VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

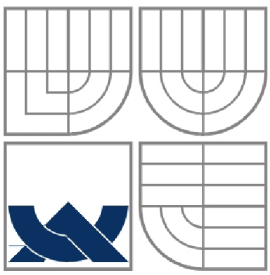
GENERÁTOR A ŘEŠITEL HRY SUDOKU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

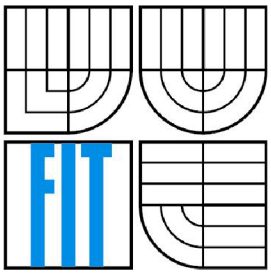
AUTOR PRÁCE
AUTHOR

ELIŠKA POLÍNKOVÁ

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GENERÁTOR A ŘEŠITEL HRY SUDOKU

SUDOKU GENERATOR AND SOLVER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ELIŠKA POLÍNKOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

MGR. FILIP GOLDEFUS

BRNO 2010

Abstrakt

Tato práce pojednává o způsobech generovaní a řešení hry Sudoku. Popisuje možné přístupy k problému řešení, a to: využití logických metod a využití metod prohledávání stavového prostoru. Praktickou část této práce tvoří návrh a implementace aplikace umožňující generování, řešení, ověřování vlastností, vytváření vlastních zadání a export her do formátu XML.

Abstract

This thesis discusses ways of generating and solving Sudoku games. It describes possible approaches to problem solving, namely: the use of logical methods and the use of state space search methods. Practical part of this work involves the design and implementation of application enabling the generating, solving, verification of attributes, creating your own sudoku problem and export games to XML file.

Klíčová slova

sudoku, generátor sudoku, řešitel sudoku, metoda zpětného navracení, prohledávání stavového prostoru, slepé prohledávání do šířky, slepé prohledávání do hloubky, logické řešení sudoku

Keywords

sudoku, sudoku generator, sudoku solver, backtracking, state space searching, breath-first search, depth-first search, solving sudoku by logic

Citace

Eliška Polínková: Generátor a řešitel hry sudoku, bakalářská práce, Brno, FIT VUT v Brně, 2010

Generátor a řešitel hry sudoku

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením Mgr. Filipa Goldefuse.
Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

……
Eliška Polínková
19. května 2010

Obsah

1 Úvod

I přesto, že první sudoku bylo v novinách otištěno již v roce 1979 pod názvem „Number Place“, vzrostla obliba sudoku mezi širokou veřejností až v posledních letech, kdy se hlavolam do Evropy vrátil z Japonska, kde získal i své současné jméno. S vzrůstající popularitou sudoku začaly vznikat také různé programy pro jeho generovaní a řešení. Bohužel i dnes se stává, že (převážně v tištěné podobě) nacházíme sudoku, u kterých nebyly dodrženy základní požadavky na ně kladené, a jsou tedy pro řešení naprosto nepoužitelná.

Tato práce se snaží podat ucelený pohled na problematiku generování a řešení sudoku. Obsahuje podrobný popis vlastností správně zadaného sudoku a ukázku několika variant hry. Dále popisuje princip generování hry sudoku a možné postupy při jeho řešení, a to: řešení pomocí logických metod, které používá člověk, a metod, které pro řešení využívají prohledávání stavového prostoru.

Praktická část této práce zahrnuje návrh generátoru sudoku, tří algoritmů pro jeho řešení a implementaci aplikace, která uživateli umožňuje snadné generování her podle zadaných parametrů, exportování těchto zadání do formátu xml, ruční vložení popř. importování vlastního sudoku a zjištění jeho obtížnosti a řešitelnosti.

Z závěru práce jsou porovnány a zhodnoceny implementované algoritmy a jsou zde nastíněna možná rozšíření programu.

2 Princip hry

Cílem hry sudoku je doplnit číslice od 1 do 9 do předvyplněné čtvercové tabulky, kterou tvoří 9x9 polí. Pole jsou dále rozděleny na 9 čtverců o velikosti 3x3 pole. Číslice je nutné doplnit tak, aby se v každé řadě, sloupci a čtverci vyskytovala každá z nich právě jednou. Ukázkové zadání hry sudoku je na obrázku 2.1.

	6	5	2		1		7	
8			7	3			9	
2			8				4	
		8			2			
7								4
			1			2		
	3				7			1
	8			6	3			7
	2		5		8	9	3	

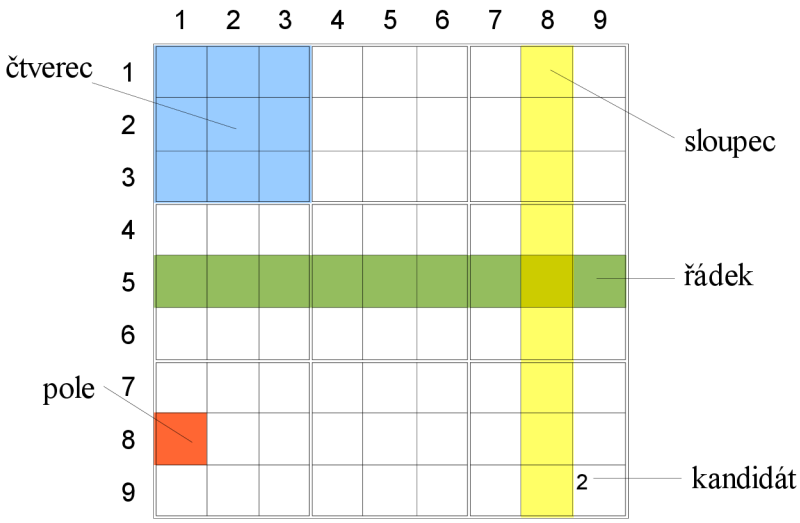
Obrázek 2.1: Ukázka zadání hry sudoku

2.1 Terminologie

V dalším textu budou používány následující pojmy, které se k řešení a generování sudoku vztahují:

pole, políčko = nejmenší oblast v sudoku, obsahuje jednu číslici,
řádek = devět polí v jednom řádku,
sloupec = devět polí v jednom sloupci ,
čtverec = seskupení devíti polí o velikosti 3x3 pole,
oblast = řádek, sloupec nebo čtverec,
kandidát = číslice, která se může, ale nemusí vyskytovat v poli.

Pojmy jsou ukázány na obrázku 2.2. Pro orientaci v ukázkových sudoku budou používány souřadnice ve tvaru [číslo řádku, číslo sloupce]. Řádky a sloupce jsou číslovány od 1.



Obrázek 2.2: Vysvětlení pojmů

2.2 Požadavky na správně zadané sudoku

Na správně zadané sudoku jsou kladeny dva požadavky a to:

1. Sudoku musí být logicky řešitelné, tzn. že všechna čísla musí jít doplnit pouze na základě logických úvah bez hádání.
2. Sudoku musí mít právě jedno možné řešení.

Sudoku existuje v několika obtížnostech, přičemž obtížnost není dána počtem „neviditelných“ polí, ale určují ji vazby mezi jednotlivými políčky a složitost metod, které je nutno použít k tomu, abychom sudoku vyřešili.

