitější vlastnosti, které musí aplikace splňovat, jsme zformulovali do následujícího seznamu:

- Lze ji jednoduše spustit, ideálně bez nutnosti instalace.
- Umožní práci se základními typy hradel (AND, OR, NOT, XOR, NAND, NOR, XNOR), jejich propojování pomocí vodičů a nastavování vstupních hodnot hradlové sítě.
- Dokáže takto vytvořený obvod odsimulovat a zobrazit hodnoty jednotlivých vodičů. Během simulace dokáže detekovat oscilaci a korektně ji zobrazit.
- Obsahuje funkce pro import a export sítí ve srozumitelném formátu.
- Obsahuje knihovnu základních logických obvodů, které lze do sítě importovat v podobě hradlové sítě nebo jako samostatnou komponentu.
- Nabídne uživateli tutoriál, který jej seznámí se základním používáním aplikace.

- Je navržena tak, aby ji bylo možno snadno dále rozšiřovat, její zdrojový kód je přehledný a dostupný případným zájemcům o další vývoj.

V rámci této práce jsme implementovali prototyp webové aplikace, která tyto požadavky splňuje a je díky otevřenému a zdokumentovanému zdrojovému kódu snadno rozšiřitelná.

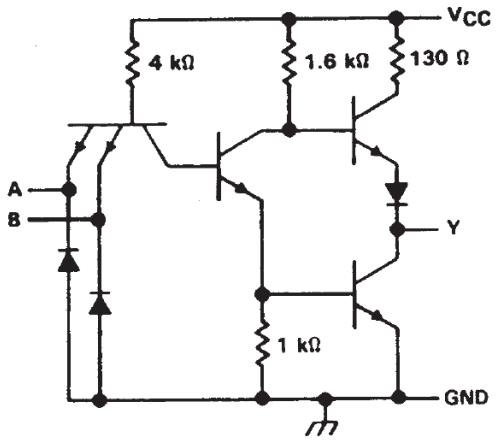
Přehled kapitol

V kapitole *1 Úvod do problematiky* se věnujeme podrobnému popisu problematiky, provádíme analýzu funkčních a nefunkčních požadavků a popisujeme výsledky rešerše podobného softwaru, který lze použít k sestavování a simulaci hradlových sítí. Kapitola *3 Popis řešení* pojednává o řešení problémů popsaných v kapitole 1 na úrovni popisu významných algoritmů použitých v aplikaci a důležitých rozhodnutních o uživatelském prostředí aplikace. *4 Uživatelská dokumentace* popisuje vytvořenou aplikaci z pohledu uživatele. Rozebírá typy a použité značení jednotlivých komponent, grafické znázornění hodnot na vodičích a další možnosti uživatelské interakce s aplikací. Kapitola *5 Programátorská dokumentace* podrobněji popisuje objektový návrh aplikace a strukturu zdrojového kódu. V kapitole *6 Vyhodnocení* se věnujeme zpětnému pohledu na proces tvorby aplikace, věnujeme se řešeným problémům a uvádíme ukázky implementace hradlových sítí v naší aplikaci.

1. Úvod do problematiky

Nezbytnými součástkami pro konstrukci počítačů a dalších digitálních zařízení jsou digitální integrované obvody, které umožňují zařízením pracovat s binárními informacemi. V digitální technice je nositelem takové informace většinou elektrické napětí, kde jsou hodnoty reprezentovány pomocí dvou napěťových úrovní, označovaných zpravidla H a L pro vysokou a nízkou úroveň napětí. Vysoká úroveň je blízká napájecímu napětí, nízká úroveň se blíží 0 voltům. Tyto dvě úrovně reprezentují hodnoty 1 a 0 používané v digitální logice, vysoká úrověň napětí zpravidla reprezentuje logickou jedničku, nízká úroveň napětí pak logickou nulu [12].

Fyzická realizace logické funkce je označována jako *hradlo* či *logický člen* [21]. K jejich implementaci jsou běžně používány tranzistory, lze je však realizovat například i pomocí elektromagnetických relé nebo optických jevů [12; 10]. Příkladem realizace pomocí tranzistorů může být například standard TTL, ve kterém je logická jednička reprezentována napětím 5 V, logická nula pak napětím 0 V [12]. Například obvod implementující logickou funkci NAND lze sestrojit pomocí tranzistorů na základě tohoto standardu způsobem uvedeným na obrázku 1.1.



Obrázek 1.1: Realizace hradla NAND pomocí tranzistorů. V_{CC} značí napájení, *GND* referenční zem, A, B jsou vstupní piny a Y je výstupní pin [33].

vstup		**AND** $\wedge$	**OR** $\vee$	**NAND** $\mid$	**NOR** $\downarrow$	**XOR** $\veebar$	**XNOR** $\leftrightarrow$
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	1	0	0	0	1

Tabulka 1.1: Pravdivostní tabulka pro významné binární logické funkce [21]

Funkce jednotlivých hradel jsou založeny na odpovídajících funkcích v Booleově logice a jejich propojování je založeno na sestavování logických formulí. Pro pochopení kontextu práce je proto nutno rozumět základům Booleovy logiky.

Booleova logika operuje nad hodnotami $\{0, 1\}$, běžně označovanými jako *nepravda* a *pravda*, pomocí unárních a binárních logických funkcí. V rámci této práce

se zabýváme *úplně zadanými logickými funkcemi*, tedy funkcemi, jejichž výstupní hodnota je definována pro všechny kombinace vstupních logických proměnných.

Tabulka 1.1 popisuje základní binární logické funkce. Kromě těchto funkcí je ještě významná unární funkce NOT, která provádí negaci vstupní hodnoty.

Množinu logických funkcí, pomocí kterých lze realizovat libovolnou logickou funkci, nazýváme *úplným systémem logických funkcí*. Úplný systém logických funkcí tvoří například dvojice AND a NOT či NOT a OR. K realizaci libovolné logické funkce lze využít i samotnou funkci NAND či NOR [28]. Této vlastnosti lze využít pro konstrukci hradel — například lze zapojením několika obvodů implementujících hradlo NAND (viz obrázek 1.1) sestrojit libovolný jiný logický člen.

Skládání logických funkcí do logických formulí odpovídá propojování vstupů a výstupů jednotlivých logických hradel do hradlových sítí. Vyhodnocení logických formulí nastavením hodnot proměnných pak odpovídá vyhodnocení výstupů hradlových sítí nastavením vstupních hodnot na volných vstupních pinech hradel.

Jako příklad převodu logické formule na hradlovou síť můžeme použít hradlovou realizaci sčítání dvou jednobitových čísel. Pracujeme tedy se dvěma vstupními proměnnými — a a b, vyjadřujícími sčítané hodnoty, a chceme dosáhnout dvou výstupů — součtu čísel s a přenosu jedničky na vyšší řád c. Požadované výstupy s a c v závislosti na vstupních hodnotách a a b jsou zapsány v tabulce 1.2.

a	b	**součet** (s)	**přenos** (c)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

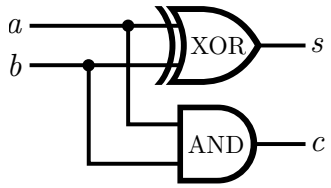
Tabulka 1.2: Pravdivostní tabulka pro obvod sčítající binární hodnoty a a b

Vztah s a c ke vstupním hodnotám lze zapsat i pomocí dvou logických formulí. Jedna bude vyjadřovat součet hodnot a a b, druhá pak přenos případné zbývající jedničky na vyšší řád:

$$s \leftrightarrow a \veebar b \qquad \text{(součet)}$$
$$c \leftrightarrow a \wedge b \qquad \text{(přenos jedničky)}$$

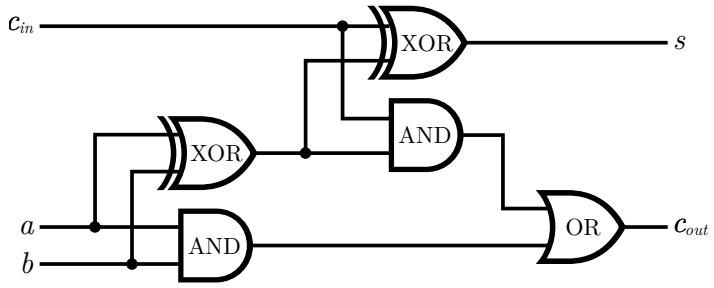
Tyto dvě logické formule lze snadno převést do hradlové reprezentace nahrazením funkcí za logická hradla a vhodným označením vstupních a výstupních pinů. Neboť navíc od obvodu očekáváme, že jeho obě části (pro součet a pro přenos jedničky) budou vždy zpracovávat stejný vstup, můžeme jejich vstupní piny propojit a vytvořit tak jednu hradlovou síť se dvěma vstupy a dvěma výstupy, jak je znázorněno na obrázku 1.2.

Pokud bychom chtěli tuto hradlovou síť využít k sestavení obvodu, který by byl schopen sčítat binární čísla o více bitech, je nutno u každého bitu vzít v potaz případné přičtení jedničky z nižšího řádu. Pracujeme tedy se třemi vstupy — k bitům a a b přibývá ještě bit c_{in}, který značí přenos jedničky. Výstupy zůstávají stejné, jen pro přehlednost značíme přebývající jedničku c_{out} místo původního c. Je zřejmé, že požadované funkcionality docílíme připojením druhého sčítacího obvodu, který na vstupu přijme výstup s původního obvodu a hodnotu c_{in}. V takové



Obrázek 1.2: Sčítání binárních hodnot pomocí hradel

podobě by logický obvod měl dva výstupy pro přenos jedničky. Jeden z nich indikuje přebývající jedničku vzniklou v situaci, kdy součet hodnot a a b je 2 (tedy oba vstupy byly nastaveny na hodnotu 1), druhý indikuje případ, kdy součet vstupů a a b je 1 a vstup c_{in} je nastaven na 1. Informace z těchto dvou výstupů můžeme sjednotit do jednoho, který indikuje přebývající jedničku, pomocí hradla XOR nebo OR, jak je znázorněno na obrázku 1.3.



Obrázek 1.3: Sčítání binárních hodnot pomocí hradel s přenosem jedničky

Kombinací menších logických obvodů můžeme sestavit větší logické moduly se složitější funkcionalitou. Například lze zapojením několika hradlových sítí pro sčítání binárních hodnot s přenosem jedničky sestavit obvod sčítající dvě čtyřbitová čísla v binární reprezentaci.

Aplikace vytvořená v rámci této práce umožňuje sestavování a simulaci logických obvodů na této úrovni abstrakce — uživatel pracuje s hodnotami z digitální logiky a s hradly, jejichž propojováním vytváří hradlové sítě, které lze simulovat nastavováním hodnot na volných vstupních pinech.

2. Analýza problému

2.1 Funkční požadavky

2.1.1 Editace hradlových sítí

Uživatel má být schopen v aplikaci sestavovat logické obvody propojováním jednotlivých hradel.

Z tohoto požadavku plyne několik dílčích problémů — je nutné, aby aplikace obsahovala editační plochu, na kterou jsou hradla uživatelem umisťována a která nabízí rozhraní pro práci se sestavovanou hradlovou sítí. Hradla na editační ploše musejí mít vyznačené vstupní a výstupní piny, které může uživatel propojovat pomocí vodičů.

Editační plocha má uživateli poskytovat rozhraní pro editaci hradlových sítí. Ač je základní funkce editační plochy jasně daná, nabízejí se různá řešení jednotlivých požadovaných funkcí.

Přidávání a odebírání hradel

Jednou z nejdůležitějších funkcí je nabídka hradel, které může uživatel přidávat na editační plochu. Nabízejí se dvě základní řešení tohoto problému. Prvním řešením je kontextové menu, které se uživateli zobrazí po kliknutí pravým tlačítkem myši na editační plochu, druhým řešením je pak zobrazení postranního menu, které by se nacházelo jednom z krajů okna aplikace.

Výhody řešení pomocí kontextové nabídky jsou následující:

- Přidávání a odebírání hradel může být implementováno v rámci jednoho kontextového menu, volba pro odstranění hradla může být zobrazena pouze při zobrazení nabídky po kliknutí pravým tlačítkem myši na nějaké hradlo.
- Pozice myši při zobrazení nabídky může být využita k umístění přidávaného hradla. Při použití na větších monitorech (či při použití myši s větší citlivostí) může být výhodou, že by uživatel pro přidání hradla nemusel dojet myší až ke straně obrazovky a zpět, jak by tomu bylo u druhého řešení.
- Kontextová nabídka se uživateli zobrazuje pouze na jeho vyžádání a když není zrovna používána, nezabírá žádné místo na editační ploše.

Nevýhody použití kontextové nabídky:

- Přidání hradla stojí uživatele nejméně dvě kliknutí (jedním kliknutím otevře nabídku, druhým kliknutím přidá hradlo). Implementace pomocí postranní nabídky by tedy uživateli ušetřila jedno kliknutí (v případě, že by se hradla přidávala na editační plochu přetažením myší).
- Toto řešení může připadat některým uživatelům méně intuitivní, než řešení pomocí postranní nabídky.

Výhody řešení používajícího postranní nabídku na kraji okna:

- Přidání hradla na editační plochu je o jedno kliknutí rychlejší, než u řešení pomocí kontextové nabídky.
- Pro některé uživatele může být tato implementace intuitivnější, neboť je nabídka hradel stále viditelná.

Nevýhody použití postranní nabídky:

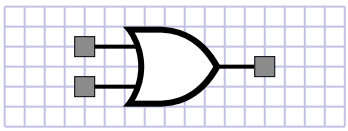
- Pokud nemá boční nabídka zabírat příliš velkou část editační plochy, je její kapacita výrazně omezena velikostí okna. Na menších oknech by tedy nebylo možno zobrazit uživateli nabídku všech hradel. Bylo by tedy nutno odhadnout ta nejpoužívanější hradla a ostatní zobrazit až v sekundární nabídce na vyžádání uživatele.
- Odebírání hradel by muselo být implementováno prostřednictvím speciální kontextové nabídky, případně by bylo možno hradla odebírat kliknutím a následným výběrem odpovídající volby v boční nabídce. Obě řešení se jeví složitější, než integrace odstranění hradla přímo do kontextové nabídky popsané v první variantě.

Rozhodli jsme se pro první řešení. Hlavním důvodem byla maximalizace editační plochy a sjednocení rozhraní pro přidávání a odebírání hradel. Při případném rozšiřování aplikace by bylo možné implementovat i postranní menu s několika nejpoužívanějšími hradly pro jejich rychlejší přidávání na editační plochu, které by se zobrazovalo pouze na dostatečně velkých obrazovkách.

Zachytávání komponent na mřížku, rozlišení mřížky

Existují dvě varianty pro umisťování hradel na editační plochu. Hradla mohou být na editační ploše umístěna volně nebo může být jejich poloha zachytávána na mřížku. Vzhledem k tomu, že by nezachytávání hradel na mřížku způsobilo zhoršení přehlednosti editační plochy a uživatel by musel strávit více času přesným polohováním jednotnlivých komponent ve snaze učinit editační plochu přehlednější, rozhodli jsme se implementovat zachytávání komponent na mřížku.

Důležitý byl výběr jemnosti mřížky vzhledem k velikosti hradel na editační ploše. Příliš jemná mřížka by měla obdobný efekt jako nezachytávání na mřížku, příliš hrubá mřížka by uživatele naopak zbytečně omezovala. Vhodné rozlišení mřížky jsme odvodili z použitých symbolů pro hradla tak, že jsme zvolili co nejhrubší mřížku, při které jsou vstupní a výstupní piny zarovnány na mřížce (viz obrázek 2.1).



Obrázek 2.1: Hradlo OR na mřížce editační plochy

Velikost editační plochy

Editační plocha může mít omezenou velikost či může být neomezená. Oba přístupy vyžadují pro dosažení uživatelské přívětivosti funkcionalitu přibližování a oddalování zobrazovaného výřezu plochy.

Výhody omezené editační plochy jsou následující:

- Pokud je omezená velikost určena podle rozměrů okna aplikace, uživatel má vždy přehled o všech komponentách na editační ploše.
- Editační plochu s omezenou velikostí lze jednoduše implementovat.

Nevýhody omezené editační plochy:

- Pevně daná velikost editační plochy může být omezující pro rozměrnější hradlové sítě, zejména pokud jsou rozměry plochy odvozeny z rozměrů okna aplikace.
- Pokud by byla omezená velikost plochy určena rozměry okna aplikace a lišila by se tedy na jednotlivých zařízeních, může při importu ze souboru nastat situace, kdy se při použití aplikace na některých zařízeních část importované hradlové sítě nachází mimo editační plochu.

Editační plocha bez omezených rozměrů má následující výhody:

- Rozměry editační plochy neomezují velikost sestavované hradlové sítě.
- V kombinaci s funkcí pro přiblížení a oddálení editační plochy může uživatel zobrazit vždy tu část plochy, která je pro něj v danou chvíli relevantní a umožňuje tak pohodlné zobrazení malých i rozsáhlých logických obvodů.

Nevýhody neomezené editační plochy:

- Neomezená editační plocha může vést ke snížení přehledu uživatele o všech umístěných komponentách, neboť některé můžou být skryté mimo aktuálně zobrazovaný výřez aplikace. S využitím funkce pro přiblížení a oddálení editační plochy by však tento problém neměl nastávat příliš často.
- Neomezenou editační plochu je poněkud složitější implementovat než editační plochu s pevně danými rozměry.