Obecně se uvádí, že minimální počet předvyplněných polí v sudoku, aby byly splněny výše uvedené podmínky, je 17. Toto číslo je za nejnižší možné považováno hlavně proto, že ještě nebylo nalezeno takové sudoku, které by mělo předvyplněno jen 16 číslic a podmínky by splňovalo. Nalezením takového sudoku se zabývá např. [1].

2.3 Obtížnost sudoku

Důležitou vlastností každého sudoku je jeho obtížnost. Ta se dá určit např. tak, že se řešící metody rozdělí do skupin podle obtížnosti a poté se při řešení sudoku kontroluje, jaká nejtěžší metoda byla použita. Podle skupiny, do které tato metoda patří, se pak určí obtížnost vygenerovaného sudoku. Výhodou tohoto přístupu je, že řešitel přesně ví, jaké metody potřebuje ovládat, aby ho vyřešil. Nevýhodou je, že skutečné úsilí k vyřešení sudoku zařazených tímto způsobem do stejné kategorie, může být velmi rozdílné.

Druhým způsobem, jak obtížnost sudoku vyhodnotit, je, že každou metodu ohodnotíme dle její náročnosti určitým počtem bodů. Poté sudoku řešíme a počítáme, kolikrát byla která metoda použita.

Výsledná obtížnost sudoku je dána sečtením bodů za všechny použité metody. Nevýhodou tohoto přístupu je narozdíl od předchozího to, že může vzniknou jednoduché sudoku, pro jehož vyřešení je třeba použít např. jen jednu složitou metodu, s níž si začátečník neporadí. Další komplikací při ověřování obtížnosti tímto způsobem může být otázka, v jakém poměru ohodnotit jednotlivé metody.

2.4 Varianty

Se vzrůstající popularitou sudoku začaly vznikat i různé alternativní varianty. Protože je těchto nových variant nepřeberné množství a neustále vznikají nové, např. pro potřeby Mistroství světa v sudoku, budou zde uvedeny pouze ty, které se klasickému sudoku podobají nejvíce, a algoritmy pro jejich generování a řešení jsou pro ně částečné přenositelné z klasického sudoku.

Diagonální sudoku

Diagonální sudoku má stejné vlastnosti jako klasické, ale navíc se číslice nesmí opakovat v obou úhlopříčkách. Ukázka je na obrázku 2.3.

		8					9	
	9				5	1		
1							3	
		4	5			2		
6				1				5
	2							7
				5				
	7			3		9		
5	3				8			

Obrázek 2.3: Diagonální sudoku

Sousledné sudoku

Sousledné sudoku se od klasického liší v tom, že v žádných dvou sousedních polí nesmí být číslice, jejichž rozdíl je roven jedné. Tento druh sudoku se od klasického vizuálně nijak neliší.

Sudoku jiného rozměru

Jedná se obecně o sudoku jiného rozměru než 9x9. Nejčastěji se vyskytují např. sudoku 4x4, 6x6, 8x8, 16x16 apod.

Velmi oblíbené je hexadecimální sudoku o rozměru 16x16, které tvoří 16 čtverců o velikosti 4x4 pole. Do hexadecimálního sudoku se doplňují číslice 0 až 9 a písmena A až F. Ukázku můžete vidět na obrázku 2.4.

	5					0	2		6		A			9	F
		D			A				9	F	8	4	6	2	
		0	E	9	F				1	3	C				
2	9	F	A			6							7	D	
B	3			2	7								5		6
	4	A				5		F			9	0			
			5	E					A		7				
					B	1	3		D	8	4				
				4	5	C		D	8	1					
				8		7					3	2			
			3	6			A		7				E	5	
8		7								5	6			1	A
	D	8							3			5	4	6	E
				F	1	D				A	C	9	3		
	E	2	6	B	4	3				C			0		
C	1			5		E		0	F					8	

Obrázek 2.4: Hexadecimání sudoku

2.5 Matematický pohled na sudoku

Z matematického hlediska lze dle [2] na sudoku nahlížet jako na speciální typ tzv. *latinského čtverce*. Latinský čtverec je pole o velikosti n x n vyplněné n číslicemi (popř. písmeny, symboly apod.) tak, že se žádná číslice nesmí opakovat v řádcích ani ve sloupcích. Pokud přidáme pravidlo, že se číslice nesmí opakovat ve vnitřních čtvercích, vznikne latinský čtverec, který zároveň splňuje podmínky pro sudoku.

3 Generování sudoku

Počet všech možných řešení sudoku byl vyčíslen na: 6 670 903 752 021 072 936 960 variant. Jedni z prvních autorů výpočtu byli Bertram Felgenhauer a Frazer Jarvis viz [3].

3.1 Generování řešení sudoku

Při generování vyplněných sudoku desek je nejjednodušší vycházet z tzv. *základního řešení (root solution)*, které je zobrazeno na obrázku 3.1. To se dá dále upravovat podle následujících pravidel tak, aby vzniklo unikátější řešení:

1. Sudoku zůstane validní, pokud se mezi sebou vymění řádky (sloupce), které jsou sdíleny třemi shodnými čtverci. Můžeme tedy mezi sebou vyměnit řádky (sloupce) číslo 1, 2, 3 nebo 4, 5, 6 nebo 7, 8, 9.
2. Můžeme mezi sebou vyměňovat jednotlivé číslice, např. do všech polí s číslem 1 uložíme číslo 2 a naopak.

Základní algoritmus dle [4], upravený do jazyka C++:

```
int x = 0; // Pomocné číslo pro výpočet nové číslice.
int sudoku[9][9]; // Dvourozměrné pole pro uložení sudoku
for (int i = 0; i < 3; i++, x++) { // Pomocné proměnné pro
    for (int j = 0; j < 3; j++, x += 3) { // výpočet řady.
       for (int k = 0; k < 9; k++, x++) { // Sloupec.
          sudoku[3 * i + j][k] = x % 9 + 1;
       }
    }
}
```

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	7	8	9	1	2	3	4	5	6
3	4	5	6	7	8	9	1	2	3
4	2	3	4	5	6	7	8	9	1
5	8	9	1	2	3	4	5	6	7
6	5	6	7	8	9	1	2	3	4
7	3	4	5	6	7	8	9	1	2
8	9	1	2	3	4	5	6	7	8
9	6	7	8	9	1	2	3	4	5

Obrázek 3.1: Základní řešení

Druhý způsob generování vychází z prázdného sudoku, do kterého se náhodně dosadí několik číslic. Poté se ověří, zda je sudoku validní, a pokud ano, vyřeší se vzniklé sudoku za použití řešícího algoritmu. Problém tohoto přístupu spočívá v nutnosti implementace velmi rychlého řešícího algoritmu, který si v rozumném čase poradí i s téměř prázdnou deskou.

3.2 Možné přístupy ke generování

3.2.1 Přidávání číslic do prázdné desky

Jednou z možností, jak sudoku generovat, je náhodně přidávat číslice do prázdné desky, zkoušet její validitu, a pokud je sudoku validní, kontrolovat, zda má sudoku jen jedno možné řešení. V okamžiku, kdy je sudoku validní, logicky řešitelné a má právě jedno možné řešení, je sudoku vygenerováno.

Tento přístup může být ale komplikovaný a časově velmi náročný, jelikož se při každém přidání číslice musí ověřit, zda pro sudoku existuje nějaké řešení a zda je sudoku logicky řešitelné, což může být hlavně pro sudoku s malým množstvím vyplněných čísel časově náročné.

3.2.2 Využití základního řešení

Jednodušší přístup ke generování sudoku je vygenerovat základní řešení, to upravit na řešení unikátnější (viz kapitola 3.1) a poté pole náhodně skrývat nebo naopak zobrazovat. Po každém skrytí (zviditelnění) políčka se provede pouze ověření logické řešitelnosti, a pokud je výsledek v pořádku, zůstane pole skryté (viditelné). Při této metodě odpadá problém s existencí řešení, který je hlavní komplikací v předchozí metodě.

V případě, kdy pole mažeme, je vhodné tento přístup optimalizovat skrýváním více polí zaráz, což zejména ze začátku, kdy lze předpokládat, že odebráním čísla z pole zůstane řešitelnost zachována, ušetří procházení celé desky při ověřování řešitelnosti. Počet skrývaných polí je vhodné se snižujícím se počtem vyplněných polí snižovat. Stejně tak u odkrývání polí je vhodné na začátku polí odkrýt více.

3.3 Generování sudoku určité obtížnosti

Nejjednodušší přístup, jak vygenerovat sudoku o určité obtížnosti, je generování náhodných sudoku a následné zjišťování jejich obtížnosti až do té doby, než se vygeneruje sudoku odpovídající obtížnosti zvolené. Problém u tohoto přístupu nastává v okamžiku, kdy se snažíme vygenerovat sudoku, na které je k vyřešení nutné aplikovat konkrétní metodu. Pokud není výskyt této metody příliš častý, může opakované generování trvat příliš dlouho.

4 Řešení sudoku

4.1 Řešení hrubou silou

Nejzákladnějším přístupem k řešení sudoku je použití tzv. „hrubé síly“. Toto řešení probíhá tak, že se sudoku prochází po jednotlivých polích a v okamžiku, kdy se narazí na pole prázdné, se do něj doplní číslice 1. Na nově vzniklé zadání sudoku se aplikuje stejná řešící metoda. Pokud je její výsledek kladný (došlo k vyřešení sudoku), zůstává číslice na svém místě a pokračuje se v hledání další nevyplněné číslice, pokud je výsledek záporný (nastala chyba a sudoku vyřešit nejde), změní se doplněná číslice na hodnotu další číslice, tj. 2. Tento postup se opakuje až do té doby, než se sudoku úspěšně vyřeší nebo než se vyčerpají všechny možnosti. V takovém případě není sudoku validní a nelze ho vyřešit.

Jelikož jsou při tomto postupu testovány všechny varianty řešení zadaného sudoku, může být tato metoda ve srovnání s metodami používajícími logiku neúnosně pomalá. Její použití je tedy vhodné pouze v případě, že v zadání chybí malý počet číslic. Metoda se dá aplikovat například na sudoku s více variantami řešení, které se nejprve vyřeší logicky a metoda řešení hrubou silou se použije až v okamžiku, kdy zbývá doplnit posledních pár číslic. Výsledkem bude první možná varianta výsledku sudoku.

4.2 Metody řešení používané člověkem

Metod řešení pomocí logických úvah existuje celá řada od nejjednodušších, bez kterých se žádný řešitel neobejde, až po speciální metody, které je potřeba znát na vyřešení nejsložitějších sudoku.

Tyto metody dělíme podle výsledku jejich aplikace na:

1. Metody pro přímé doplnění číslice
   Číslici můžeme doplnit do pole, pokud je její výskyt v konkrétním poli jistý. To může nastat ve dvou případech:

   a) Pro dané políčko existuje pouze jeden kandidát.
   b) V určité oblasti se kandidát vyskytuje jen na jedné pozici.

   Obě uvedené metody jsou detailněji popsány v kapitole 4.2.1. Použitím pouze těchto metod vyřešíme jen ty nejjednodušší sudoku. V drtivé většině případů se člověk neobejde bez metod eliminace kandidátů.

2. Metody pro postupnou eliminaci kandidátů
   Jakmile nejde další číslice do sudoku doplnit přímo, přichází na řadu metody používané pro eliminaci kandidátů. Těchto metod existuje velké množství. Základní jsou podrobně popsány v kapitole 4.2.2 a některé složitější v kapitole 4.2.3.

Následuje detailní popis metod používaných pro řešení sudoku. Názvy metod se do češtiny nepřekládají, proto jsou zachovány v původním anglickém znění.

4.2.1 Metody pro přímé doplnění číslice

Naked singles

Pokud se v poli vyskytuje pouze jeden kandidát, můžeme jeho hodnotu do pole dosadit.

Na obrázku 4.1 můžeme metodu aplikovat na modře vybarvené pole. Mezi číslicemi, které na toto pole mají vliv (jsou vypsány tučně) se vyskytují všechny kromě čísla 3. Do pole tedy můžeme dosadit číslo 3.

Hidden singles

Pokud se v některé oblasti vyskytuje určitý kandidát pouze jednou (pouze na jedinné pozici), můžeme do tohoto pole hodnotu kandidáta dosadit.

Na obrázku 4.2 je uveden příklad metody *Hidden singles*. V řádku č. 4 (modře podbarveném) se může číslice 1 vyskytovat pouze v žlutém poli, můžeme ji tedy dosadit.

	1	2	3	4	5	6	7	8	9
1				**8**					
2	**5**	**2**			**1**			**4**	**8**
3						**9**			
4							3	7	
5		3		**6**		2			
6		9							
7	9						4	3	
8		4	3				9		
9				**7**					2

Obrázek 4.1: Metoda „Naked Singles“

	1	2	3	4	5	6	7	8	9
1			9					2	3
2				5	3		**1**		
3									
4				4				7	
5	**1**	6							
6				3			6		**1**
7	9				4			4	
8		3		5				8	
9						**1**			9

Obrázek 4.2: Metoda „Hidden Singles“

4.2.2 Základní metody eliminace kandidátů

Popis následujících metod je volně převzat z [5]. Tyto metody mají na rozdíl od předchozích dvou za úkol pouze eliminovat kandidáty. Ještě před jejich aplikací je tedy vhodné a u složitějších metod i nutné všechny kandidáty vyplnit.

Pointing

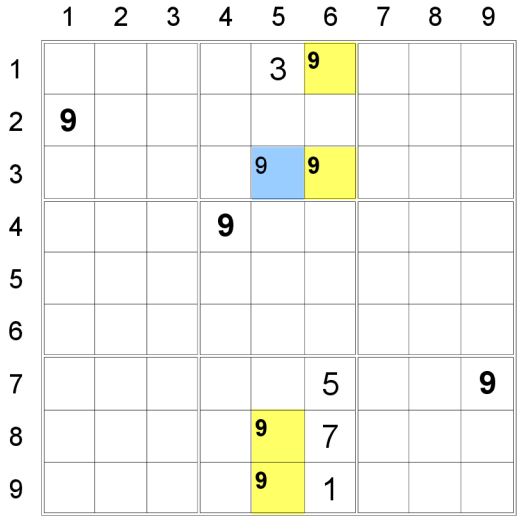
Pokud jsou všechny výskyty určité kandidátní číslice v rámci čtverce v jednom řádku (sloupci), můžeme tuto číslici eliminovat ve zbytku daného řádku (sloupce).