Rozhodli jsme se pro implementaci neomezené editační plochy, neboť výhody této varianty výrazně převyšují její nevýhody.

Propojování hradel pomocí vodičů

Uživatel vytváří hradlové sítě propojováním jednotlivých hradel vodiči. Realizaci procesu propojení dvou hradel lze pojmout dvěma odlišnými způsoby. První způsob nechává volbu, kudy povede vodič, čistě na uživateli, který kromě propojovaných pinů určí i přesnou dráhu vodiče. Výhodou manuálního propojování pinů je, že uživatel má naprostou kontrolu nad tím, kudy který vodič povede. Manuální propojování pinů je však pro uživatele výrazně pracnější než algoritmické.

Druhý způsob je založen na rozhraní, ve kterém uživatel pouze zvolí piny, které chce propojit, a algoritmus najde co nejlepší cestu. Největší výhodou tohoto

přístupu je zrychlení a usnadnění práce s editorem, neboť se uživatel otázkou dráhy pro vodič nemusí zabývat. Nevýhodou je pak především to, že lze obtížně formalizovat definici nejlepší cesty a s tím spojená implementace algoritmu, který má nejlepší cestu mezi dvěma piny nalézt.

Rozhodli jsme se pro druhé řešení. Obtížný byl výběr měřítka kvality cesty. Nebylo možno hledat nejkratší cestu mezi zvolenými piny, neboť by v některých případech docházelo k nepřehlednému křížení vodičů. Jako možné řešení se nabízela možnost nalézt jiné měřítko, které by bylo možno použít jako metriku v některém z algoritmů pro hledání nejkratší cesty, případně využít některý z pokročilých algoritmů používaných pro kreslení grafů.

Vzhledem ke komplexitě problematiky kreslení grafů jsme se rozhodli implementovat modifikaci algoritmu A* [23] a jako metriku jsme zvolili součet počtu ohybů vodiče s počtem míst, ve kterých se vodič kříží s jiným. Podrobněji se volbě tohoto algoritmu věnujeme v sekci *3.1 Hledání nejlepší cesty pro vodiče*.

2.1.2 Simulace hradlových sítí

Kromě sestavování hradlových sítí je ještě potřeba, aby bylo možno sestavenou síť odsimulovat na zadaných vstupech a zobrazit výstup ze sítě. K tomu by byly vhodné další dvě komponenty — vstupní prvek, pomocí kterého by bylo možno nastavit binární vstup do nějaké komponenty, a výstupní prvek, který by dokázal zvýraznit výstup z vybrané komponenty.

Důležitý byl výběr samotného simulačního algoritmu, který lze pojmout dvěma odlišnými způsoby. První způsob spočívá v synchronní simulaci jednotlivých komponent na editační ploše — u všech jsou synchronně v pravidelných intervalech vyhodnoceny hodnoty vstupních pinů a v závislosti na jejich hodnotách jsou nastaveny hodnoty výstupních pinů. Výhody tohoto simulačního algoritmu jsou následující:

- Algoritmus je vhodný pro simulaci logických obvodů, ve kterých je důležité zpoždění komponent. Je tedy vhodný pro realističtější simulaci komponent pro sestavování složitých digitálních obvodů.
- Algoritmus striktně pracuje s hodnotami 0 a 1, není u něj třeba zvlášť ošetřovat detekci cyklů a oscilace.

Tento algoritmus má i některé nevýhody:

- Neboť veškeré vyhodnocování algoritmu probíhá na úrovni jednotlivých komponent, v každé iteraci algoritmu jsou simulovány všechny komponenty na editační ploše, tedy i takové, jejichž vstupní stavy se nezměnily. Algoritmus tedy může být v případě velkého množství komponent na editační ploše výpočetně náročný.

Druhý algoritmus je založený na prohledávání hradlové sítě od pinu, ve kterém proběhla první změna stavu. Výhody tohoto algoritmu jsou následují:

- Algoritmus simuluje pouze ty komponenty, v nichž došlo ke změně stavu. V případě velkého množství komponent na editační ploše může tento přístup vést k menším nárokům na výpočetní kapacitu. Příkladem může být situace, ve které se na editační ploše nachází více hradlových sítí, které nejsou navzájem propojeny vodiči.

- Algoritmus musí pro úspěšné doběhnutí detekovat cykly v simulované síti. Detekce cyklů lze využít k nalezení oscilace, kterou je pak možno zobrazit jako samostatný stav.

S tímto algoritmem jsou spojené i některé nevýhody:

- Algoritmus není vhodný pro simulaci obvodů, u kterých je důležité zpoždění komponent.
- Tento algoritmus je implementačně náročnější než algoritmus simulující síť na úrovni jednotlivých komponent.

Rozhodli jsme se pro simulaci hradlové sítě pomocí algoritmu založeném na prohledávání do šířky. Důvodem k tomuto rozhodnutí bylo zaměření na výukový aspekt aplikace. U algoritmu lze využít detekci oscilace (který je v aplikaci zobrazován jako samostatný stav), je výpočetně méně náročný a neboť má aplikace sloužit především pro práci s jednoduššími obvody a nikoli pro realistickou simulaci komponent v sítích pracujících s jejich zpožděním, nevýhody tohoto algoritmu byly jasně převáženy jeho výhodami. Jeho implementace je podrobněji popsána v sekci *3.2 Simulace hradlové sítě*.

Kromě oscilace jsme se rozhodli implementovat ještě čtvrtý stav, který značí neznámou hodnotu. Díky tomuto stavu je z hradlové sítě zřejmé, která hradla jsou korektně připojena na vstupní komponenty a lze jasně rozlišit neznámý výstup od výstupní hodnoty 0.

2.1.3 Import a export hradlových sítí

Aplikace má umožnit export uživatelem vytvořené hradlové sítě do souboru. Síť z takto vytvořeného souboru má být možno také importovat zpět do aplikace. Aby byl exportní formát uživatelsky přívětivý, bylo nutno pro export hradlových sítí využít některého z textových formátů. Díky tomu může například uživatel snadno zkopírovat exportovanou síť do schránky či v hradlové síti provádět změny přímo úpravou hodnot v textovém souboru. Místo textových souborů bylo možno využít binární reprezentaci hradlové sítě ve vlastním formátu. Jedinou výhodou této varianty by však byla menší velikost souborů se sítěmi. Vzhledem k výhodám textové reprezentace tedy využití binárních souborů nepřipadalo v úvahu.

Podstatný byl výběr formátu textového dokumentu. Vzhledem ke struktuře dat, se kterými pracujeme, bylo zřejmé, že méně strukturované formáty (například CSV) nebudou dostatečně uživatelsky přívětivé. Vhodnějším způsobem reprezentace je stromová struktura, která je bližší charakteru dat. Jednotlivé komponenty mohou být reprezentovány jako stromy obsahující strukturované informace o jejich typu, poloze, rotaci, vodičích a dalších důležitých vlastnostech.

Existuje velké množství textových formátů pro serializaci dat ve stromové struktuře, pro potřeby této aplikace jsme zvažovali jen ty, které jsou nejčastěji používané pro práci se strukturovanými daty ve webovém prostředí. Prvním zvažovaným formátem byl standard XML [15]. Výhody tohoto formátu jsou následující:

- Formát XML je velmi rozšířený a je považován za jeden ze standardních formátů pro práci se strukturovanými daty [17; 18]. Díky této skutečnosti

existuje velké množství nástrojů pro práci s tímto formátem, které může uživatel využít pro editaci uložených hradlových sítí.

- Vhodným výběrem struktury dokumentu a názvů datových položek lze dosáhnout toho, že je obsah soubor srozumitelný i běžnému uživateli.

S formátem XML jsou spojené i některé nevýhody:

- Práce s daty ve formátu XML v aplikaci napsané v jazyce JavaScript je poněkud složitější, než práce s daty ve formátu JSON. Pro přístup k datům je nutno inicializovat DOM parser, pomocí kterého lze přistupovat k jednotlivým datovým položkám [26].

- Oproti jiným formátům je XML poněkud neefektivní, neboť u neprázdných datových položek je jejich název opakován v otevírací i zavírací značce. Tato vlastnost vede ke zbytečně velkým souborům.

Druhým zvažovaným formátem byl standard JSON [5], který je podobně jako formát XML založen na stromové struktuře dokumentu, odlišuje se však od něj jednodušší a úspornější syntaxí. Jeho výhody jsou následující:

- Vhodným výběrem struktury dokumentu a názvů datových položek lze dosáhnout toho, že je obsah soubor srozumitelný i běžnému uživateli.

- Práce s formátem JSON je v jazyce JavaScript snadnější než s formátem XML. K převádění dat mezi textovou (JSON) a objektovou reprezentací lze použít jednoduchých nativních funkcí jazyka [8].

- Syntaxe formátu JSON je úspornější než syntaxe formátu XML.

Nevýhodou formátu JSON je jeho menší podpora mimo webové aplikace. Pokud by tedy uživatel chtěl použít pokročilé nástroje pro práci se soubory hradlových sítí, formát XML by byl vhodnější.

Třetím zvažovaným formátem pro serializaci dat byl standard YAML [14]. Oproti formátu JSON, který pro vyznačování jednotlivých datových položek používá složené závorky, YAML pro značení úrovně zanoření ve stromě používá předsazení mezerami. Jeho výhody jsou následující:

- Vhodným výběrem struktury dokumentu a názvů datových položek lze dosáhnout toho, že je obsah soubor srozumitelný i běžnému uživateli.

- Syntaxe formátu YAML je úspornější než syntaxe formátu XML.

Některým uživatelům se může použití předsazení pro serializaci stromové struktury jevit méně intuitivní než použití viditelných znaků.

Poslední variantou byla možnost nevyužít existujících textových formátů a vytvořit vlastní formát, který by přesně vyhovoval potřebám aplikace. Tento přístup však skýtá příliš mnoho nevýhod. Pro načtení a generování souborů ve vlastním textovém formátu by bylo nezbytné implementovat vlastní parser a generátor. Vlastní formát by navíc neměl mimo naši aplikaci žádnou podporu, uživatel by tedy nemohl například využít zvýrazňování syntaxe v editoru či validačních nástrojů.

Rozhodli jsme se využít formát JSON, neboť nejvíce odpovídá potřebám aplikace. Hlavním argumentem pro výběr tohoto formátu bylo jeho snadné použití v prostředí jazyka JavaScript, stručnost a uživatelská srozumitelnost.

2.1.4 Další funkční požadavky

Významným funkčním požadavkem je nástroj pro seznámení začínajícího uživatele se základním používáním aplikace. Standardním nástrojem pro řešení takového požadavku je úvodní tutoriál, který v aplikaci částečně plní funkci uživatelské dokumentace.

V aplikaci byla využita standardní implementace tutoriálu pomocí dialogového okna, které v jednotlivých krocích uživatele seznamuje s jednotlivými funkcemi editoru.

Dalším funkčním požadavkem je knihovna hradlových sítí, ze které má být možno načítat hotové logické obvody. Uživateli by měla být k dispozici nabídka logických obvodů, které lze importovat na editační plochu. Menu sítí, které lze importovat, bylo integrováno do kontextové nabídky a pro sítě byl využit stejný formát souboru jako pro import a export uživatelsky vytvořených hradlových sítí.

2.2 Technologie pro splnění funkčních požadavků na editační plochu

Pro splnění výše popsaných požadavků bylo nutno najít nástroj, který by umožnil jejich realizaci ve webovém prostředí. Vzhledem k tomu, že celé rozhraní aplikace je spíše vektorového, než rastrového charakteru (symboly pro hradla jsou tvořeny jednoduchými tvary, hradla jsou propojována pomocí vodičů v podobě lomených čar), hledali jsme nástroj, který by nám poskytoval rozhraní pro zobrazování interaktivní vektorové grafiky na webové stránkce.

Knihoven, které mají usnadnit práci s vektorovou grafikou ve webovém prostředí, existuje poměrně velké množství, žádná však dostatečně nevyhovovala našim potřebám a rozhodli jsme se tedy rozhraní pro práci s vektorovou grafikou implementovat sami. V následujícím textu projdeme knihovny, které jsme v rané fázi projektu vyzkoušeli, nebo o jejich vyzkoušení uvažovali. Při rešerši byl kladen důraz na úplnost knihovny — pro pohodlné použití by měla umožnit práci se základními tvary, import grafiky ze souborů (pro načítání symbolů hradel) a rozhraní pro tvorbu interaktivního prostředí, tedy podporu naslouchání uživatelským akcím (kliknutí, tažení myši, naslouchání událostem klávesnice) — a možnost jejího použití v open source projektu (knihovna by tedy měla být také zveřejněna pod open source licencí). Z knihoven objevených při rešerši zde uvádíme ty, které nejlépe odpovídaly popsaným požadavkům.

2.2.1 Přehled grafických knihoven

Raphaël[1] je kompaktní open source knihovna zaměřená na práci s vektorovu grafikou na webových stránkách. Programátor může jejím prostřednictvím vytvořit plochu, na které pak skládáním jednochých objektů vytváří interaktivní aplikaci. Jako jediná ze zmiňovaných knihoven dokáže grafiku korektně zobrazovat i v poměrně starých verzích prohlížečů (Firefox od verze 3, Internet Explorer od verze 6). Úroveň práce s vektorovou grafikou probíhá na velmi nízké úrovni a cílem knihovny bylo nabídnout vyšší podporu vektorové grafiky v prohlížečích

[1] Domovská stránka knihovny Raphaël: `http://dmitrybaranovskiy.github.io/raphael/`

a zároveň poskytnout jednotné rozhraní napříč prohlížeči [13]. Vzhledem k tomu, že je v současnosti podpora SVG v prohlížečích dostatečná [30; 27] a protože Raphaël pracuje s elementy na velmi nízké úrovni, použití této knihovny by práci neulehčilo.

Snap.svg[2] je open source knihovna pro práci se SVG dokumenty vnořenými do HTML stránky, stvořená autorem výše zmiňované knihovny Raphaël. Oproti knihovně Raphaël nabízí mnohem širší škálu nástrojů, od metod, které mají zpohodlnit upravování elementů, po podporu přesouvání elementů na ploše. V rané fázi projektu se tato knihovna jevila jako nejvhodnější kandidát, při implementaci základního rozhraní se však objevily problémy se zobrazováním prvků při změně velikosti okna prohlížeče a neboť se je nepodařilo v rozumné době vyřešit, nemohli jsme tuto knihovnu použít.

Svg.js[3] je další open source knihovna, která slibuje zjednodušení práce se SVG na webu. Podobně jako Raphaël nabízí jednoduché rozhraní pro práci s elementy na nízké úrovni. Bohužel však oproti knihovně Raphaël nenabízí potřebné rozhraní pro obsluhu událostí (zejména pro tažení myší) a bylo by nutno pro implementaci uživatelské interakce použít další knihovnu.

GoJS[4] je knihovna určená pro práci s diagramy. Má poměrně pěkné aplikační rozhraní a nabízí práci s diagramy jak na vysoké úrovni (nabízí například snadný způsob propojování bodů pomocí křivek), tak na úrovni jednotlivých objektů. Bohužel však není open source a pokud by byl tento projekt používán a případně rozšiřován i mimo akademické okruhy, bylo by nutno zakoupit drahou licenci.

Vyzkoušeli jsme ještě několik dalších knihoven (*interact.js*[5], *bonsai*[6] a *Pablo*[7]), avšak všechny měly podobné nedostatky — chyběla u nich nějaká důležitá funkcionalita (například rozhraní pro obsluhu událostí, případě podpora nějakterých typů elementů, které by byly pro tvorbu editačního prostředí praktické), kterou by sice bylo v některých případech možno doplnit vytvořením pluginu k dané knihovně, ale celkově by to pouze zvýšilo pracnost projektu.

I přes rozsáhlou rešerši se nám nepodařilo najít knihovnu odpovídající stanoveným požadavkům, rozhodli jsme se proto potřebnou funkcionalitu pro práci se SVG naimplementovat sami s využitím nativní podpory SVG dokumentů v prohlížeči a pomocí knihovny *jQuery*[8].

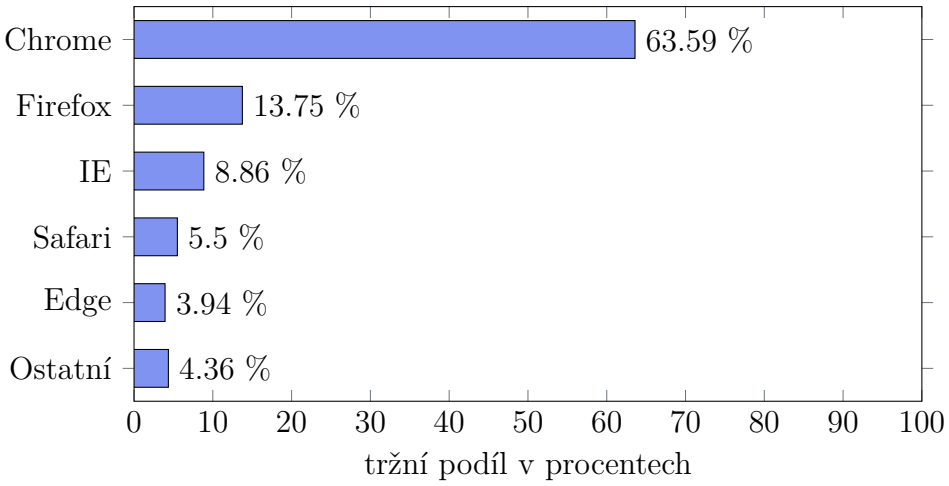
2.3 Nefunkční požadavky

2.3.1 Podpora aplikace ve webových prohlížečích

Aplikace by měla být použitelná v nejpoužívanějších webových prohlížečích. Vzhledem k velkému množství různých prohlížečů bylo nutno rozhodnout, se kterými prohlížeči by měla být aplikace kompatibilní.

Pro získání přehledu o podílu jednotlivých prohlížečů na trhu jsme využili dat

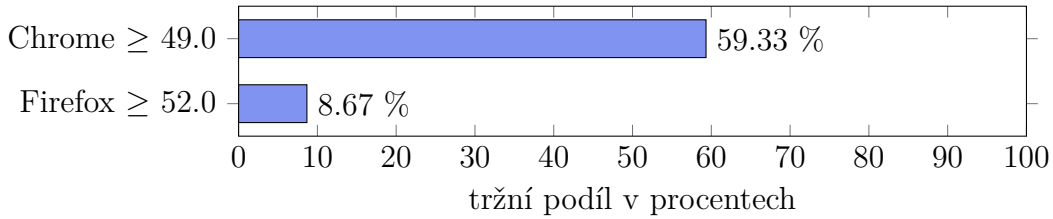
[2]Domovská stránka knihovny Snap.svg: `http://snapsvg.io/`
[3]Domovská stránka knihovny Svg.js: `http://svgjs.com/`
[4]Domovská stránka knihovny GoJS: `https://gojs.net/`
[5]Domovská stránka knihovny interact.js: `http://interactjs.io/`
[6]Domovská stránka knihovny bonsai: `https://bonsaijs.org/`
[7]Domovská stránka knihovny Pablo: `http://pablojs.com/`
[8]Domovská stránka knihovny jQuery: `https://jquery.com/`



Obrázek 2.2: Tržní podíl prohlížečů na stolních počítačích v roce 2017 [6]

zveřejňovaných společností StatCounter [6; 7]. Ze získaných dat ilustrovaných na obrázku 2.2 vyplývá, že nejpoužívanějšími prohlížeči jsou Google Chrome a Mozilla Firefox. Rozhodli jsme se zaměřit na podporu těchto dvou prohlížečů.

Z podrobnějších dat popisujících podíl jednotlivých verzí prohlížečů vyplývá, že podporou prohlížečů Google Chrome od verze 49.0 a Mozilla Firefox od verze 52.0 dosáhneme 68% pokrytí (ilustrováno na obrázku 2.3). Vybrané minimální verze prohlížečů odpovídají nejstarším verzím, které mají na trhu podíl přesahující 2 %. Obě tyto verze již podporují moderní aplikační rozhraní používané v aplikaci a velkou část novější verze jazyka ECMAScript — ECMAScript 2015 [35]. Tato verze výrazně rozšiřuje syntax oproti předchozím verzím jazyka. Mimo jiné například umožňuje syntax tříd a dědičnosti, zjednodušený syntax funkcí a objektů, různé typy cyklů a nové datové struktury [31]. Zejména nový syntax tříd a funkcí vedl k rozhodnutí psát práci v této verzi jazyka.



Obrázek 2.3: Tržní podíl prohlížečů Google Chrome od verze 49.0 a Mozilla Firefox od verze 52.0 na stolních počítačích v roce 2017 [7]

I přes zaměření na specifické prohlížeče jsme se rozhodli dosáhnout větší zpětné kompatibility prostřednictvím kompilace kódu z modernější verze jazyka ECMAScript 2015 [3] do starší verze ECMAScript 5.1 [2], která je široce podporována i ve starších verzích prohlížečů [34]. Ke kompilaci jsme použili nástroj Babel[9], který je považován za standard pro kompilaci ECMAScriptu [4].

[9]Domovská stránka nástroje Babel: `https://babeljs.io/`

2.4 Podobné nástroje

Existuje velké množství nástrojů, které mohou sloužit k tvorbě a simulaci hradlových sítí. V této sekci jsou zmíněny ty nástroje, které se jevily jako nejvhodnější pro použití ve výuce a které byly dostupné zdarma.

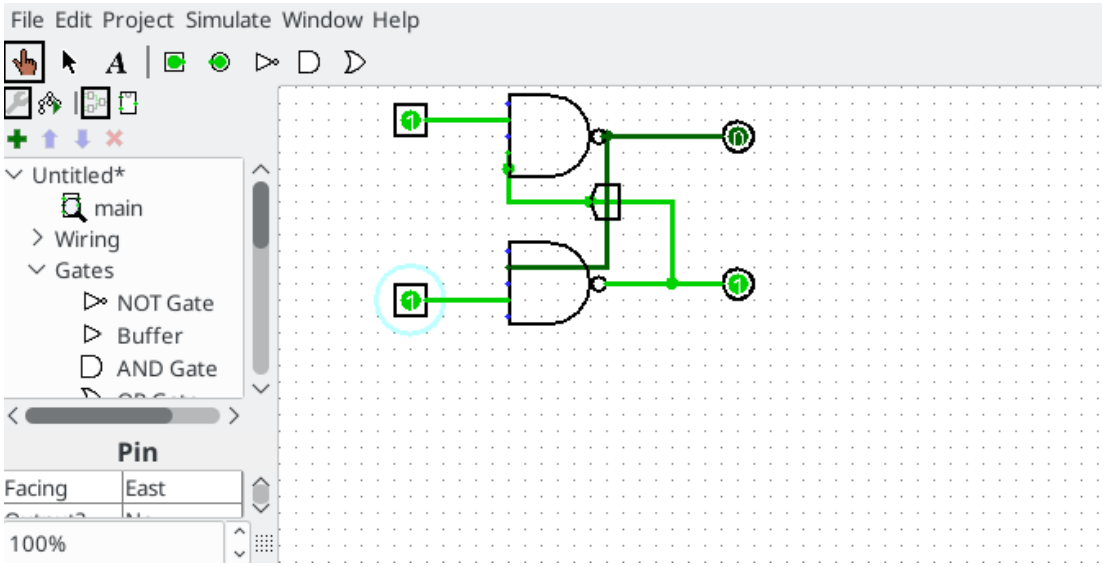
2.4.1 Logisim

Logisim[10] je pokročilý nástroj pro práci logickými obvody. Kromě hradel nabízí uživateli širokou škálu dalších komponent například pro matematické operace nad binární reprezentací přirozených čísel, paměťové prvky a širokou škálu vstupních a výstupních komponent. Umožňuje uživateli také jednoduše zabalit vytvořenou síť do jedné komponenty, se kterou lze pak dále pracovat.

K instalaci a spuštění programu stačí z webové stránky projektu stáhnout balíček `jar` a spustit jej v běhovém prostředí pro jazyk Java.

Největší nevýhodou aplikace je její nepříliš intuitivní ovládání. Uživatel musí ručně přepínat mezi editačním a simulačním módem aplikace (například přidávání a propojování prvků lze provádět jen v editačním módu, nastavování vstupních hodnot pouze v simulačním módu, ale odebírat prvky lze jak v editačním, tak i v simulačním módu aplikace). V editačním módu je občas obtížné poznat, zda je komponenta správně připojena k vodiči (připojovací piny jsou zobrazeny jen u některých komponent) a při přesouvání komponent volí program poněkud nepředvídelné cesty pro vodiče.

Aplikace navíc automaticky propojuje křížící se vodiče, takže chce-li uživatel, aby se křížící vodiče nepropojily, musí nejprve umístit speciální prvek „tunnel“ na místo budoucího křížení, pak se vodiče nepropojí. Toto chování může být žádoucí při plánování plochých spojů, avšak při výuce hradlové logiky bude spíš komplikací.



Obrázek 2.4: Klopný obvod v programu Logisim

[10] `http://www.cburch.com/logisim/index.html`

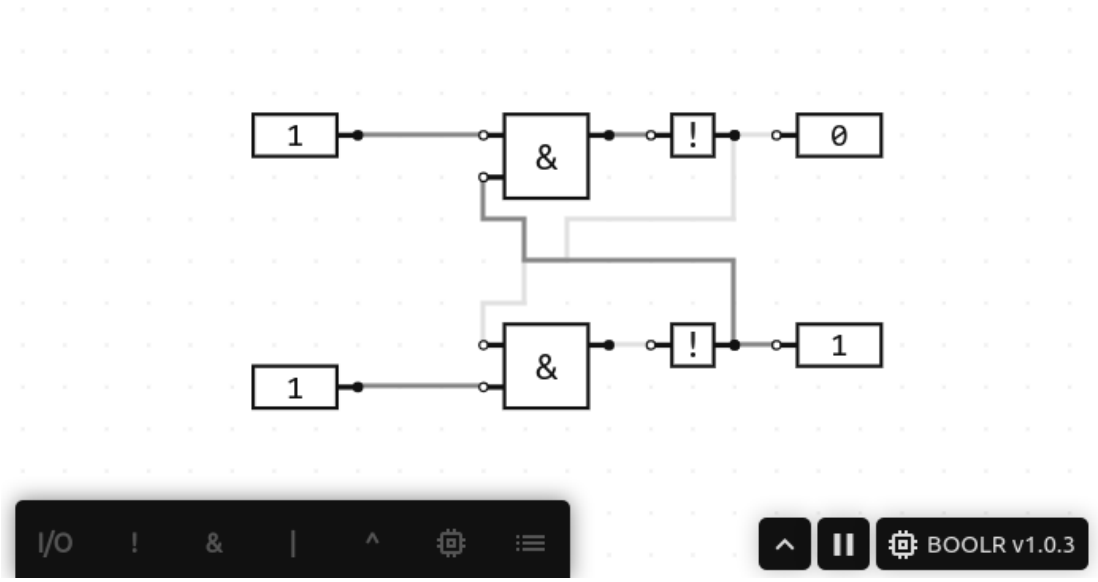
2.4.2 Boolr

Aplikace BOOLR[11] usiluje o co nejjednodušší rozhraní pro práci s logickými obvody — dokud například si uživatel nedefinuje další vlastní komponenty (sestavením ze stávajících komponent), může pracovat pouze s hradly NOT, AND, OR a XOR.

BOOLR umožňuje i ukládání sítí do souboru a rozhraní je velmi intuitivní, jen je opět ke spuštění potřeba jednoduchá uživatelská instalace (stažení archivu, jeho rozbalení a spuštění správného spustitelného souboru). Nevýhodou je, že se při přesouvání komponent aktualizuje jen poslední článek vodiče, takže pro zachování přehlednosti i jednoduchých obvodů je vhodné po přesunu komponenty vodič odebrat a opět přidat. Algoritmus pro hledání nejlepší cesty mezi komponentami navíc minimalizuje délku cesty a nebere ohled na počet ohybů vodiče, což může v některých případech vést k tomu, že i jednoduché obvody vypadají zbytečně komplikovaně.

Použití barevnějšího rozhraní (například by bylo možno použít barevného značení stavů na obvodech místo použití dvou různých odstínů šedé) by možná mohlo pomoci uživateli rychleji se zorientovat ve složitějších obvodech.

Aplikace pracuje striktně se stavy 1 a 0, takže například hradla, která nemají připojené žádné vstupy se chovají stejně jako hradla, která mají na obou vstupech stav 0. Toto rozhodnutí vývojářů vede ke drobné chybě, kdy je možno vytvořit oscilátor pomocí komponent nepřipojených k žádnému vstupu.



Obrázek 2.5: Klopný obvod v programu Boolr

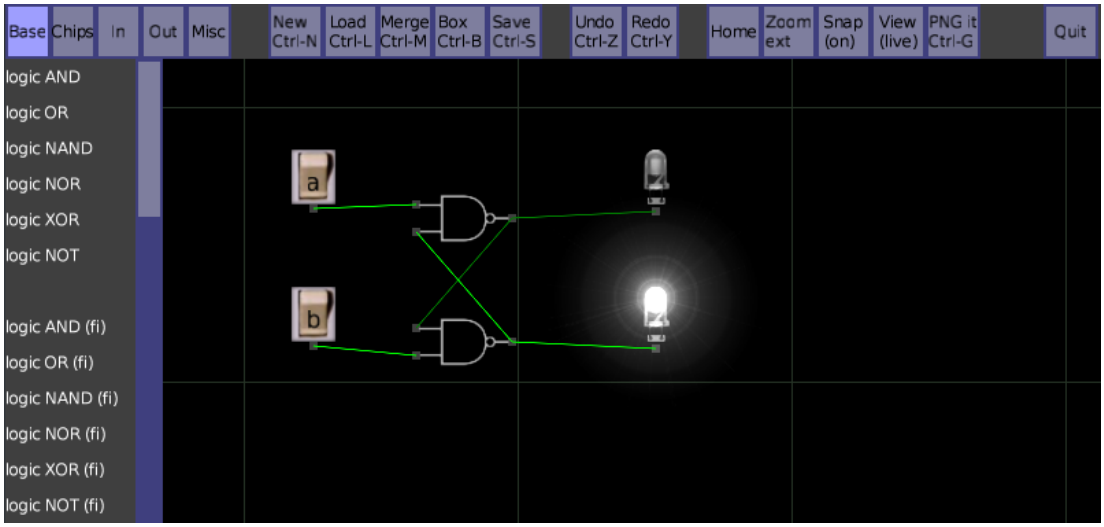
[11] http://boolr.me/

2.4.3 Atanua

Nástroj Atanua[12] vznikl původně jako komerční software, jeho autor se však později rozhodl zpřístupnit jej zdarma veřejnosti [24]. Ke spuštění je třeba podobně jako u výše zmíněných nástrojů stáhnout archiv, rozbalit jej a spustit správný soubor.

Aplikace nabízí širokou škálu hradel a čipů, také je uživateli k dispozici poměrně velké množství různých vstupních a výstupních prvků. Ovládání však není příliš přehledné a kombinace ustálených symbolů pro některé a barevných fotorealistických ikon pro jiné prvky může vést ke zhoršení přehlednosti sestavovaného obvodu.

Další poměrně velkou nevýhodou programu, se kterou jsme se během testování setkali, je propojování komponent pomocí vodičů. Při propojení dvou pinů se mezi nimi vytvoří rovný vodič vedoucí nejkratší cestou. Podle návodu by mělo být možno poté vytvořit na vodiči kotevní body a jejich přesouváním pak nastavit cestu vodiče podle svého uvážení. Při testování programu se nám však ani po přesném následování návodu ze stránek tvůrce nepodařilo vytvořit takový kotevní bod. Také se nám nepodařilo objevit, zda jde prvky na editační ploše otáčet.



Obrázek 2.6: Klopný obvod v programu Atanua

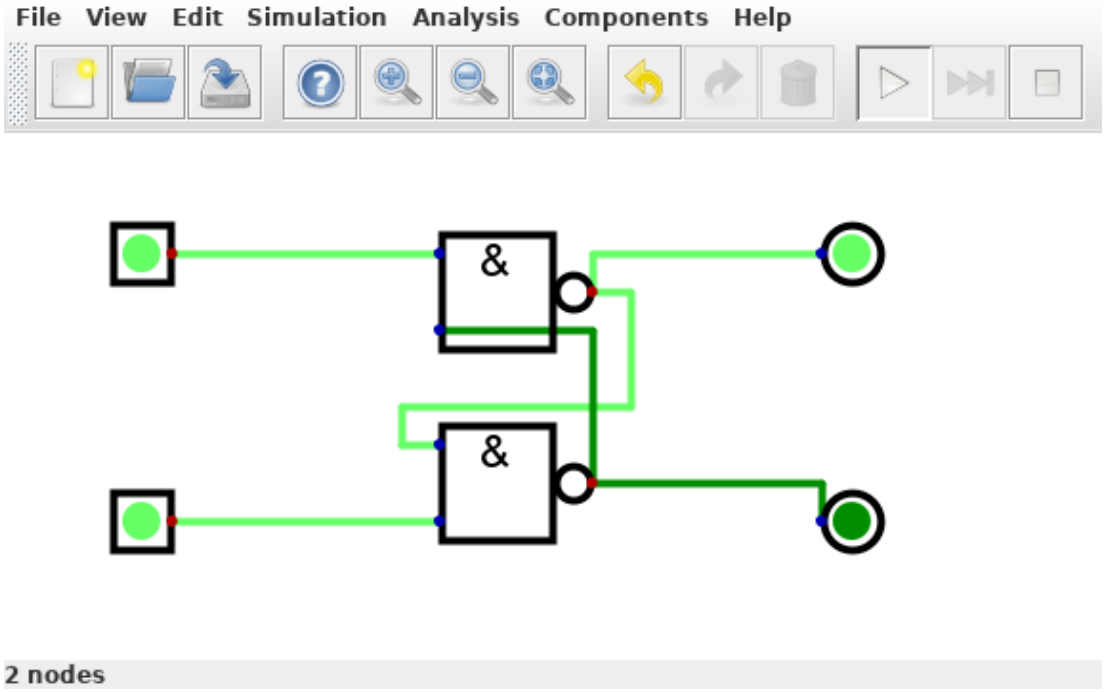
[12] http://sol.gfxile.net/atanua/index.html

2.4.4 Digital

Ač je aplikace Digital[13] v době psaní této práce stále ještě ve fázi vývoje, nabízí už všechnu funkcionalitu pro základní práci s logickými obvody. Instalace probíhá obdobně jako u výše uvedených nástrojů: Je nutno stáhnout balíček, rozbalit jej a spustit správný soubor.