Na obrázku 4.3 vidíme sudoku, kde je možné tuto metodu aplikovat. Ve spodním čtverci uprostřed mohou být devítky pouze ve dvou polích. Ať devítku dosadíme do jakéhokoli z nich, musíme ji v obou případech eliminovat v modře vyznačeném poli.

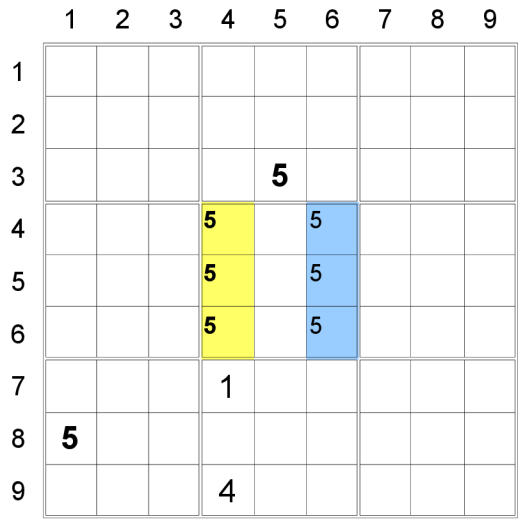
Claiming

Pokud se v řádku (sloupci) vyskytuje určitá kandidátní číslice pouze v rámci jednoho čtverce, můžeme ve zbytku tohoto čtverce kandidáta eliminovat.

Ukázku této metody vidíme na obrázku 4.4. Ve čtvrtém sloupci se číslice 5 může vyskytovat pouze ve třech žlutě vyznačených polích, které jsou v jednom čtverci. Ve zbytku tohoto čtverce (modře vyznačená pole) můžeme pětku eliminovat.



Obrázek 4.3: Metoda „Pointing“



Obrázek 4.4: Metoda „Claiming“

Naked subsets

Pokud n polí v oblasti obsahuje n shodných kadidátů, můžeme tyto kandidáty eliminovat z ostatních polí v této oblasti.

Například na obrázku 4.5 jsou v 5. řádku tři políčka, která obsahují tři kandidáty, a to 1, 2 a 3. Víme tedy, že se tyto číslice nemůžou vyskytovat v řádku nikde jinde, což znamená, že v modře vybarvených políčkách můžeme čísla 1, 2 a 3 eliminovat.

Hidden subsets

Metoda *Hidden subsets* pracuje na stejném principu jako *Hidden singles*, ale s tím rozdílem, že platí pro n kandidátů, kteří se v oblasti vyskytují pouze n-krát v n shodných polích. Ostatní kandidáty v těchto n polích můžeme eliminovat.

Ukázku metody pro n = 2 můžeme vidět na obrázku 4.6. Číslice 1 a 4 jsou kandidátní pouze v modře podbarvených polích. Z toho vyplývá, že pokud bude číslice 4 v poli [1, 1], musí být číslice 1 v poli [2, 2] a naopak. V obou těchto polích se mohou vyskytovat jen číslice 1 a 4 což znamená, že zbylé kandidáty v nich můžeme zrušit.

	1	2	3	4	5	6	7	8	9
1									
2									
3			2						
4	7			4	5		2	8	1
5	8	123	369	169	123	123	7	4	5
6		5	4		8	7			3
7				3					
8				2					
9			1						

Obrázek 4.5: Metoda „Naked Subsets“

	1	2	3	4	5	6	7	8	9
1	134	2	5						
2	9	148	378						
3	87	378	6	1	4				
4									
5									
6	3								
7			4						
8			1						
9									

Obrázek 4.6: Metoda „Hidden subsets“

X-Wings

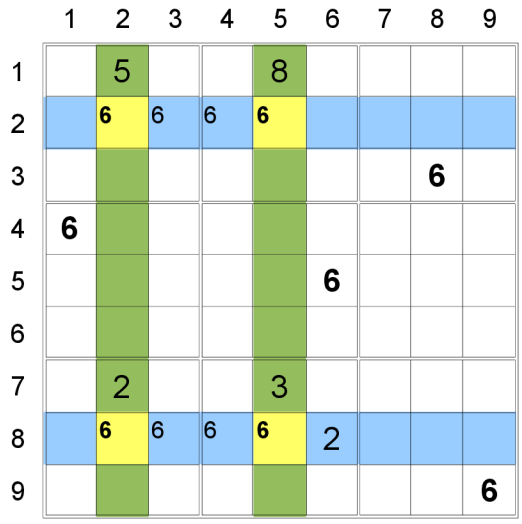
Tuto metodu lze aplikovat, pokud v sudoku existují dva řádky (sloupce), které mají určitého kandidáta pouze ve dvou shodných sloupcích (řádcích).

V sudoku na obrázku 4.7 se kandidátní číslice 6 v 2. a 5. sloupci vyskytuje pouze v řadách 2 a 8. Ve zbytku těchto řad (modře vyznačená pole) můžeme kandidáta 6 eliminovat, jelikož existují jen dvě varianty umístění číslice 6 v sloupcích 2 a 5. Buď bude šestka na pozicích [2, 2] a [8, 5], nebo na pozicích [2, 5] a [8, 2]. V obou těchto případech se bude muset kandidát 6 eliminovat ve zbytku řádků 2 a 8.

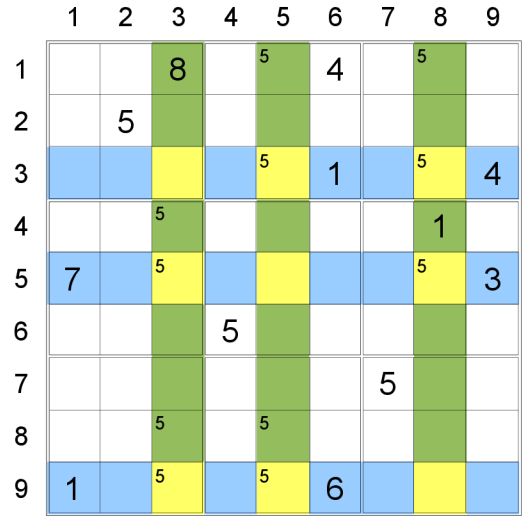
Swordfish, Jellyfish

Jde o metody pracující na stejném principu jako *X-Wings*, ale jsou rozšířené na více políček. Metodu *Swordfish* použijeme, když tři řádky (sloupce) obsahují určitého kandidáta pouze ve třech sloupcích. Dle [2] není u této metody nutné, aby se kandidát vyskytoval v řádcích (sloupcích) vždy třikrát, ale je nutné, aby se v každé řadě (sloupci) vyskytoval alespoň jednou. Pokud tato pravidla platí, můžeme kandidáta ve zbytku sloupců (řádků) eliminovat. Metoda *Jellyfish* pracuje na stejném principu, ale je rozšířena na 4 řádky a sloupce.