Editační plocha pracuje s neviditelnou mřížkou, na kterou se zachytávají komponenty a vodiče. Uživateli je k dispozici široká škála komponent a čipů a jejich umisťování na editační plochu je intuitivní. Aplikace pracuje podobně jako *Logisim* s editačním a simulačním módem.

Některé ovládání aplikace je však poněkud zmatečné. Uživatel může například přidat vodič buď postupným kliknutím na dva piny, které si přeje propojit, nebo může vodič vytvořit kliknutím a tažením po ploše. Podobně jako u aplikace Logisim však není občas jasné, zda je vodič správně připojen ke komponentě. V některých případech je použito trochu nestandardní ovládání — například komponenty se přesouvají tak, že na ně uživatel musí nejprve kliknout, pak komponenta následuje pohyby myši, dokud uživatel znovu neklikne. Standardnější přístup by však byl, že se komponentou pohybuje držením tlačítka myši a tažením.



Obrázek 2.7: Klopný obvod v programu Digital

[13] https://github.com/hneemann/Digital

2.4.5 Simulator.io

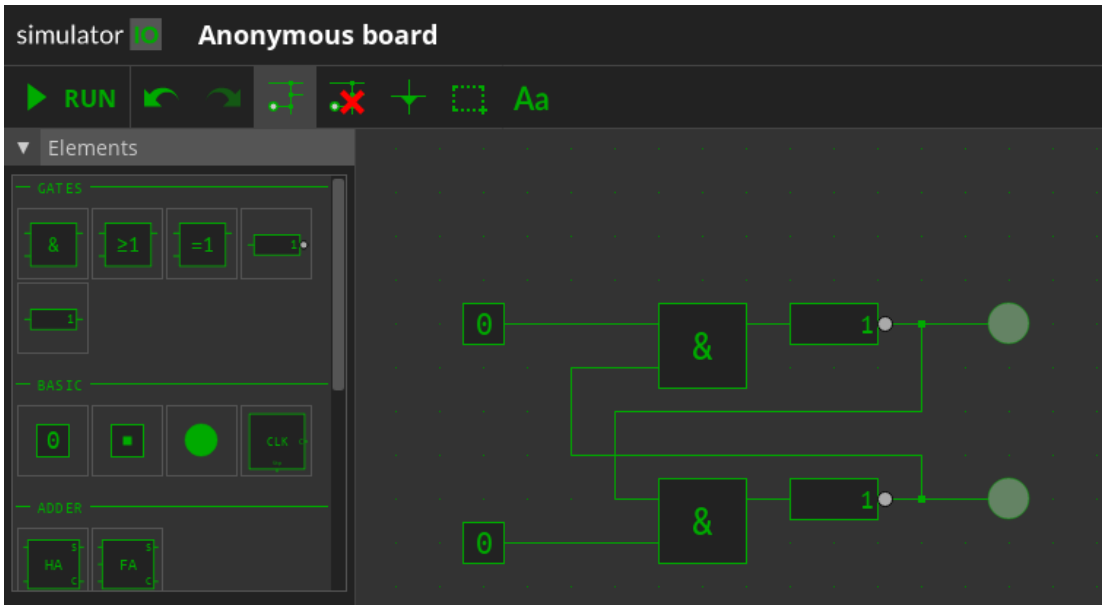
Simulator.io[14] je webový nástroj, tedy pro jeho spuštění stačí uživateli navštívit jeho webovou adresu. Nabízí práci s hradly AND, OR, XOR a NOT a několika čipy (multiplexor a demultiplexor, několik typů paměťových buněk...).

Práce s aplikací je poměrně intuitivní, přidávání jednotlivých komponent je jednoduché a propojování pomocí vodičů funguje pěkně. Uživatel sice musí cestu vodiče navrhnout sám, ale ovládání je jednoduché, takže to není na obtíž.

Poněkud zmatečná je práce s editační plochou. Jediný způsob, jakým lze přesunout výřez editační plochy, je kolečkem myši plochu oddálit a přiblížit ji jinde. Logický obvod také nejde jednoduše exportovat do souboru, je však možno vytvořit tzv. snapshot, tedy unikátní URL, která uživateli otevře simulátor s uloženým obvodem.

V editačním módu uživatel přepíná mezi dvěma nástroji, z nichž jeden slouží k přidávání komponent a vodičů na editační plochu, druhý pak slouží k jejich odebírání. Při testování této aplikace se však nutnost neustále přepínat mezi těmito nástroji jevila zbytečně pracná.

Podobně jako aplikace Boolr, i simulator.io pracuje výhradně se stavy 1 a 0 a tedy lze vytvořit oscilující obvod bez vstupních prvků.



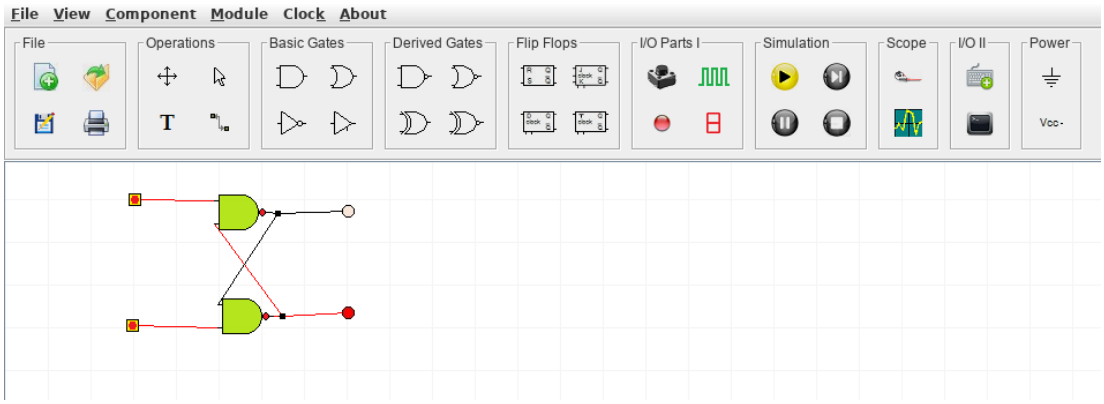
Obrázek 2.8: Klopný obvod v programu Simulator.io

[14] https://simulator.io/

2.4.6 Digital Logic Design

Aplikace Digital Logic Design[15] (DLD) je další nástroj, k jehož instalaci je nutno stáhnout archiv, rozbalit jej a pak pustit balíček `.jar` v běhovém prostředí Javy. Nabízí poměrně velké množství složitějších komponent, od sčítaček po generátor náhodných čísel. Dále je možno vytvořený obvod zabalit do komponenty a dále s ním pracovat jako s jedním celkem.

Uživatelské rozhraní mi však připadalo velmi zmatečné. Ač je například na pozadí editoru zobrazena mřížka, prvky se na ni nezachytávají a je tedy na uživateli, jak přesně dokáže komponenty umisťovat. Pro propojování komponent je nutno přepnout na jiný nástroj, pak je potřeba s přesností na několik pixelů vybrat pin, ke kterému si uživatel přeje vodič připojit.



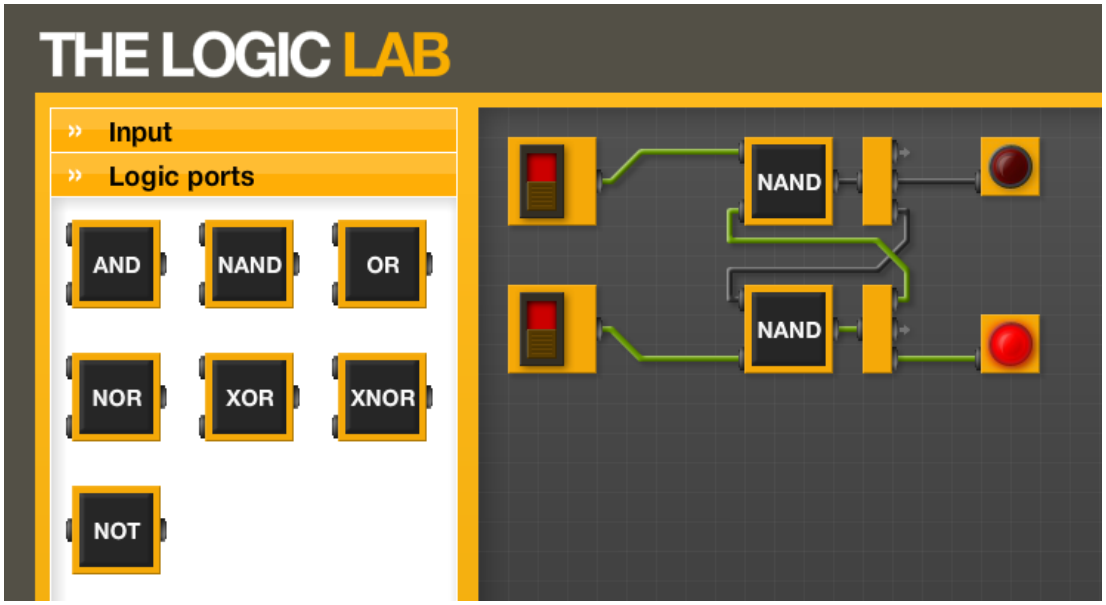
Obrázek 2.9: Klopný obvod v programu Digital Logic Design

2.4.7 The Logic Lab

The Logic Lab[16] je webová aplikace, která nabízí základní sadu hradel a vstupních a výstupních prvků a tři komponenty s funkcionalitou klopného obvodu. Editační rozhraní je přehledné, ale dosti omezené. Prvky není možno rotovat editační plocha je omezena velikostí prohlížeče a pro připojení více vodičů k výstupu nějaké komponenty je nutno použít speciální komponentu, která rozdistribuuje vstupní signál do tří výstupních pinů.

Velkou nevýhodou aplikace je její běhové prostředí vyžadující Flash plugin, jehož podpora v prohlížečích bude kvůli velkému množství bezpečnostních chyb [36] a ukončení vývoje této technologie [11] dlouhodobě klesat [32; 25].

[15] `http://www.digitalcircuitdesign.com/`
[16] `http://www.neuroproductions.be/logic-lab/`



Obrázek 2.10: Klopný obvod v programu The Logic Lab

2.4.8 Shrnutí rešerše podobných nástrojů

Instalace všech vyzkoušených nástrojů proběhla bez komplikací. Ze zkušeností z testovaných nástrojů plyne, že nejčastějším nedostatkem je špatná ovladatelnost a neintuitivní provedení uživatelského rozhraní.

Bylo možno pozorovat dva přístupy k simulaci. První skupina aplikací jasně oddělovala editační a simulační mód aplikace, druhá skupina pak prováděla simulaci přímo při práci se sítí. Při testování aplikací působil druhý přístup intuitivněji, bude tedy vhodné se tímto přístupem inspirovat a použít jej i při vývoji vlastní aplikace.

Nejvíce inspirace lze čerpat z nástroje Boolr, který implementuje nekonečnou editační plochu, výše zmiňovanou průběžnou simulaci a ukládání a načítání hradlových sítí prostřednictvím textových souborů. Co se přehlednosti sítí týče, i přes nestandardní zobrazování komponent nejvíce inspirace poskytuje aplikace The Logic Lab, jejíž systém pro zobrazování vodičů velmi ulehčuje práci — vodiče jsou dostatečně široké, jsou vedeny přehledně po mřížce a jejich hodnoty jsou vyznačeny dostatečně odlišnými barvami, aby byl uživateli na první pohled jasný stav hradlové sítě.

Z rešerše zároveň plyne, že u všech nástrojů existuje prostor pro zlepšení a na základě získaných zkušeností jsme byli schopni vytvořit vlastní nástroj, který ještě lépe odpovídal potřebám snadného použití ve výuce.

3. Popis řešení

V této kapitole se zabýváme realizací požadavků popsaných v sekci *2 Analýza problému*. Věnujeme se výběru vhodné technologie pro implementaci grafické stránky editoru a dalším zásadním rozhodnutím, která padla při návrhu aplikace.

Rozhodnutí vytvářet webovou aplikaci jasně určuje technologie, se kterými budeme pracovat — aplikace bude tvořena kódem napsaným v jazycích JavaScript, HTML a CSS.

Pokud bude navíc navržena tak, že se veškerá logika odehrává v prohlížeči uživatele a není třeba po načtení stránky dále komunikovat se serverem, bude možno aplikaci spouštět poměrně jednoduše i bez připojení k internetu.

3.1 Hledání nejlepší cesty pro vodiče

Pokaždé, kdy dojde v hradlové síti k nějaké změně spojené s vodiči, tedy ve chvíli, kdy uživatel určí, které dva piny chce propojit pomocí vodiče či pokud uživatel přesune či rotuje hradlo připojené k nějakému vodiči, aplikace se pokusí najít pro dotyčný vodič co nejpřehlednější cestu.

Pro implementaci algoritmu hledajícího co nejpřehlednější cestu bylo nutno specifikovat definici přehledné cesty. Je zřejmé, že nebylo možno hledat nejkratší cestu mezi zvolenými piny, neboť v některých případech dochází k nepřehlednému křížení vodičů.

Z analýzy problému již víme, že jednou z možností bylo využít některý z pokročilých algoritmů používaných pro kreslení grafů, tato možnost však byla zamítnuta pro svou složitost. Druhá možnost spočívala v nalezení vlastní metriky pro hledání nejkratší cesty.

Pro přehlednost vodičů je podstatné, aby byla jejich dráha co nejjednodušší a aby se vodiče příliš nekřížily. Rozhodli jsme se proto metriku pro ohodnocení cesty postavit na počtu ohybů vodiče a počtu jeho křížení s jinými vodiči. Důležitá byla též volba váhy těchto dvou faktorů na výslednou cenu nalezené cesty. Experimentálně jsme dospěli k závěru, že použitý algoritmus dosahuje nejlepších výsledků, pokud počet ohybů vodiče a počet křížení mají stejnou váhu.

Podstatným rozhodnutím byla volba vhodného algoritmu, který bude společně se zvolenou metrikou použit pro hledání nejkratší cesty. Rozhodli jsme se pro modifikaci algoritmu A* [23]. Tento algoritmus je založený na principech Dijkstrova algoritmu [16], který použitím odhadu vzdálenosti ke konkrétnímu cílovému bodu rozšiřuje o heuristiku převzatou z algoritmu pro uspořádané prohledávání, umožňující rychlejší nalezení nejkratší cesty z počátečního bodu do specifikovaného cílového bodu [29].

K výběru tohoto algoritmu vedlo několik faktorů. A* je pro vyhledávání nejkratší cesty v mřížce efektivnější, než algoritmus pro uspořádané prohledávání či Dijkstrův algoritmus a je nezřídka používán i v praxi [29; 19]. Podstatnou výhodou algoritmu A* je skutečnost, že k nalezení nejvýhodnější cesty lze použít dvou různých metrik. Použitím manhattanské metriky pro ohodnocení odhadu vzdálenosti z otevřeného bodu do cíle a použitím vlastní metriky pro ohodnocení nalezené cesty z počátečního bodu do otevřeného bodu dokázal algoritmus rychle nalézt přehlednou cestu mezi zadanými piny.

Protože používáme algoritmus v poměrně řídce zaplněné mřížce, provedli další jsme modifikaci algoritmu, která ve fázi objevování nových bodů přidá do fronty kromě přímých sousedů zkoumaného bodu až 50 neobsazených bodů v každém směru. Tato úprava vedla k rychlejšímu nalezení přehledné cesty zejména pro delší vodiče v řídce zaplněné editační ploše.

vstup: *start*, *end*, *blockedNodes*, *inconvenientNodes*
výstup: *posloupnost bodů*

```
1  nechť openNodes a closedNodes jsou prázdné množiny bodů
2  nechť distance(a, b) je součet vzdáleností bodů a a b na osách x a y
3  nechť fScore a gScore jsou mapování bodů do kladných reálných čísel
4  fScore(start) := distance(start, end)
5  openNodes := openNodes ∪ {start}
6  dokud |openNodes| > 0 opakuj
7      nechť C je takový bod z openNodes, že má nejmenší fScore
8      openNodes := openNodes \ {C}
9      pokud C = end
10         výstup: sestav cestu, vrať ji na výstup
11     closedNodes := closedNodes ∪ {C}
12     pro ∀S ∈ {nahoru, doprava, dolů, doleva} opakuj
13         N := C
14         pro ∀i ∈ {0, 1, ..., 50} opakuj
15             posuň N o 1 směrem S
16             pokud N ∈ blockedNodes
17                 break
18             pokud N ∈ closedNodes
19                 continue
20             // potrestej ohyb vodiče přičtením 1 ke skóre
21             newGScore := gScore(C) + 1
22             pokud N ∈ inconvenientNodes
23                 // potrestej křížení vodičů přičtením 1 ke skóre
24                 newGScore := newGScore + 1
25             pokud newGScore >= gScore(N)
26                 continue
27             pokud N ∉ openNodes
28                 fScore(N) := gScore(N) + distance(N, end)
29                 openNodes := openNodes ∪ {N}
30 chyba: cesta nenalezena
```

Algoritmus 1: Hledání nejlepší cesty pro vodič

Výsledný algoritmus očekává na vstupu souřadnice pinů, které má propojit (v pseudokódu jsou značeny *start* a *end*), množinu bodů, přes které nesmí vést cesta (*blockedNodes* — to jsou body obsazené hradly a jinými komponentami) a množinu bodů přes které sice může vést cesta, ale není to vhodné

(*inconvenientNodes* — to jsou body, přes které už vede nějaký vodič). Výstupem algoritmu je pak v případě úspěchu posloupnost bodů definujících cestu. Kontrolu, zda není bod obsazený komponentou, by bylo možno provést též ověřením pozice bodu v seznamu obdélníků obsazených komponentami. Body obsazené nějakou komponentou však netvoří přesný obdélník, k této metodě by tedy bylo nutno připojit i množinu výjimek, tedy bodů, které nejsou obsazené, přestože se nacházejí v nějakém z obdélníků, či které naopak jsou obsazené, ač se v žádném obsazeném obdélníku nenacházejí. V případě hradlových sítí s velkým množstvím komponent by toto vylepšení mohlo přinést zlepšení výkonu aplikace.