Ukázka aplikace metody *Swordfish* pro řádky 3, 5 a 9 je uvedena na obrázku 4.8. Žlutě vyznačená jsou pole, v nichž jediných se může vyskytovat kandidát 5 (v rámci řádků 3, 5 a 9). Kandidáti se tedy vyskytují pouze ve sloupcích 3, 5 a 8. Nyní víme, že existují pouze dvě varianty výskytu číslic 5 v řádcích 3, 5 a 9. Buď bude pětka na pozicích [3, 5], [5, 8] a [9, 3], nebo na pozicích [3, 8], [5, 3] a [9, 5]. V zeleně vybarvených polích můžeme tedy pětku eliminovat.



Obrázek 4.7: Metoda „X-Wings“



Obrázek 4.8: Metoda „Swordfish“

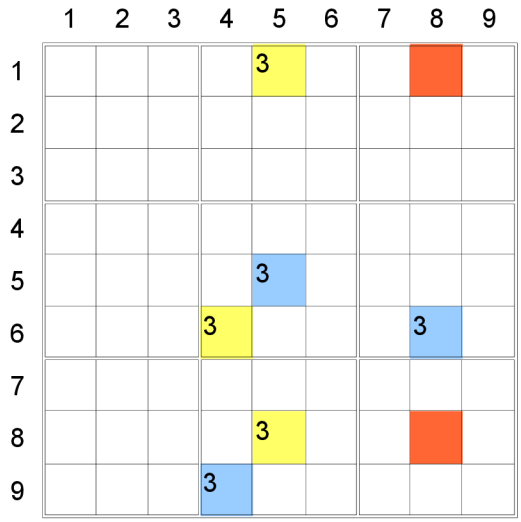
4.2.3 Pokročilé metody eliminace kandiátů

Následující principy metod vychází z [2] a [5].

Coloring

Metoda *Coloring* je dle [6] založena na analýze na sebe navazujících logických kroků – řetězců. Pokud se v oblasti vyskytuje určitý kandidát pouze dvakrát, vzniká mezi políčky, v nichž se vyskytuje, speciální vztah – pokud je číslice v prvním políčku, v druhém už být nemůže, a pokud v prvním políčku číslice nebude, musí být automaticky v poli druhém. Tento vztah se nazývá *silné propojení* (*strong connection*) .

Princip této metody je nejjednodušší vysvětlit na příkladu (viz obrázek 4.9). V obrázku jsou pro přehlednost vypsány pouze ty pole, ve kterých se může nacházet kandidát č. 3, a mezi nimiž platí výše uvedený vztah. Samotné barvení probíhá tak, že vybereme jednu dvojici polí se dvěma kandidáty v jedné oblasti a pro každé pole zvolíme jinou barvu. Vybereme tedy např. pole [6, 4], obarvíme ho žlutě a vyhledáme všechna neobarvená pole, které jsou s ním ve vztahu (v tomto případě pole [9, 4] a [6, 8]), a obarvíme je druhou barvou (modře). Takto postujujeme, až vybarvíme všechna navazující pole. Nyní může dojít k samotné eliminaci, která proběhne v polích, které sdílí oblast s alespoň jedním políčkem od každé barvy (pole [1, 8], které sdílí řádek se žlutým polem [1, 5] a modrým polem [5, 8], a pole [8, 8], které sdílí sloupec s modrým pole [6, 8] a žlutým polem [8, 5]).

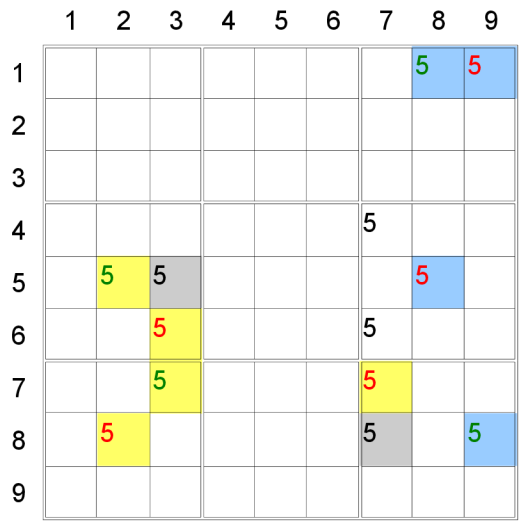


Obrázek 4.9: Metoda „Coloring“

Multi – coloring

Tato technika se používá v případě, kdy v sudoku existuje více řetězců získaných metodou *Coloring*, a pokud jdou vzniklé řetězce propojit tzv. *slabým propojením* (*weak connection*). Slabé propojení mezi dvěma buňkami znamená, že pokud se do jednoho pole kandidát dosadí, ve druhém poli už toto číslo být nemůže, ale naopak pokud zjistíme, že v prvním poli být kandiát nemůže, neznamená to, že v druhém poli kandidát bude. Tyto propojení musíme do sudoku také vhodně poznačit.

Ukázka takových řetězců je uvedena v obrázku 4.10. V tomto sudoku můžeme najít dva řetězce, a to modrý a žlutý. Mezi nimi vzniká slabé spojení mezi políčky [8, 2] a [8, 9]. Bude-li číslo 5 v políčku [8, 2], v poli [8, 9] již být nemůže, ale pokud na pozici [8, 2] pětka nebude, v poli [8, 9] pořád být může i nemusí. Další slabé propojení je mezi políčky [5, 2] a [5, 8]. Spojení jsou v obrázku naznačena barvou písma. Zelená (true) znamená, že pokud v tomto poli kandidát bude, nemůže být v polích vypsaných červeně (false). Díky tomu, že výše uvedená spojení jsou navíc mezi políčky, z nichž jedno je true a druhé false, mohou být nyní řetězce propojeny do jednoho silně propojeného. Z toho dle pravidel pro eliminaci kandidátů podle metody *Coloring* vyplývá, že můžeme kandidáta eliminovat ve všech polích, které sdílí oblast s políčkem true a zároveň sdílí oblast s políčkem false. V příkladu se jedná o šedě vybarvená pole [5, 3] a [8, 7].

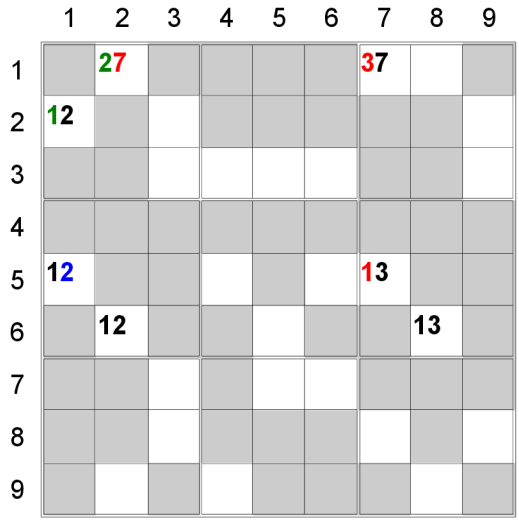


Obrázek 4.10: Metoda „Multicoloring“

Forcing Chains

Tato metoda se svým principem blíží k hádání, ale na rozdíl od obyčejného náhodného dosazování je člověk schopen ji jednoduše provést. Obtížnost metody závisí na tom, kolik kandidátů obsahují pole, na které se metoda aplikuje.