Algoritmus pracuje se dvěma mapováními bodů na vzdálenost, standardně označovanými $fScore$ a $gScore$. Mapování $gScore$ slouží pro ukládání vzdálenosti k výchozímu bodu, kde je jako vzdálenost brán počet ohybů vodiče. Mapování $fScore$ pak odpovídá odhadu vzdálenosti z výchozího do cílového bodu za předpokladu, že by cesta vedla před daný bod. Hodnota $fScore$ pro nějaký bod a odpovídá součtu $gScore(a) + distance(a, end)$, kde $gScore(a)$ je počet ohybů vodiče z výchozího bodu do a a $distance(a, end)$ je manhattanská vzdálenost (součet vodorovné a svislé vzdálenosti) mezi bodem a a cílovým bodem.

Algoritmus 1 popisuje hledání cesty na vysoké úrovni, v reálné implementaci bylo navíc potřeba vyřešit rychlý přístup k *openNodes* kombinací prioritní fronty a struktury `Set` a zpětné sestavení cesty po dosažení cílového bodu. Toho lze dosáhnout pomocí mapy, do které se při objevování nových bodů ukládá, ze kterého bodu byly objeveny.

Další alternativou k tomuto řešení je optimalizace cest všech existujících vodičů v rámci jednoho algoritmu. Tato alternativa by přinesla výkonnostní zlepšení při importu rozsáhlých sítí a při vhodné implementaci by dokázala nalézt přehlednější rozvržení vodičů na editační ploše. Při práci s hradlovou sítí však uživatel až na výjimečné případy upravuje polohu pouze malého množství vodičů. V těchto situacích je použitý algoritmus efektivnější. Zajímavým rozšířením aplikace může být implementace algoritmu, který by dokázal efektivně kombinovat oba přístupy, či který by využíval pokročilých algoritmů z oblasti kreslení grafů.

3.2 Simulace hradlové sítě

K tomu, aby mohla aplikace na pinech a vodičích zobrazovat správné hodnoty, slouží simulační algoritmus, který provede analýzu struktury sítě a nastaví všechny vodiče a piny na správné hodnoty vzhledem k nastavení hodnot na vstupních prvcích.

V analýze problému jsme v sekci *2.1.2 Simulace hradlových sítí* zvolili řešení založené na statické simulaci hradlové sítě prohledáváním grafu do šířky. Za kořen grafu je brán první výstupní pin, jehož stav byl změněn, a následně je změna stavu tohoto pinu ohlášena všem vodičům, které jsou k tomuto pinu připojeny, ty zas změnu stavu nahlásí vstupním pinům, ke kterým jsou připojeny. Z analýzy problému také víme, že tento algoritmus pracuje se čtyřmi stavy. Hodnoty 0 a 1 z digitální logiky jsou rošířeny o dvě další hodnoty — hodnotu značící oscilaci a o neznámou hodnotu.

Vstupní piny vyvolají zpracování stavu komponentami, kterým náleží. Tyto komponenty vypočítají stavy svých výstupních pinů. Tyto nové stavy jsou pak algoritmem zařazeny do další vlny simulace.

U této podoby algoritmu by však nastaly potíže v případě, kdy se v hradlové síti nachází cyklus. Algoritmus proto kontroluje průchod signálu a ve chvíli kdy objeví, že do nějakého výstupního pinu dorazit signál, který už v něm v průběhu simulace byl, označí tento pin jako zacyklený.

U zacykleného pinu pak algoritmus sleduje, zda se jejich hodnota v rámci cyklu mění či nikoli. Pokud ano, nastaví jeho hodnotu na oscilující, jinak je mu ponechána tatáž hodnota, která mu byla v každé iteraci cyklu nastavována. Cyklus je následně přerušen a výsledná hodnota pinu je pak propagována dále v síti.

3.3 Synchronizace stavu aplikace s uživatelským rozhraním

Existuje několik různých přístupů popisujících provázání logického stavu webové aplikace a stavu objektového modelu dokumentu (DOM), se kterým uživatel pracuje.

Jeden z krajních přístupů spočívá v uchovávání veškerého logického stavu aplikace přímo v DOM — v kontextu této aplikace by tedy například přímo u DOM objektu znázorňujícího nějaký vstupní či výstupní pin byl v nějakém parametru uveden seznam vodičů, které jsou k němu připojeny. Kód napsaný v JavaScriptu by pak při interakci uživatele s nějakým prvkem vždy nalezl v DOM správnou komponentu, zjistil její stav z parametrů a v závislosti na akci uživatele pak uložil nový stav zpět do DOM. Samotný kód v JavaScriptu by pak nepotřeboval uchovávat žádný vnitřní stav. Tento přístup může být vhodný pro aplikace malého rozměru. Vzhledem k rozsahu aplikace tedy tato varianta nepřipadala v úvahu.

Jiný přístup využívá například knihovna React[1], která používá jednosměrnou datovou vazbu — datové položky jsou navázány na komponenty ve virtuálním objektovém modelu dokumentu. Při změně hodnoty datové položky je pak změna provázané komponenty vyrenderována do reálného DOM. Místo vazby v opačném směru knihovna využívá funkcí pro obsluhu uživatelských událostí.

Odlišný přístup je použit ve frameworku AngularJS[2], využívajícím obousměrnou vazbu. Framework sleduje nejen změny vnitřního stavu aplikace, ale i změny stavu v DOM a provádí jejich vzájemnou synchronizaci.

Naše aplikace pracuje s velmi specifickým datovým modelem, jehož základem je orientovaný graf, jehož vrcholy tvoří komponenty, hrany jsou tvořeny vodiči. Každému vodiči je přiřazena jedna ze čtyř možných hodnot, vyplývající ze stuktury grafu a nastavení vstupních komponent. Samotné uživatelské rozhraní aplikace však není příliš složité. Sestává z editační plochy obsahující vizualizaci datového modelu a několika dalších poměrně jednoduchých prvků.

Datový model je v naší aplikaci ovlivňován ze strany uživatele nepřímo prostřednictvím uživatelských interakcí, vztah modelu a uživatelského rozhraní je tedy podobný přístupu využívanému v knihovně React. Neboť je však naše aplikace z hlediska uživatelského rozhraní a zobrazovaných komponent poměrně jednoduchá, rozhodli jsme se React nevyužít a vazbu mezi uživatelským rozhraním a daty implementovat ručně.

[1] Domovská stránka knihovny React: `https://reactjs.org/`

[2] Domovská stránka frameworku AngularJS: `https://angular.io/`

Každému prvku na editační ploše, se kterou může uživatel interagovat, náleží v objektovém návrhu aplikace jedna třída, popisující veškerou interaktivitu a uchovávající vnitřní stav komponenty. Jednou z datových položek této třídy je odkaz na DOM element, jehož podoba je při změně vnitřího stavu prvku aktualizována.

Vazba ve směru od uživatelského rozhraní k datům je implementována pomocí detekce uživatelských událostí. Poté, co uživatel provede interakci s nějakým prvkem, je prostřednictvím řídící třídy nalezen správný objekt, kterému přísluší DOM element, se kterým uživatel interagoval. Na základě interakce je aktualizován vnitřní stav tohoto objektu, jehož změna je pak reflektována v provázaném DOM elementu. Objekt má též možnost při zpracování uživatelské interakce o změně informovat řídící třídu, prostřednictvím které lze ovlivnit stav dalších komponent. Pokud uživatel například odstraní z editační plochy hradlo, které je připojeno k jiným komponentám pomocí vodičů, přes řídící třídu dojde i k ostranění všech vodičů připojených k ostraňovanému hradlu.

3.4 Grafická podoba aplikace

3.4.1 Grafické provedení editační plochy

Prostředí, ve kterém uživatel pracuje s hradly, by mělo být jednoduché a přehledné. Rozhodlu jsme se proto znázornit editační plochu pomocí mřížky, na kterou se komponenty i vodiče zachytávají a zlepšují tak přehlednost logického obvodu.

Součástí každé komponenty jsou vstupní či výstupní piny, které jsou znázorněny malým čtvercem zabarveným podle aktuální hodnoty pinu. Klikáním na piny je může uživatel propojovat pomocí vodičů.

Vodiče jsou zobrazovány jako linka vedoucí po hranách mřížky, každý vodič je znázorněn barvou odpovídající jeho aktuální hodnotě.

Aby uživatele neomezovala velikost editační plochy, uživatel pracuje s pohledem, který lze oddalovat, přibližovat a posouvat (a tím zvětšovat, zmenšovat a přesouvat výřez plochy). K implementaci této funkce jsme využili atribut *viewbox* v kořenovém elementu SVG dokumentu, který definuje výřez dokumentu, který je uživateli zobrazován. Uživatel pak přesouváním pohledu upravuje parametry tohoto atributu. Rozměry editační plochy jsou v kontextu této aplikace takřka neomezené [20].

3.4.2 Znázornění komponent

Zobrazení hradel by mělo být intuitivní a ideálně by použité symboly měly být standardní i mimo aplikaci. Využili jsme specifikace IEEE Std 91a-1991 [1], která popisuje dvě sady symbolů pro znázorňování logických členů — *Rectangular shape symbols* a *Distinctive shape symbols*. Symboly ze sady *Rectangular shape symbols* jsou tvořeny obdélníkem, ve kterém je zaznačena logická operace (& pro hradlo AND, ≥ 1 pro hradlo OR...). Sada *Distinctive shape symbols* definuje pro každý běžný typ hradla jednoznačný tvar. Po zkušenostech, které jsme získali průzkumem podobných aplikací (viz *2.4 Podobné nástroje*), jsme dospěli k závěru, že značení hradel jednoznačně rozlišitelnými tvary je pro potřeby editoru

vhodnější a v aplikaci jsme se tedy rozhodli použít symboly ze sady *Distinctive shape symbols* (viz přílohu *A.1 Grafické znázornění hradel v aplikaci*).

Kromě hradel existují v aplikaci ještě další komponenty: vstupní a výstupní prvek a takzvané černé skříňky. Vstupní prvek slouží k nastavení vstupních hodnot při simulaci hradlové sítě, výstupní prvek graficky zvýrazňuje hodnotu zvoleného výstupního pinu. Černé skříňky jsou bezstavové komponenty, jejichž chování je definováno pravdivostní tabulkou. Neboť ve výše uvedeném standardu nejsou tyto prvky popisovány, rozhodli jsme se zobrazit vstupní prvek jako jednoduchý spínač, výstupní prvek jako kruh vyplněný barvou odpovídající aktuálnímu stavu vstupního pinu a černou skříňku jako obdélník s krátkým textovým popiskem.

4. Uživatelská dokumentace

Hlavním prvkem celé aplikace je editační plocha znázorněná mřížkou. Na tuto plochu může uživatel umisťovat jednotlivé logické komponenty hradlové sítě a navzájem je propojovat pomocí vodičů. Komponenty je možno otáčet a přesouvat na editační ploše.

K práci s prvky slouží kontextové menu, které je spuštěno kliknutím pravým tlačítkem myši na editační plochu nebo na libovolný síťový prvek na této ploše. Menu obsahuje nabídku na přidání síťových prvků a pokud uživatel spustil nabídku kliknutím na nějaký prvek na editační ploše, je v ní navíc zobrazena volba, která umožňuje uživateli tento prvek z editační plochy odebrat.

Součástí aplikace je také knihovna logických obvodů a černých skříněk, které může uživatel načíst na editační plochu. K načítání logických obvodů slouží položka *Add a network* v kontextovém menu, k načítání černých skříněk pak položka *Add a blackbox*.

Při první návštěvě aplikace nebo po kliknutí na příslušné tlačítko v plovoucí nabídce v pravém dolním rohu aplikace se uživateli zobrazí tutoriál, který jej provede základním používáním aplikace.

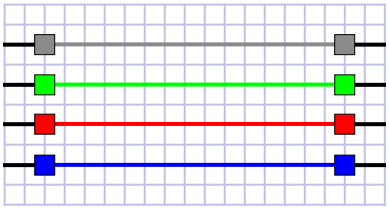
4.1 Logické stavy v aplikaci

Standardní dvouhodnotová logika pracuje s hodnotami *pravda* a *nepravda*, které jsou typicky znázorňovány jako 1 a 0. V hradlové logice je však nutno počítat i se situacemi, které se v běžné výrokové logice nevyskytují — s případem, kdy stav není definován, a s případem, kdy se v hradlové síti nachází cyklus, který může způsobit oscilaci mezi dvěma různými stavy.

Součástí každé logické komponenty je alespoň jeden vstupní či výstupní pin, který může být propojen pomocí vodiče s nějakým jiným pinem, typicky náležícím jiné komponentě. Propojení dvou pinů vodičem reprezentuje jejich logické propojení v hradlové síti, typické tedy je, že je výstupní pin nějaké komponenty pomocí vodiče propojen se vstupním pinem jiné komponenty.

Signál na vodiči nabývá takové hodnoty, jakou reprezentuje výstupní pin, ke kterému je vodič připojen. Pokud je vodič připojen k nějakému vstupnímu pinu, nastavuje jeho hodnotu tak, aby odpovídala hodnotě vodiče.

Vodiče a vstupní piny se mohou nacházet ve čtyřech logických stavech (*pravda*, *nepravda*, *nedefinovaný stav* a *oscilace mezi několika různými stavy*), každý je v aplikaci znázorněn jinou barvou (viz Obrázek 4.1). Pro potřeby tohoto textu používáme pro tyto stavy symboly 1, 0, *unk* a *osc*.



Obrázek 4.1: Piny propojené vodičem ve stavech (sestupně) *unk*, 1, 0 a *osc*

Piny a vodiče nabývají hodnot 1 a 0 výhradně v případech, pokud je tento stav jednoznačně dán strukturou sítě a nastavením vstupních hodnot. Pokud nelze pomocí aktuální kombinace vstupních hodnot a struktury sítě jednoznačně určit stav nějakého pinu či vodiče, je jeho stav nastaven na *unk*. Pokud se v síti vyskytuje cyklus, který způsobuje, že se v pinu či ve vodiči střídá několik různých hodnot, je stav tohoto pinu či vodiče nastaven na *osc* a tento nový stav je dále propagován v simulaci sítě. (Algoritmus pro simulaci je podrobněji popsán v sekci *3.2 Simulace hradlové sítě.*)

4.2 Logické komponenty

Komponenty a jejich vzájemné propojení definuje chování hradlové sítě. Komponenty je možno přidávat na editační plochu pomocí kontextového menu a je možno je členit na tři druhy — hradla, vstupní a výstupní prvky a černé skříňky. Všechny tři druhy komponent ale mají společné vlastnosti — je možné je přesouvat po editační ploše, je možno jimi otáčet a navíc má každá komponenta alespoň jeden vstupní či výstupní pin, který lze využít k propojení komponenty s dalšími prvky na editační ploše pomocí vodiče.

Každé komponentě přísluší nějaká funkce, která v závislosti na kombinaci stavů na vstupních pinech (pokud nějaké má) a vnitřním stavu (v případě vstupního prvku) nastaví hodnoty svých výstupních pinů.

4.2.1 Hradla

Aplikace umožňuje pracovat se sedmi různými typy hradel. Šest z těchto hradel má dva vstupní piny, jedno má jeden vstupní pin. Jejich grafické znázornění (viz přílohu A.1) je převzato ze sady symbolů *distinctive shape* popsané ve standardu IEEE Std 91a-1991 [1].

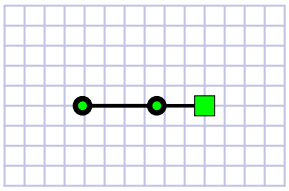
Funkce hradel vychází z příslušných funkcí z výrokové logiky, jsou však rozšířeny o hodnoty *unk* a *osc*. V Tabulce 4.1 jsou popsány funkce pro hradla se dvěma vstupními piny. Jednovstupové hradlo NOT bylo rozšířeno analogicky: pokud je vstupní hodnota 0 nebo 1, na výstupní pin je vrácena negace vstupní hodnoty, vstupní hodnoty *unk* a *osc* jsou předány na výstup.

vstup		**AND**	**OR**	**NAND**	**NOR**	**XOR**	**XNOR**
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	0
0	*unk*	0	*unk*	1	*unk*	*unk*	*unk*
0	*osc*	0	*osc*	1	*osc*	*osc*	*osc*
1	1	1	1	0	0	0	1
1	*unk*	*unk*	1	*unk*	0	*unk*	*unk*
1	*osc*	*osc*	1	*osc*	0	*osc*	*osc*
unk	*unk*	*unk*	*unk*	*unk*	*unk*	*unk*	*unk*
unk	*osc*	*unk*	*unk*	*unk*	*unk*	*unk*	*unk*

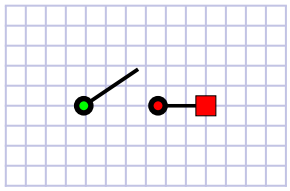
Tabulka 4.1: Pravdivostní tabulka pro hradla se dvěma vstupními piny [21]

4.2.2 Vstupní a výstupní prvky

Vstupní prvek (v aplikaci značený jako *Input box*) umožňuje uživateli nastavovat vstupní hodnoty hradlové sítě. Na rozdíl od hradel nemá žádný vstupní pin a pracuje s vnitřním stavem. Ten může nabývat stavů *on* a *off*. Výstupní pin reflektuje stav komponenty, tedy jeho hodnota nastavena na 1, pokud je vstupní prvek ve stavu *on*, a na 0, pokud je vstupní prvek ve stavu *off*.



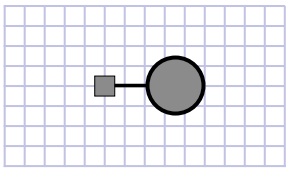
(a) Vstupní prvek ve stavu *on*



(b) Vstupní prvek ve stavu *off*

Obrázek 4.2: Vstupní prvek

Výstupní prvek (v aplikaci značený jako *Output box*) slouží ke zvýraznění stavu významných výstupních pinů v hradlové síti. Má jeden vstupní pin, jehož hodnota je pak barevně reflektována vybarvením těla výstupního prvku.



Obrázek 4.3: Výstupní prvek

4.2.3 Černé skříňky

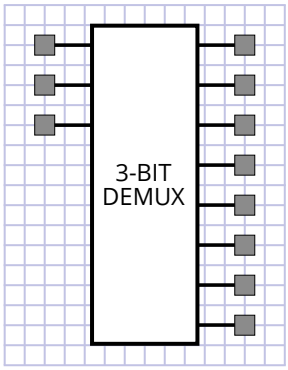
Třetím druhem logické komponenty je černá skříňka. Jde o obecnou bezstavovou komponentu, jejíž chování je definováno pravdivostní tabulkou (která může obsahovat i hodnoty *unk* a *osc*).

Aplikace nabízí v rámci knihovny sítí i sadu černých skříněk, které je možno využít při tvorbě hradlových sítí. Dále je možno definovat vlastní černé skříňky a načíst je do aplikace pomocí funkce pro import sítě (viz *4.4 Import a export hradlové sítě*).

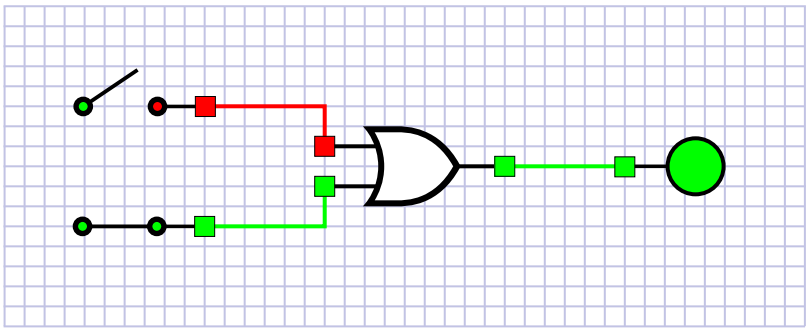
4.3 Propojování komponent pomocí vodičů

Vstupní a výstupní piny komponent je možno propojovat pomocí vodičů. Uživatel propojí dva piny tak, že postupně klikne na dva piny, které chce propojit. Mezi těmito dvěma piny pak vznikne vodič, který mezi nimi přenáší hodnotu (tedy vstupní pin nastavuje na stejnou hodnotu, jakou má výstupní pin). Zároveň je vodič znázorněn barvou odpovídající svému aktuálnímu stavu.

Při přidávání nového vodiče se aplikace pokouší nalézt co nejpřehlednější cestu mezi propojovanými piny. Algoritmus je popsán v sekci *3.1 Hledání nejlepší cesty pro vodiče*.



Obrázek 4.4: 3 bitový demultiplexor v podobě černé skříňky



Obrázek 4.5: Příklad propojení komponent pomocí vodičů na jednoduché hradlové síti sestavené ze dvou vstupních prvků, hradla OR a výstupního prvku.

4.4 Import a export hradlové sítě

Aplikace umožňuje export a import hradlových sítí ve formátu JSON[1]. Kombinaci importu a exportu lze využít také k duplikaci sestaveného obvodu při stavbě složitějších sítí a k tvorbě vlastních černých skříněk.

4.4.1 Definice vlastní komponenty pomocí pravdivostní tabulky

Neboť exportní formát pracuje s černými skříňkami jako s komponentami, u kterých zná jen počet vstupních a výstupních komponent a pravdivostní tabulku, je možno využít textové reprezentace hradlových sítí k importu vlastních černých skříněk do hradlové sítě.

Uživatel si může tedy například definovat komponentu *gateway* se dvěma vstupními a jedním výstupním pinem. Na prvním vstupním pinu bude černá skříňka očekávat hodnotu, kterou má opakovat na výstupu. Hodnota však bude předána na výstup pouze v případě, že je druhý vstupní pin nastaven na hodnotu 1. V případě, že je jeho druhý vstupní pin nastaven na hodnotu 0, na výstupním pinu bude nezávisle na prvním vstupním pinu vždy hodnota 0. Pokud je na druhém vstupním pinu hodnota *unk* nebo *osc*, hodnota výstupního pinu je *unk*.

[1]Příklad exportované sítě je uveden v příloze *A.2 Formát souboru pro textovou reprezentaci hradlové sítě.*

V následujícím příkladu je v textové reprezentaci popsána právě takováto černá skříňka. Pravdivostní tabulka je reprezentována jako pole, jehož každý prvek značí jednu kombinaci vstupních stavů a odpovídající nastavení výstupních pinů[2]. V tomto zápise používá aplikace pro stavy 0, 1, *unk*, *osc* číselné hodnoty 0, 1, 2, 3. Například první kombinace zapsaná jako $[0, 1, 0]$ značí, že pokud je první vstupní pin ve stavu 0 a druhý vstupní pin ve stavu 1, pak stav výstupního pinu má být 0.

```
{
  "boxes": [
    {
      "name": "Gateway",
      "category": "blackbox",
      "inputs": 2,
      "outputs": 1,
      "table": [
        [0, 1, 0],
        [1, 1, 1],
        [2, 1, 2],
        [3, 1, 3],
        [0, 0, 0],
        [1, 0, 0],
        [2, 0, 0],
        [3, 0, 0]
      ]
    }
  ]
}
```

Pokud se černá skříňka setká se kombinací vstupních stavů, která není v tabulce popsána, nastaví všechny výstupní piny na *unk*. V pravdivostní tabulce tedy stačí popsat jen ty kombinace vstupních stavů, při kterých má být nějaký výstupní pin nastaven na jinou hodnotu než *unk*.

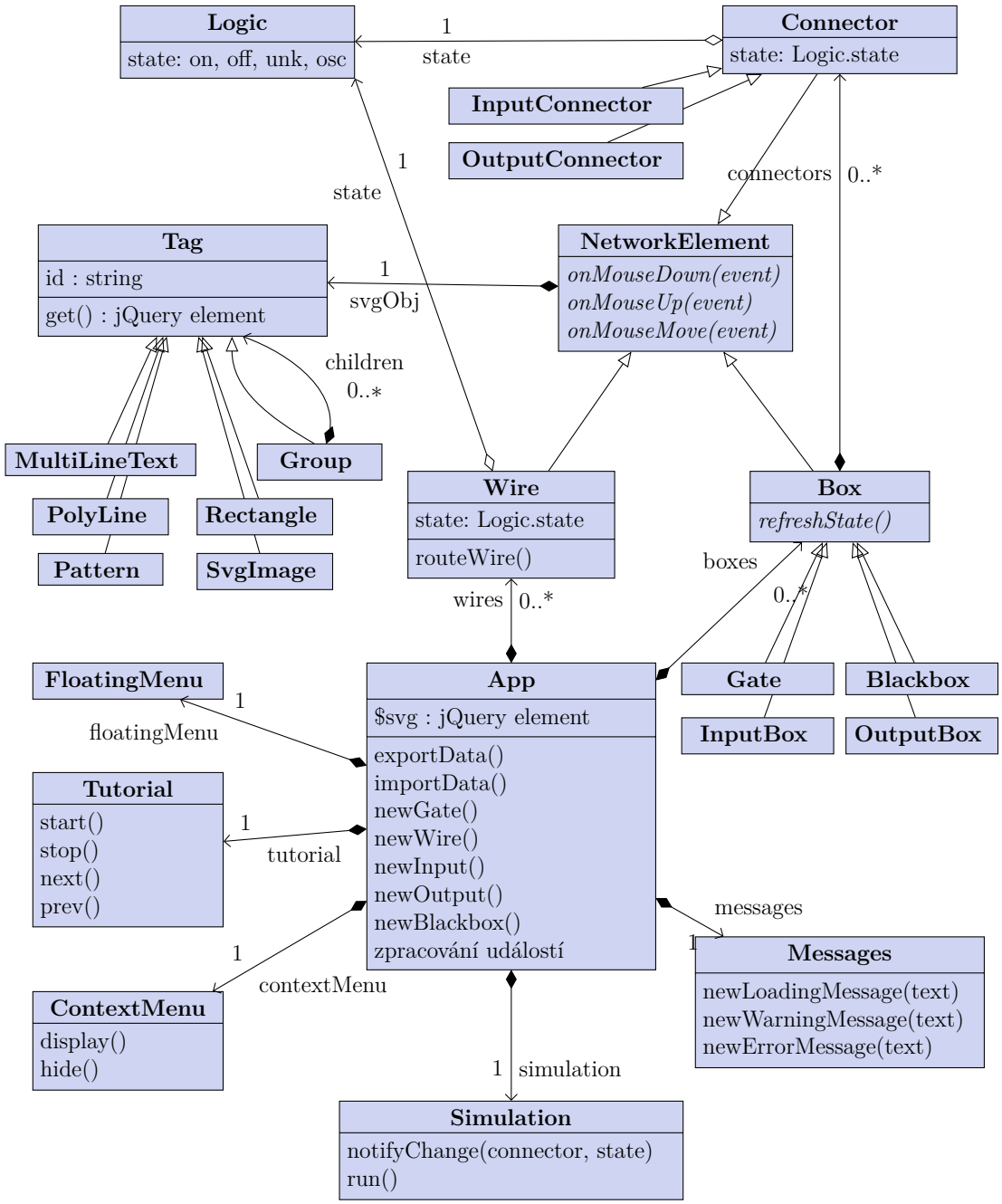
Jedním z možných vylepšení aplikace (viz *6.3 Možná vylepšení*) je rozšíření syntaxe pro definici černých skříněk pomocí pravdivostní tabulky. Možným rozšířením je například umožnění používat řetězce `"unk"` a `"osc"` místo číselných hodnot 2 a 3, možnost označit komponentu za symetrickou (jeden řádek vstupních hodnot by reprezentoval kromě zadané kombinace vstupních hodnot i kombinaci těchto vstupních hodnot v opačném pořadí) či možnost označení libovolného stavu řetězcem `"*"` nebo hodnotou `null`.

[2]Nejprve jsou uvedeny stavy vstupních pinů v pořadí odshora dolů, následně stavy výstupních pinů v pořadí odshora dolů.

5. Programátorská dokumentace

5.1 Objektový návrh aplikace

Obrázek 5.1 znázorňuje významné třídy jejich a vazby. Nejvýznamnější třídou je `App`, která propojuje jednotlivé části programu do jedné aplikace.



Obrázek 5.1: Významné třídy a vazby v objektovém návrhu aplikace

Nejvýznamnější třídy zobrazené na obrázku 5.1 popisujeme v následujícím textu.

5.1.1 Třída `App`

Třída `App` obstarává interakci mezi ostatními částmi aplikace a plní roli řídící třídy podle návrhového vzoru Mediator [22]. Obsahuje editační plochu a reference na významné objekty používané v aplikaci.

Podstatnou funkcí této třídy je obsluha událostí na editační ploše. Veškeré uživatelské operace prováděné na editační ploše jsou zachyceny v kořenovém elementu vnořeného SVG dokumentu, kde je událost zpracována. Následně je vyhodnocen správný cíl události (instance potomka třídy `NetworkElement`), se kterým uživatel chtěl interagovat, a tomu je pak událost předána. Pokud uživatelská akce navíc způsobila změnu hodnoty některého z pinů, třída `App` obstará spuštění simulace prostřednictvím třídy `Simulation`.

Po spuštění aplikace provádí `App` její inicializaci — obstará vytvoření editační plochy a inicializaci tříd tvořících uživatelské rozhraní (`FloatingMenu`, `ContextMenu`, `Messages`, a `Tutorial`, které obstarávají inicializaci a funkcionalitu plovoucího a kontextového menu, zobrazování hlášek uživateli, a případné zobrazení tutoriálu).

5.1.2 Třída `Logic`

Statická třída `Logic` obsahuje definice logických hodnot a definuje funkce používané při simulaci logických obvodů. Významná je zejména její položka `state` reprezentující výčet logických stavů. Kdekoli aplikace pracuje s logickými stavy, jsou reprezentovány jako položky výčtu `state`.

5.1.3 Třída `NetworkElement` a její potomci

Potomci třídy `NetworkElement` reprezentují jednotlivé interaktivní prvky zobrazované na editační ploše, tedy komponenty a vodiče. Významnými přímými potomky této třídy jsou `Box`, `Wire` a `Connector`.

Třída `Box` popisuje společné rozhraní pro komponenty. Jejími potomky jsou třídy reprezentující hradla (`Gate`), černé skříňky (`Blackbox`) či vstupní a výstupní prvky (`InputBox` a `OutputBox`). Každá komponenta obsahuje alespoň jeden vstupní či výstupní pin.

Třída `Connector` definuje společné rozhraní vstupních a výstupních pinů. Její potomci — `InputConnector` a `OutputConnector` pak třídu rozšiřují o vlastnosti specifické vstupním a výstupním konektorům. Vstupní konektory o změně svého stavu informují komponentu (instanci třídy `Box`), jejíž jsou součástí, výstupní komponenty o změně svého stavu informují všechny připojené vodiče.

Vodiče jsou reprezentovány instancemi třídy `Wire`. Kromě přenosu informace od výstupního pinu do vstupního pinu řídí třída i vykreslování vodiče, tedy zavolání příslušných metod pro nalezení co nejpřehlednější cesty.

V kapitole *3.3 Synchronizace stavu aplikace s uživatelským rozhraním* popisujeme vztah jednotlivých objektů a jejich reprezentace v objektovém modelu dokumentu. Jednotlivé objekty jsou reprezentovány právě instancemi potomků třídy `NetworkElement`.

Potomci třídy `NetworkElement` obsahují referenci na svou DOM reprezentaci používanou ve vnořeném SVG dokumentu, kterou udržují aktuální vzhledem ke

svému vnitřnímu stavu. K její konstrukci a následným úpravám používají třídu `Tag` či některého z jejích potomků.

5.1.4 Třída `Tag` a její potomci

Třída `Tag` a její potomci slouží ke konstrukci DOM elementů a k jejich úpravám v SVG dokumentu reprezentujícím editační plochu. Potomci této třídy reprezentují stejnojmenný element v SVG dokumentu. Například třída `Rectangle` slouží ke konstrukci obdélníků, používaných například pro vizuální reprezentaci pinů, a třída `PolyLine` slouží ke konstrukci lomených čar, používaných mimo jiné pro zobrazování vodičů.

5.1.5 Třída `Simulation`

Třída `Simulation` obsahuje simulační algoritmus a rozhraní pro jeho ovládání. Důležitými metodami jsou `run` a `notifyChange`. Metoda `run` je volána po změně struktury sítě (tedy po přidání či odebrání vodiče) či při změně hodnoty vstupní komponenty. Funkce `notifyChange` je volána v případě, kdy se změní hodnota některého výstupního pinu. Změna hodnoty tohoto výstupního pinu je pak zpracována v další vlně simulačního algoritmu.

5.2 Stručná struktura zdrojového kódu

Zdrojový kód je rozdělen do jednotlivých modulů podle jejich významu v objektové struktuře aplikace. Nejvýznamnější moduly jsou následující:

- `App` obsahuje řídící třídu aplikace
- `editorElements` poskytuje třídy reprezentující jednotlivé prvky na editační ploše, například hradla či vodiče
- `svgObjects` poskytuje třídy sloužící ke konstrukci DOM elementů v SVG dokumentu reprezentujícím editační plochu, například pro konstrukci obdélníku či textového pole
- `Simulation` obsahuje třídu pro řízení simulace logických obvodů
- `Logic` obsahuje statickou třídu s definicemi logických hodnot a funkcí používaných při simulaci logických obvodů
- `findPath` poskytuje funkci pro hledání nejpřehlednější cesty
- `routeWorker` je modul definující skript pro nalezení přehledných cest pro vodiče v sekundárním vlákně v případě importování hradlové sítě

Aplikace obsahuje další moduly, které obstarávají dílčí funkce pro chod aplikace, mezi ně patří výše zmíněné třídy tvořící uživatelské rozhraní (například kontextové menu či úvodní tutoriál), které jsou podrobně zdokumentovány ve zdrojovém kódu aplikace a ve vygenerované dokumentaci, která je součástí aplikace.