Jednoduchý příklad pro pole se dvěma kandidáty, na kterém si princip metody vysvětlíme, je na obrázku 4.11. Šedě vybarvené pole v obrázku představují pole vyplněné. Nejprve vybereme jedno pole se dvěma kandidáty, např. [1, 2], a na zkoušku do něj jednoho z nich dosadíme – v obrázku naznačeno zelenou barvou číslice. Pokud bude v tomto poli dvojka, znamená to, že v poli [2, 1] musí být jednička. Pokud v poli [2, 1] bude jednička musí být v poli [5, 1] dvojka. Tímto způsobem by se vyplnila všechna navazující pole. Nyní se vrátíme na začátek k poli [1, 2] a dosadíme do něj druhého kandidáta – červená barva. Pokud bude v tomto poli sedmička, musí být v poli [5, 7] jednička a v poli [5, 7] opět dvojka (vyznačná modře). Tím pádem je jasné, že sedmičku můžeme z pole [5, 7] eliminovat a dosadit do něj napevno číslici 2.



Obrázek 4.11: Metoda „Forcing Chains“

4.3 Řešení prohledáváním stavového prostoru

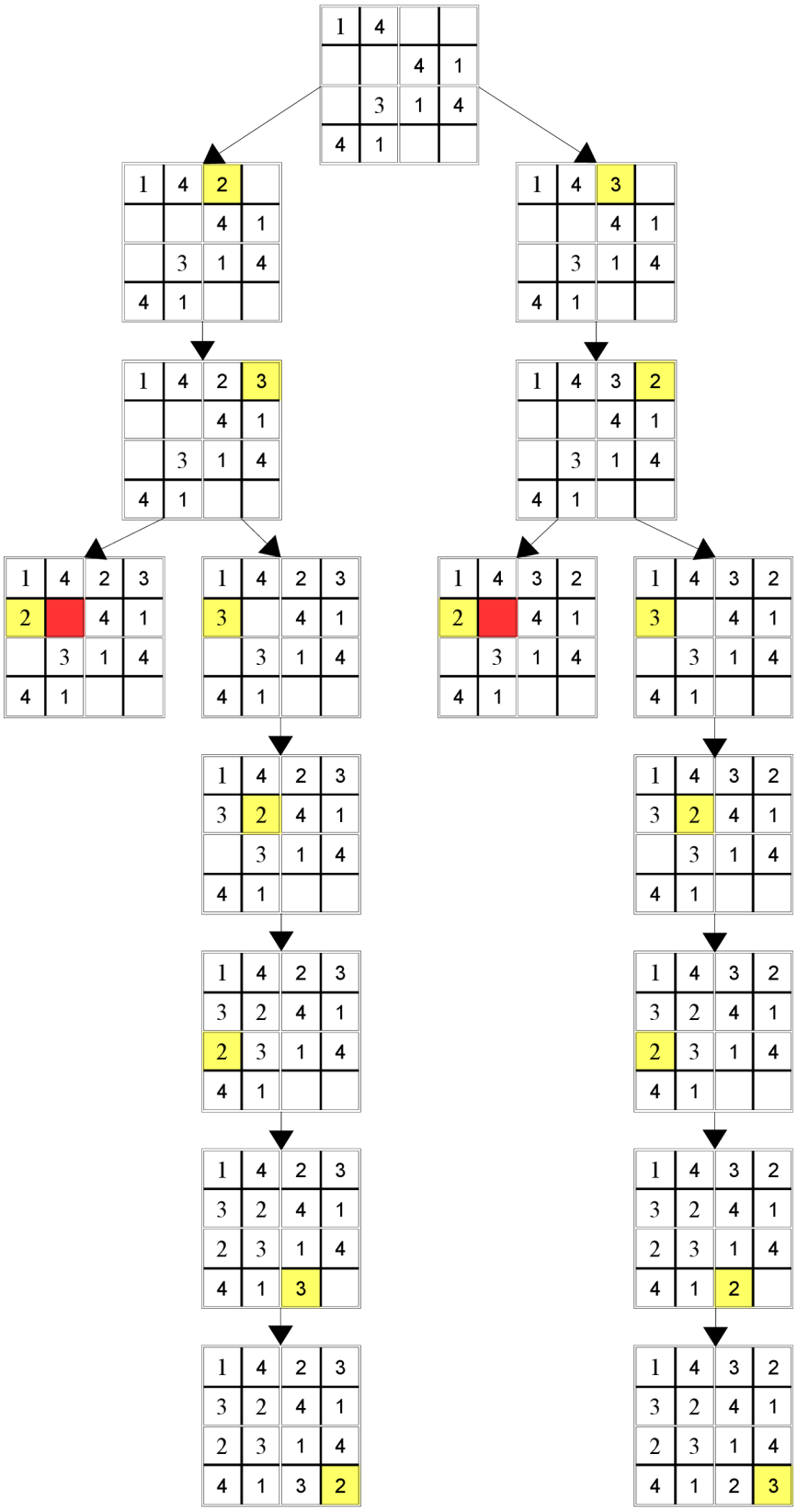
Z pohledu umělé inteligence, lze na sudoku nahlížet jako na úlohu, která je dle [7] definována:

1. počátečním stavem (zadání sudoku),
2. množinou konečných stavů (všechny možné řešení sudoku),
3. množinou operátorů, pomocí nichž se dá mezi stavy přecházet (přidání/odebrání číslice na určité pozici).

Výsledkem takovéto úlohy bude v případě sudoku posloupnost trojic [řada, sloupec, číslice], které udávají, jaká číslice se má do kterého pole dosadit. Řešení sudoku (konečný stav) vznikne postupným dosazením těchto číslic (postupnou aplikací operátorů) na zadání sudoku.

Pro řešení tohoto typu úloh se využívají metody řešení založené na prohledávání stavového prostoru. *Stavový prostor* můžeme dle [8] zobrazit jako stromový graf, kde jsou stavy úlohy reprezentovány uzly grafu a přechody mezi těmito stavy (aplikace operátorů) jsou reprezentovány hranami. Řešení úlohy se získá průchodem grafu z počátečního do cílového stavu.

Ukázku takového grafu pro tzv. dětské sudoku o rozměru 4x4 vidíme na obrázku 4.12. V příkladu se jako operátor uvažuje přidání číslice na určitou pozici – žlutě vyznačená pole. Červeně označená pole jsou ta, pro něž neexistuje vhodná číslice pro doplnění. Na těchto místech končí prohledávání neúspěchem. Cílové stavy jsou v tomto případě dva (což odporuje požadavkům na správně zadané sudoku viz kapitola 2.2). Obě řešení se nachází ve stejné *hloubce,* tj. vzdálenosti do kořenového uzlu. Do kterého cílového stavu se řešitel dostane jako první, závisí na tom, jaká číslice se doplní na první volné pole (ve směru procházení polí zleva doprava, shora dolů) v prvním kroku řešení.



Obrázek 4.12: Ukázka prohledávání stavového prostoru

K řešení těchto úloh se mohou použít tzv. *neinformované metody prohledávání* (nemají informaci o cílovém stavu), které můžeme dle [9] rozdělit z hlediska pořadí, v jakém jsou uzly ve stromu expandovány, na:

1. *Slepé prohledávání do šířky* (*breath-first search - BFS*) – nejprve se expanduje uzel s nejmenší hloubkou.