6. Vyhodnocení

Podařilo se nám vytvořit prototyp webové aplikace splňující stanovené cíle. Aplikaci je možno snadno spustit ve webovém prohlížeči a umožňuje uživateli práci se sedmi nejběžnějšími typy hradel, ze kterých může jejich vzájemným propojováním pomocí vodičů vytvořit hradlovou síť. Aplikace dokáže takto sestavenou síť odsimulovat a zobrazit na výstupních pinech správné hodnoty, vyplývající ze struktury sítě a uživatelem nastavených vstupních hodnot.

Hradlové sítě je možno z aplikace exportovat do souboru ve formátu JSON, který je strojově i uživatelsky snadno čitelný. Tento formát lze využít i k importu sítí do aplikace.

Aplikace obsahuje knihovnu základních logických obvodů, které lze do sítě importovat v podobě hradlové sítě nebo jako černou skříňku, a jednoduchý tutoriál, který uživatele seznámí se základním používáním aplikace.

Celý projekt byl navržen tak, aby jej bylo možno snadno dále rozšiřovat, a jeho zdrojový kód je dostupný pod open-source licencí.

6.1 Řešené problémy

Nejnáročnějším problémem je bezpochyby hledání co nejpřehlednější cesty pro vodiče. Zvolený algoritmus plní svou práci dobře, ale při velkém počtu vodičů v omezeném prostoru jde stále o časově náročnou operaci. Výpočetní náročnost nalezení nejlepší cesty pro všechny vodiče při importu sítě jsme vyřešili přesunutím výpočtu do vedlejšího vlákna pomocí Web workeru [9]. V případné další práci na projektu by bylo možné algoritmus založit na složitějších algoritmech. Vhodným úvodem do problematiky by mohly být akademické práce na témata související s rovinným nakreslením grafů.

Rozhodnutí nevyužít žádnou knihovnu pro zobrazování vektorové grafiky vedlo k náročnější implementaci. Museli jsme se potýkat s implementačními detaily, jejichž řešení bychom se využitím knihovny vyvarovali.

6.2 Ukázka použití aplikace

6.2.1 Bistabilní klopný obvod

Mezi poměrně jednoduché, avšak pro výpočetní techniku velmi významné obvody patří bistabilní klopné obvody. V následujícím textu jsme popsali funkci bistabilního klopného obvodu a názorně ukázali jeho sestavení a simulaci v naší aplikaci.

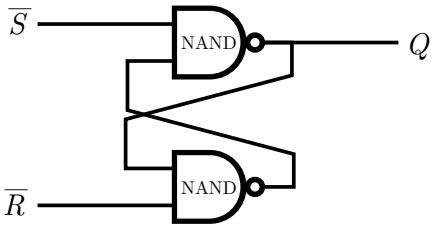
Bistabilní klopné obvody mohou nastávat dvou výstupních stavů, z nichž jsou oba stabilní a lze mezi nimi libovolně přepínat nastavováním vstupních hodnot. Takové obvody lze používat jako paměťové prvky.

Jedním z nejjednodušších takových obvodů je klopný obvod RS, který používá dvě vstupní hodnoty, běžně označované jako R (reset) a S (set), pomocí kterých lze nastavovat hodnotu výstupního pinu, běžně označovaného Q. Pravdivostní tabulka 6.1 popisuje chování takového obvodu. Nastavení výstupního pinu na hodnotu 1 probíhá nastavením S na hodnotu 1, pro nastavení výstupní hodnoty

R	S	Q
0	0	beze změny
0	1	1
1	0	0
1	1	nedefinovaný stav

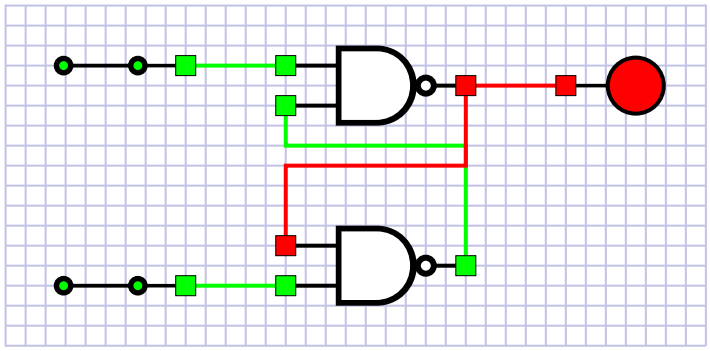
Tabulka 6.1: Pravdivostní tabulka klopného obvodu RS

na 0 je nutno nastavit vstup R na hodnotu 1. Pokud jsou oba vstupní piny nastaveny na 0, stav Q zůstane nezměněn. Nastavení S i R na hodnotu 1 způsobí nedefinovaný stav. [12]



Obrázek 6.1: Schéma implementace klopného obvodu RS pomocí hradel NAND

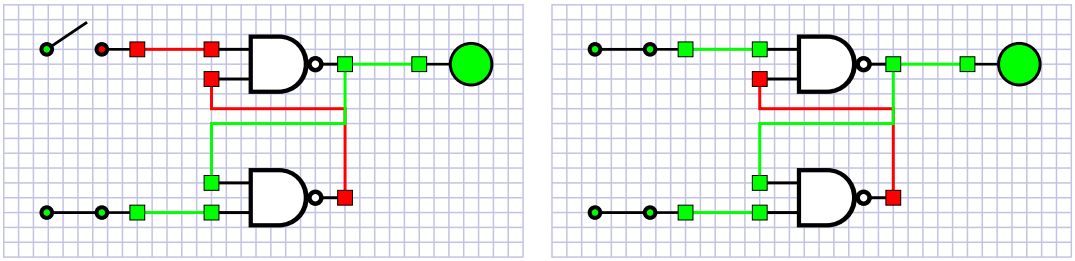
Klopný obvod RS lze sestavit ze dvou hradel NAND. Oproti výše uvedené tabulce jeho nejjednodušší implementace očekává na vstupech místo hodnot R a S jejich negace. Schéma této implementace klopného obvodu RS je znázorněno na obrázku 6.1.



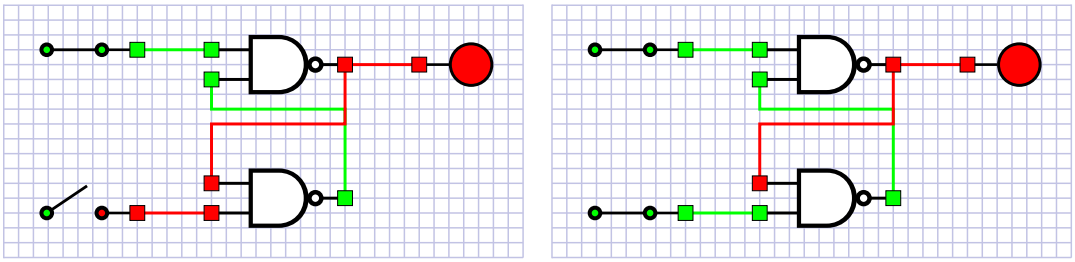
Obrázek 6.2: Klopný obvod RS sestavený v naší aplikaci

Sestavit tento obvod v aplikaci je jednoduché. Stačí prostřednictvím kontextové nabídky přidat na editační plochu dvě hradla NAND, dva vstupní prvky a jeden výstupní prvek. Komponenty následně propojíme pomocí vodičů tak, aby sestavená hradlová síť odpovídala schématu. Vzniklá hradlová síť je znázorněna na obrázku 6.2. Obě vstupní hodnoty jsme nastavili na 1 (neboť tato implementace na vstupních pinech očekává negace R a S), výstupní hodnota je 0.

Můžeme využít simulačního algoritmu aplikace a vyzkoušet, za námi sestavený klopný obvod správně funguje. Kliknutím na vstupní prvek odpovídající vstupu $\overline{S}$ nastavíme jeho hodnotu na 0 a tím se výstupní hodnota hradlové sítě změní na 1. Po nastavení vstupního prvku zpět do výchozí polohy (tedy 1) se výstupní hodnota hradlové sítě nezmění. Tento postup je znázorněn na obrázku 6.3.



Obrázek 6.3: Nastavení výstupní hodnoty na 1 v klopném obvodu RS sestaveném v naší aplikaci



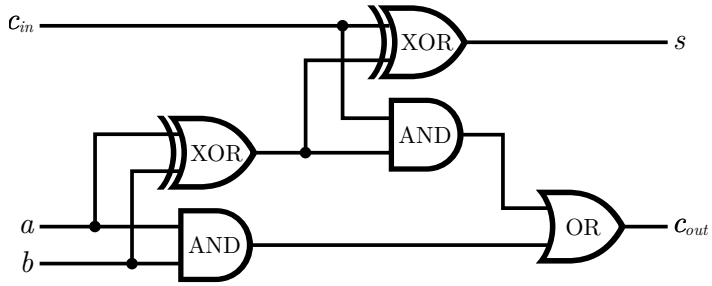
Obrázek 6.4: Nastavení výstupní hodnoty na 0 v klopném obvodu RS sestaveném v naší aplikaci

Nastavení výstupní hodnoty na 0 probíhá analogicky pomocí vstupního prvku, který odpovídá vstupu $\overline{R}$. Postup je znázorněn na obrázku 6.4.

Klopný obvod RS lze také importovat na editační plochu z knihovny hradlových sítí.

6.2.2 Hradlová síť pro součet čtyřbitových čísel

Složitější hradlovou sítí je logický obvod pro sčítání binárně reprezentovaných čísel. V tomto příkladu navážeme na schéma obvodu pro sčítání binárních hodnoty s přenosem přebývající jedničky, se kterým jsme se seznámili v kapitole *1 Úvod do problematiky*. Obvod je znázorněn na obrázku 6.5.



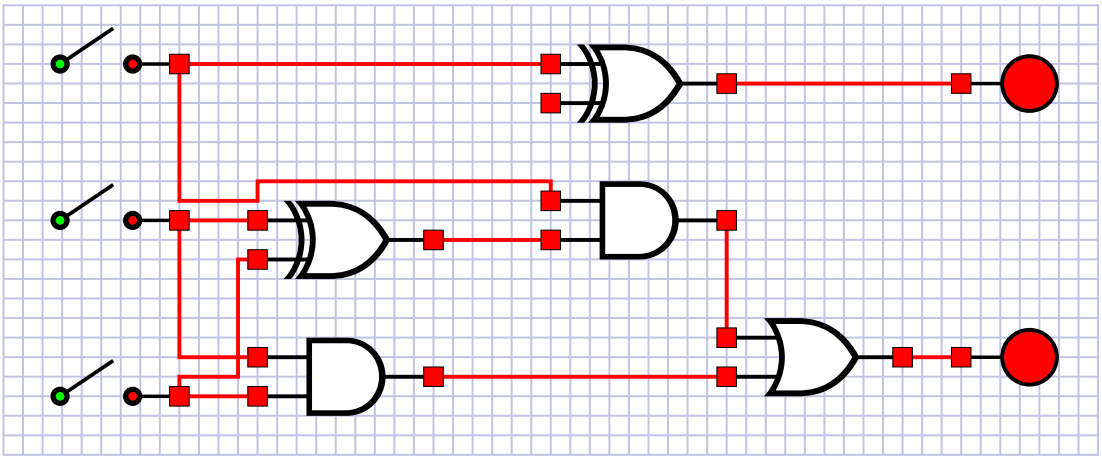
Obrázek 6.5: Sčítání binárních hodnot pomocí hradel s přenosem jedničky

Sestavení takového obvodu je poměrně jednoduché. Opět stačí pomocí kontextové nabídky přidat na editační plochu potřebná hradla a vstupní a výstupní prvky, přesunout je na vhodné místo a propojit je vodiči tak, aby hradlová síť odpovídala schématu. Síť vytvořená v naší aplikaci podle schématu je znázorněna na obrázku 6.6.

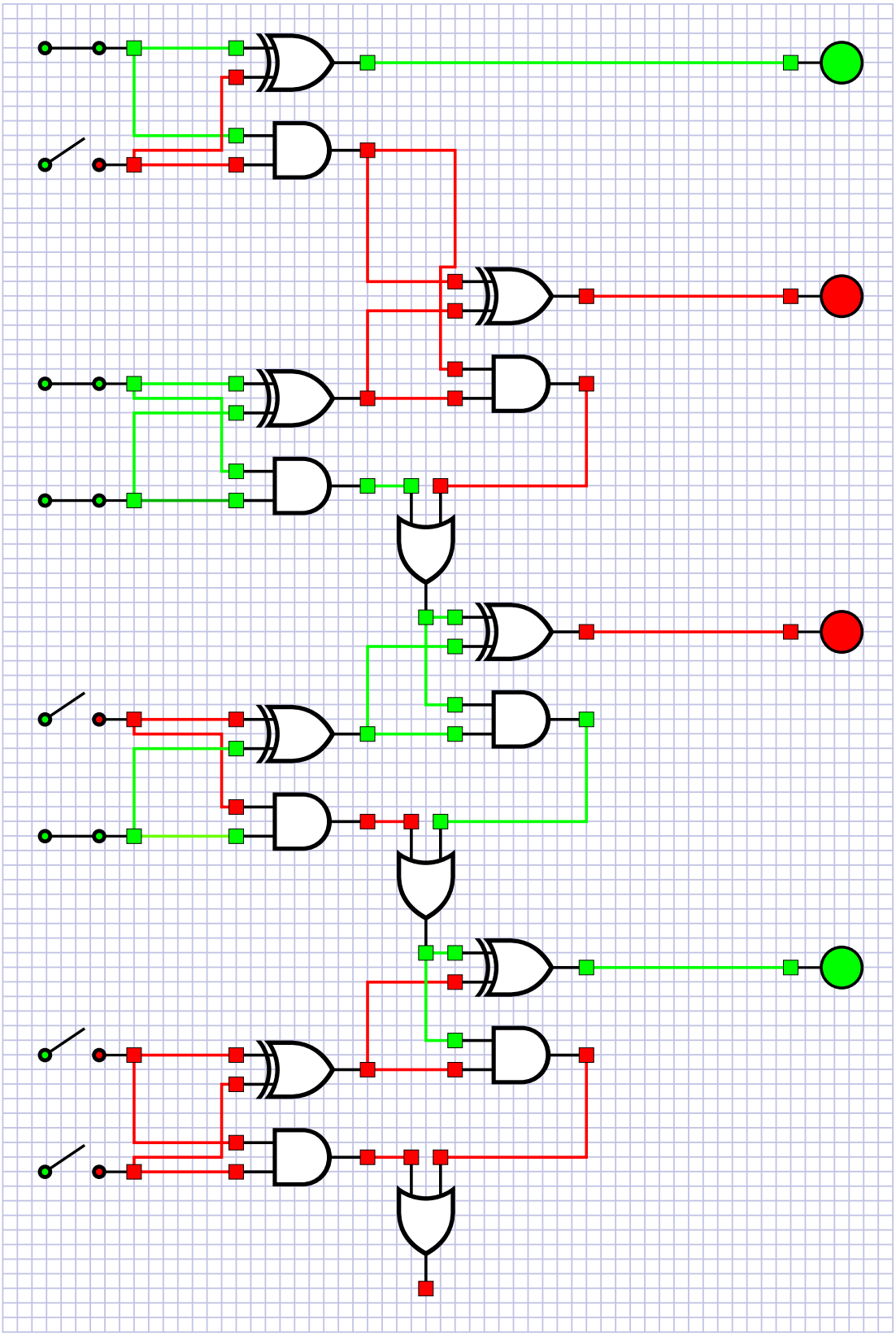
Tento obvod můžeme jeho exportem a následným importem do sítě zduplikovat a připojením výstupu odpovídajícího c_{out} prvního obvodu ke vstupu odpovídajícímu c_{in} druhého obvodu můžeme snadno sestavit obvod pro sčítání dvou dvoubitových čísel.

Duplikací tohoto nově sestaveného obvodu a analogickým propojením c_{out} prvního obvodu s c_{in} druhého obvodu získáme obvod pro sčítání dvou čtyřbitových čísel. Z obvodu můžeme následně ještě odebrat volný vstup pro přebývající jedničku a získáme tak síť znázorněnou na obrázku 6.7.

Hradlová síť pro součet dvou bitů s přenosem jedničky i sestavený sčítací obvod pro součet dvou čtyřbitových čísel jsou i součástí knihovny hradlových sítí a uživatel je tedy může i bez nutnosti manuálního sestavování importovat na editační plochu.



Obrázek 6.6: Obvod pro sčítání binárních hodnot s přenosem jedničky sestavený v naší aplikaci



Obrázek 6.7: Obvod pro sčítání dvou čtyřbitových čísel vytvořený v naší aplikaci. Na obrázku je znázorněn součet čísel 3 (`1100`) a 6 (`0110`).

Závěr

Na základě analýzy problému jsme vytvořili základní představu o aplikaci: Ústředním prvkem byla neomezená editační plocha se zachytávací mřížkou, na kterou uživatel umisťuje jednotlivá hradla a propojuje je pomocí vodičů, jejichž cesta je určována pomocí algoritmu A*.

Uživatel může na editační plochu přidat také vstupní prvky, pomocí kterých nastavuje hodnoty vstupních pinů a zároveň tím spouští simulaci hradlové sítě, která je založena na prohledávání hradlové sítě a detekci cyklů. Výstupní piny komponent pak mohou v závislosti na struktuře sítě nabývat čtyř různých hodnot — hodnotám 0 a 1 známým z digitální logiky, hodnotě značící oscilaci a neznámé hodnotě.

Díky robustnímu objektovému návrhu a využití modernější syntaxe jazyka JavaScript je aplikace snadno rozšiřitelná. Její zdrojový kód jsme zveřejnili na službě GitHub pod open-source licencí.[1] K aplikaci je také vytvořena automaticky generovaná dokumentace a všechny důležité části programu jsou podrobně okomentovány a vysvětleny přímo ve zdrojovém kódu.

Práce byla průběžně zveřejňována na službě GitHub i s odkazem ke spuštění ještě před jejím dokončením. Díky této skutečnosti jsme ještě během práce na projektu obdrželi zpětnou vazbu. Jejím obsahem byla pochvala jednoduchého uživatelského rozhraní, vhodného pro začínající studenty a prosba o radu ohledně používání aplikace na studijních tabletech, tedy bez použití myši. Ač je aplikace cílena primárně pro použití na stolních počítačích, podpora dotykových zařízení může být zajímavým rozšířením.

6.3 Možná vylepšení

Pro pohodlnější sestavování obvodů by bylo možno rozšířit sadu vstupních a výstupních komponent. Mohly by být třeba přidány komponenty pro práci s číselným vstupem a výstupem, tedy piny vstupního prvku by znázorňovaly binární zápis uživatelem zadané číselné hodnoty, výstupní prvek by pak v závislosti na nastavení svých pinů zobrazoval odpovídající číslo. Lze také rozšířit knihovnu hradlových sítí a černých skříněk o další obvody. Větší přehlednosti hradlových sítí by bylo možno docílit také implementací funkcionality větvení vodičů. Taková funkce by vizuálně zjednodušila sítě s velkým množstvím komponent připojených k jednomu výstupnímu pinu. S vodiči se pojí ještě jedno možné vylepšení, které by umožňovalo seskupit více vodičů do sběrnice.

Bylo by možno také přidat další funkce zlepšující uživatelské prostředí — takovou funkcí by například byl postranní panel, ze kterého by uživatel mohl tažením myši přidat na plochu nejčastěji používané komponenty, možnost hromadně vybrat a přesunout část komponent na editační ploše, či výše zmíněná podpora dotykových zařízení.

Dalším možným vylepšením je rozšíření syntaxe importního formátu, popsané v sekci 4.4.1.

[1] Adresa projektu: `https://github.com/janjaromirhorak/hradla/`

Bylo by také možno rozšířit současný simulační algoritmus o možnost krokování či o volbu umožňující přepnutí aplikace do režimu pracujícího se zpožděním komponent a využívajícím lokální simulační algoritmus (viz *2.1.2 Simulace hradlových sítí*).

Seznam použité literatury

[1] IEEE graphic symbols for logic functions (includes IEEE Std 91a-1991 supplement, and IEEE Std 91-1984). *IEEE Std 91a-1991 IEEE Std 91-1984*. 1991. DOI: 10.1109/IEEESTD.1991.81068.

[2] (2011). *ECMAScript 2015 Language Specification*. Ecma International, Geneva [online]. [cit. 2018-06-15]. Dostupné z: `https://ecma-international.org/ecma-262/5.1/`.

[3] (2015). *ECMAScript 2015 Language Specification*. Ecma International, Geneva [online]. [cit. 2018-05-11]. Dostupné z: `https://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf`.

[4] (2016). *Technology Radar: Babel*. ThoughtWorks [online]. [cit. 2018-06-26]. Dostupné z: `https://www.thoughtworks.com/radar/tools/babel`.

[5] (2017). *The JSON Data Interchange Syntax*. Ecma International, Geneva [online]. [cit. 2018-06-15]. Dostupné z: `https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf`.

[6] (2018). *Desktop Browser Market Share Worldwide*. StatCounter [online]. [cit. 2018-06-16]. Dostupné z: `http://gs.statcounter.com/browser-market-share/desktop/worldwide/#monthly-201701-201801-bar`.

[7] (2018). *Desktop Browser Version Market Share Worldwide*. StatCounter [online]. [cit. 2018-06-16]. Dostupné z: `http://gs.statcounter.com/browser-version-market-share/desktop/worldwide/#monthly-201701-201801-bar`.

[8] (2018). *JavaScript Object Notation*. MDN Web Docs. Mozilla [online]. [cit. 2018-06-15]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON`.

[9] (2018). *Using Web Workers*. MDN Web Docs. Mozilla [online]. [cit. 2018-06-26]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers`.

[10] Abbasian, K., Sadeghi, R. a Sadeghi, P. Design of optical tunable cnot (xor) and xnor logic gates based on 2d-photonic crystal cavity using electro-optic effect. *Digest Journal of Nanomaterials and Biostructures*, **12** (2). 2017.

[11] Adobe (2017). *Flash & The Future of Interactive Content*. Adobe Blog [online]. [cit. 2018-05-22]. Dostupné z: `https://theblog.adobe.com/adobe-flash-update/`.

[12] Balch, M. (2003). *Complete Digital Design: A Comprehensive Guide to Digital Electronics and Computer System Architecture*. Professional Engineering. McGraw-Hill, New York. ISBN 978-0-071-40927-8.

[13] Baranovskiy, D. (2013). *Raphaël – JavaScript Library* [online]. [cit. 2018-05-06]. Dostupné z: http://dmitrybaranovskiy.github.io/raphael/.

[14] Ben-Kiki, O., Evans, C. a Net, I. d. (2009). *YAML Ain't Markup Language (YAML™) Version 1.2* [online]. [cit. 2018-06-25]. Dostupné z: http://yaml.org/spec/1.2/spec.html.

[15] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F. a Cowan, J. (2006). *Extensible Markup Language (XML) 1.1 (Second Edition)*. W3C Recommendation. World Wide Web Consortium (W3C) [online]. [cit. 2018-06-16]. Dostupné z: https://www.w3.org/TR/2006/REC-xml11-20060816/.

[16] Cormen, T. H., Leiserson, C. E., Rivest, R. L. a Stein, C. (2001). *Introduction to Algorithms, Second Edition*. MIT Press, Cambridge. ISBN 0-262-03293-7.

[17] Cover, R. (2005). *XML Applications and Initiatives: Contents Listing for XML Applications and Industry Initiatives* [online]. [cit. 2018-06-16]. Dostupné z: http://xml.coverpages.org/xmlApplications.html.

[18] Cover, R. (2005). *Extensible Markup Language (XML)* [online]. [cit. 2018-06-16]. Dostupné z: http://xml.coverpages.org/xml.html.

[19] Cui, X. a Shi, H. A*-based Pathfinding in Modern Computer Games. *IJCSNS International Journal of Computer Science and Network Security*, **11**(1). 2010.

[20] Dahlström, E., Dengler, P., Grasso, A., Lilley, C., McCormack, C., Schepers, D., Watt, J., Ferraiolo, J., Jun, F. a Jackson, D. (2011). *Basic Data Types and Interfaces – Real number precision*. World Wide Web Consortium (W3C) [online]. [cit. 2018-05-08]. Dostupné z: https://www.w3.org/TR/SVG11/types.html#Precision.

[21] Friedman, A. D. a Menon, P. R. (1983). *Teorie a návrh logických obvodů*. SNTL – Nakladatelství technické literatury, Praha.

[22] Gamma, E., Helm, R., Johnson, R. a Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Boston. ISBN ISBN 0-201-63361-2.

[23] Hart, P. E., Nilsson, N. J. a Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, **4**(2). 1968. ISSN 0536-1567. DOI: 10.1109/TSSC.1968.300136.

[24] Komppa, J. (2013). *Atanua – End of Life Notice* [online]. [cit. 2018-05-05]. Dostupné z: http://sol.gfxile.net/atanua/license.html.

[25] Laforge, A. (2017). *Saying goodbye to Flash in Chrome*. The Keyword [online]. [cit. 2018-05-22]. Dostupné z: https://www.blog.google/products/chrome/saying-goodbye-flash-chrome/.

[26] Leithead, T. (2016). *DOM Parsing and Serialization: DOMParser, XMLSerializer, innerHTML, and similar APIs*. World Wide Web Consortium (W3C) [online]. [cit. 2018-06-16]. Dostupné z: https://www.w3.org/TR/DOM-Parsing/.

[27] Microsoft (2011). *What's New in Internet Explorer 9* [online]. [cit. 2018-05-06]. Dostupné z: http://msdn.microsoft.com/en-us/library/ff974378(VS.85).aspx.

[28] Nolt, J., Rohatyn, D. a Varzi, A. (1998). *Schaum's outline of theory and problems of logic*. McGraw-Hill, New York. ISBN 978-0-07-046649-4.

[29] Patel, A. (2016). *Introduction to A**. Red Blob Games [online]. [cit. 2018-06-18]. Dostupné z: https://www.redblobgames.com/pathfinding/a-star/introduction.html.

[30] Powers, S. (2010). *Using SVG For Flexible, Scalable, and Fun Backgrounds, Part I* [online]. [cit. 2018-05-06]. Dostupné z: https://alistapart.com/article/using-svg-for-flexible-scalable-and-fun-backgrounds-part-i.

[31] Rauschmayer, A. (2015). *Exploring ES6: Upgrade to the Next Version of JavaScript*. Leanpub.

[32] Smedberg, B. (2017). *Firefox Roadmap for Flash End-of-Life* . The Mozilla Blog. Mozilla [online]. [cit. 2018-05-22]. Dostupné z: https://blog.mozilla.org/futurereleases/2017/07/25/firefox-roadmap-flash-end-life/.

[33] Texas Instruments Inc. (1988). *SN5400, SN54LS00, SN54S00, SN7400, SN74LS00, SN74S00, Quadruple 2-input positive-NAND gates* [online]. [cit. 2018-05-19]. Dostupné z: http://www.datasheet.free.fr/ttl/sn_7400.pdf.

[34] Zaytsev, J., Arnott, L. a Pushkarev, D. *ECMAScript 5 compatibility table* [online]. [cit. 2018-06-16]. Dostupné z: https://kangax.github.io/compat-table/es5/.

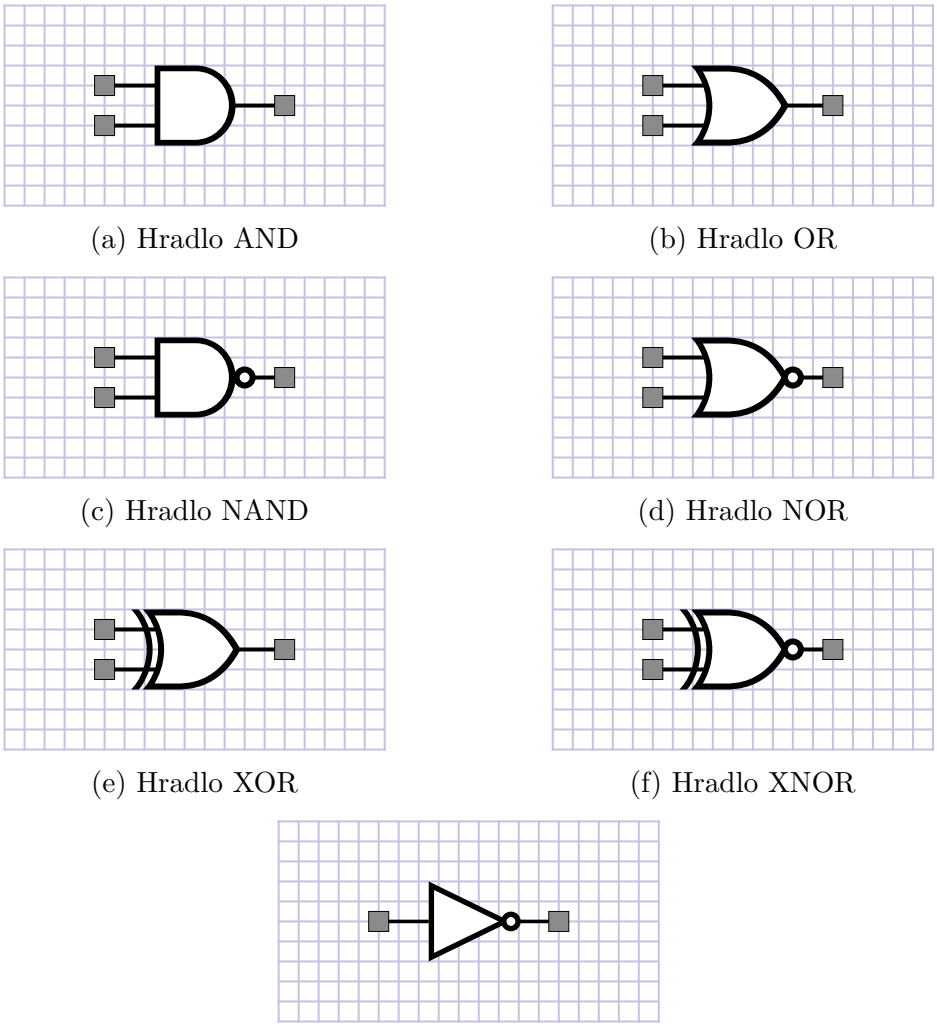
[35] Zaytsev, J., Arnott, L. a Pushkarev, D. *ECMAScript 6 compatibility table* [online]. [cit. 2018-03-19]. Dostupné z: https://kangax.github.io/compat-table/es6/.

[36] Özkan, S. (2018). *Adobe Flash Player: Vulnerability Statistics*. CVE Details [online]. [cit. 2018-05-22]. Dostupné z: https://www.cvedetails.com/product/6761/Adobe-Flash-Player.html?vendor_id=53.

Seznam obrázků

A. Přílohy

A.1 Grafické znázornění hradel v aplikaci



(a) Hradlo AND

(b) Hradlo OR

(c) Hradlo NAND

(d) Hradlo NOR

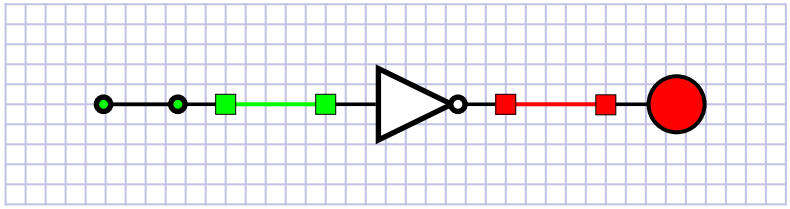
(e) Hradlo XOR

(f) Hradlo XNOR

(g) Hradlo NOT

Obrázek A.1: Grafické znázornění hradel v aplikaci

A.2 Formát souboru pro textovou reprezentaci hradlové sítě



Obrázek A.2: Jednoduchá hradlová síť sestávající ze vstupního a výstupního prvku, které jsou propojeny přes hradlo NOT.

Na obrázku A.2 je příklad jednoduché hradlové sítě. Textová reprezentace této sítě ve formátu JSON je následující:

```
{
  "boxes": [
    {
      "name": "input",
      "category": "other",
      "transform": {
        "items": [
          {"name": "translate", "args": [15, 16]}
        ]
      },
      "connections": [{"index": 0, "wireId": 0}],
      "isOn": true
    },
    {
      "name": "not",
      "category": "gate",
      "transform": {
        "items": [
          {"name": "translate", "args": [27, 16]}
        ]
      },
      "connections": [
        {"index": 0, "wireId": 1},
        {"index": 1, "wireId": 0}
      ]
    },
    {
      "name": "output",
      "category": "other",
      "transform": {
        "items": [
          {"name": "translate", "args": [41, 16]}
        ]
      },
      "connections": [{"index": 0, "wireId": 1}]
    }
  ]
}
```

A.3 Popis elektronické přílohy

Součástí práce je elektronická příloha obsahující zdrojové kódy aplikace a složku se sestavenou aplikací připravenou ke spuštění. Celá elektronická příloha je zabalena do archivu TAR, komprimovaného metodou Gzip.

A.3.1 Souborová struktura přílohy

- `dist` — adresář obsahující sestavenou aplikaci
  - `css`, `fonts`, `img`, `js`, `library` — adresáře se zkompilovanými soubory a dalšími zdroji použitými v aplikaci
  - `docs` — adresář obsahující dokumentaci
  - `index.html` — HTML soubor sloužící ke spuštění aplikace
- `src` — adresář zdrojových souborů k aplikaci
  - `es6` — adresář se zdrojovými soubory v jazyce ECMAScript 2015
  - `html` — adresář se šablonami soubory v jazyce HTML
  - `scss` — adresář s kaskádovými styly v jazyce Sass
  - `fonts`, `help`, `img` — adresáře obsahující další pomocné soubory: písma, obsah stránky s uživatelskou nápovědou a obrázky použité v aplikaci
- `library` — adresář obsahující soubory jednotlivých hradlových sítí a černýsh skříněk v textovém formátu, který umožňuje import těchto sítí do aplikace
- `gulpfile.js` — popis sestavovacího procesu pro nástroj *Gulp*
- `package.json` — konfigurační soubor pro nástroj *npm*
- `README.md` — základní informace o sestavovacím procesu a struktuře aplikace