2. *Slepé prohledávání do hloubky* (*depth-first search - DFS*) – nejprve se expanduje uzel s největší hloubkou.

Oba tyto přístupy pracují se seznamem *OPEN*, který slouží pro uchovávání ještě neexpandovaných uzlů. Metoda *BFS* navíc využívá seznam *CLOSED*, který uchovává informace o již expandovaných uzlech.

Pro porovnání vlastností a hodnocení metod prohledávání stavového prostoru se využívají 4 kritéria, viz [7]:

1. časová náročnost,
2. paměťová náročnost,
3. optimálnost (Je nalezené řešení nejlepším řešením?),
4. úplnost (alezne metoda existující řešení?).

Na hodnotu časové a paměťové náročnosti mají dle [10] vliv následující faktory:

O	počet operátorů
b	faktor větvení – průměrný počet přímých následníků uzlu
d	hloubka, ve které se nachází nejlepší řešení dané úlohy
m	maximální hloubka prohledávání

Tabulka 4.1: Parametry pro určení paměťové a časové náročnosti

Následuje popis vybraných algoritmů a jejich vlastností dle [7].

4.3.1 Algoritmus BFS

1. Sestroj frontu *OPEN* a umísti do ní počáteční uzel.
2. Pokud je fronta *OPEN* prázdná, ukonči prohledávání neúspěchem – úloha nemá řešení.
3. Vybez z fronty *OPEN* první uzel.
4. Pokud je vybraný uzel cílový, ukonči prohledávání jako úspěšné a vrať posloupnost operátorů nebo stavů, které tvoří cestu od kořenového uzlu k uzlu cílovému. Pokud uzel není cílový, pokračuj.
5. Uzel vybraný v předchozím bodě expanduj, všechny jeho přímé následníky ulož do fronty *OPEN* a pokračuj bodem 2.

Vlastnosti algoritmu *BFS* jsou následující:

Časová náročnost	exponenciální $O(b^{d+1})$
Paměťová náročnost	exponenciální $O(b^{d+1})$
Úplnost	ano
Optimálnost	ano

Tabulka 4.2: Vlastnosti algoritmu BFS

Problémem algoritmu *BFS* v jeho základní podobě je opakované generování jednou již generovaných uzlů. K zamezení opakovaného generování se využívá výše zmíněná fronta *CLOSED*, která slouží k uložení již expandovaných uzlů. Upravený algoritmus pracuje tak, že při každém vybrání uzlu *z OPEN,* uloží tento uzel do *CLOSED*. Při expandování uzlu se pak do fronty *OPEN* umísťují pouze ty uzly, které ještě nejsou ani v *OPEN* ani v *CLOSED*. I přesto, že použitím seznamu *CLOSED* dojde k velké redukci expandovaných uzlů, se tento algoritmus pro svou časovou a paměťovou náročnost v praxi téměř nepoužívá.

4.3.2 Algoritmus DFS

Základní algoritmus *DFS* je shodný s algoritmem *BFS* s tím rozdílem, že místo fronty využívá zásobníku *OPEN*, čímž je zajištěno vybírání posledního vloženého uzlu, a tedy prohledávání do hloubky. Algoritmus v základní variantě opět není prakticky použitelný. Je tedy nutné upravit jeho poslední bod tak, aby se po expandování uzlu uložily do zásobníku *OPEN* pouze ty uzly, které se v něm již nenacházejí.

Použitím seznamu *CLOSED* ztrácí metoda svou lineární paměťovou náročnost, proto se tato modifikce v praxi nepoužívá.

Časová náročnost	exponenciální $O(b^{m})$
Paměťová náročnost	lineární O(bm)
Úplnost	ne
Optimálnost	ne

Tabulka 4.3: Vlastnosti algoritmu DFS

4.3.3 Metoda zpětného navracení

Metoda zpětného navracení neboli *backtracking* je speciální variantou prohledávání do hloubky. Narozdíl od algoritmu *DFS* se u zpětného navracení generuje pouze jeden následník uzlu, a teprve pokud by generování skončilo neúspěchem, bude se generovat následník další.

Velkou výhodou této metody je, že má oproti klasickému prohledávání do hloubky minimální paměťovou náročnost. Časová náročnost, úplnost a optimálnost zůstávají shodné s *DFS,* a to: časová náročnost exponenciální, metoda neúplná a neoptimální.

Časová náročnost	exponenciální $O(b^m)$
Paměťová náročnost	konstantní m
Úplnost	ne
Optimálnost	ne

Tabulka 4.4: Vlastnosti algoritmu zpětného navracení

4.3.4 Možné optimalizace

Všechny tyto metody jsou sice funkční, ale v praxi se s nimi v jejich základní podobě téměř nesetkáme. Důvodem je to, že je při jejich aplikaci nutné generovat velký počet uzlů, což vede k neúměrnému zvýšení paměťové i časové náročnosti. Paměťovou náročnost lze částečně zredukovat použitím metody zpětného navracení, ale ani to není dostačující.

Dle [11] můžeme pro konkrétní úlohy vytvořit pravidla, která zmenší rozsah výběru stavů ve stavovém prostoru. Těmto pravidlům říkáme *heuristické informace* a nově vzniklým metodám pak *heuristické metody prohledávání*. Jednou z možností, jak zlepšit vlastnosti metody prohledávání stavového prostoru, je vybírání z uzlů určených pro expanzi vždy ten nejnadějnější (ten, u kterého je nejpravděpodobnější nalezení řešení). K vybrání nejvhodnějšího uzlu se používá např. metoda uspořádání prvků v seznamu *OPEN* dle určitého kritéria. K tomu je ovšem nutné dokázat ohodnotit podle zadaných kritérií všechny uzly. K tomuto účelu se používá tzv. *ohodnocující funkce*.

5 Návrh

V této kapitole je podrobněji rozebrán návrh jednotlivých řešících algoritmů, generátoru zadání, generátoru řešení a návrh výsledné aplikace.

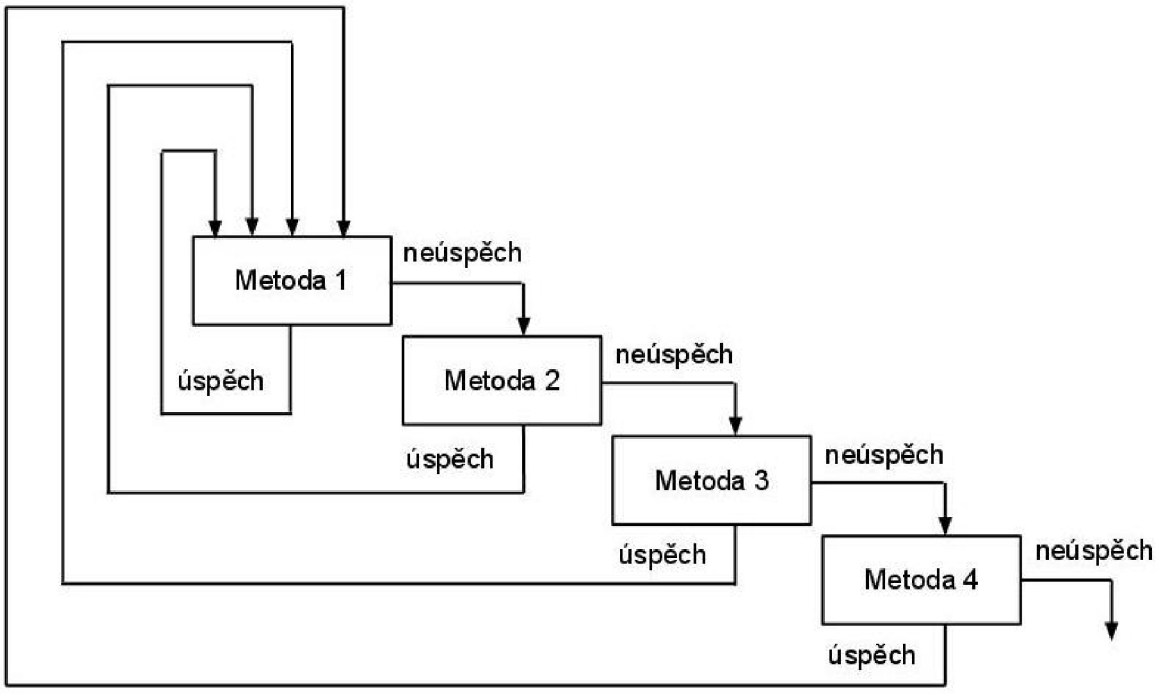
5.1 Řešitelé

5.1.1 Řešitel 1

První řešící algoritmus bude simulovat logické řešení za pomoci metod používaných člověkem. Výhodou tohoto přístupu je, že umožňuje zjistit, zda je sudoku logicky řešitelné, a na základě metod použitých k jeho vyřešení určit jeho obtížnost. Implementace takového algoritmu je proto nutná pro generování validních sudoku.

Na začátku algoritmu je třeba inicializovat hodnoty kandidátních číslic pro všechna pole v sudoku. Dále pak celý algoritmus probíhá v cyklu, dokud se sudoku nevyřeší, nebo dokud jeden cyklus neproběhne celý, aniž by se doplnilo číslo nebo eliminovat kandidát. Jednotlivé metody jsou za sebou seřazeny dle obtížnosti od nejlehčích po nejtěžší tak, že se v každém cyklu provede pouze jedna metoda.

Naznačení průběhu jádra algoritmu je uvedeno na obrázku 5.1. Algoritmus nejprve vyzkouší první (nejjednodušší) metodu, a pokud uspěje, zkusí metodu aplikovat znovu. Pokud neuspěje, vyzkouší další metodu v pořadí atd. Podrobný návrh logických metod je uveden v kapitole 5.1.4.



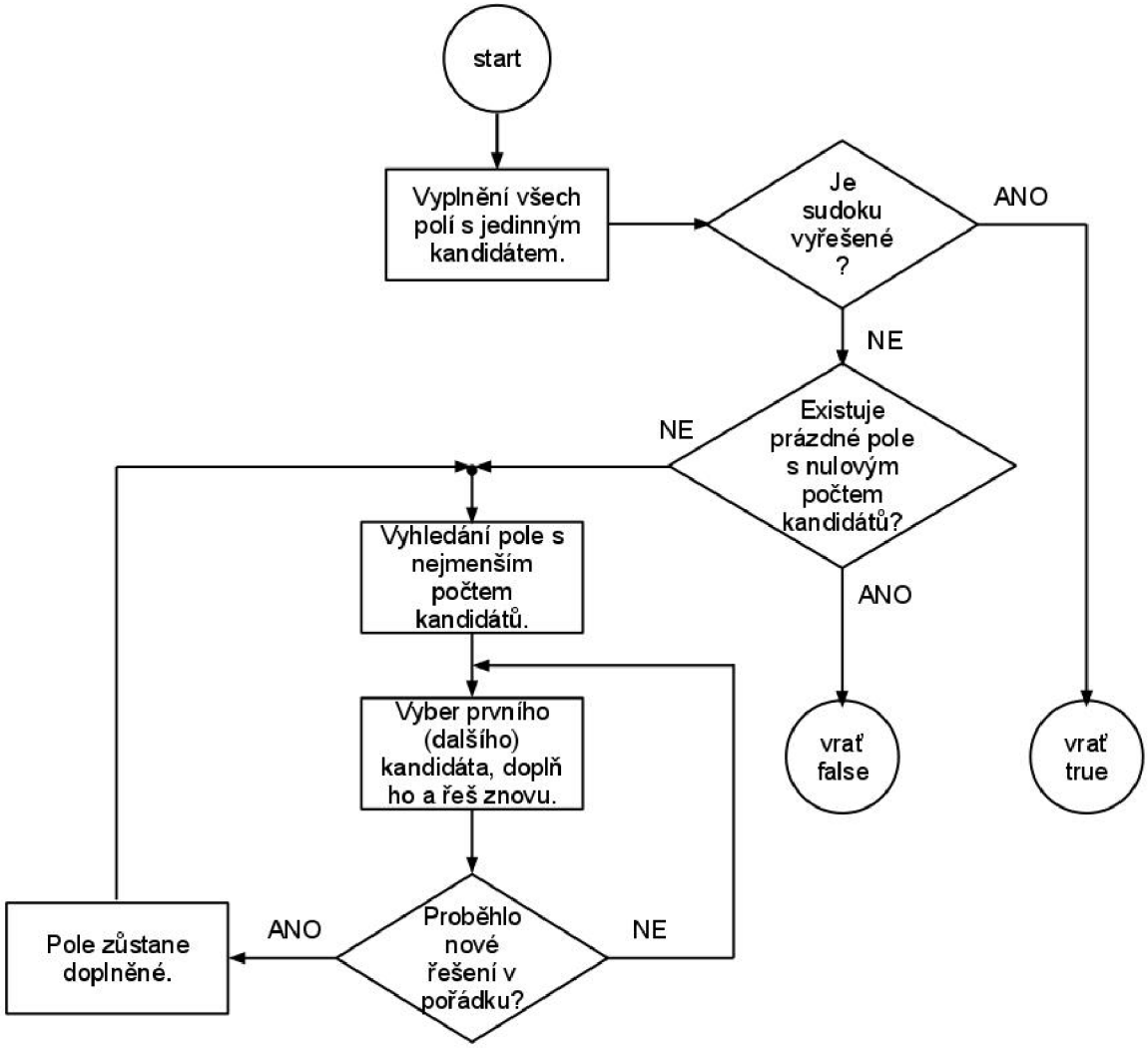
Obrázek 5.1: Návrh algoritmu řešícího sudoku logikou

5.1.2 Řešitel 2

Druhý řešitel bude při řešení využívat metodu prohledávání stavového prostoru. Na začátku tohoto algoritmu se pro každé pole zjistí množina kandidátních číslic, vyplní se pole, pro která existuje jen jeden kandidát, a na výslednou desku se aplikuje vybraná metoda prohledávání stavového prostoru.

Pro tento algoritmus byla zvolena metoda zpětného navracení (viz kapitola 4.3.3), která má velmi malou paměťovou náročnost, protože generuje pouze jednoho následníka a teprve v případě neúspěchu se generuje následník další. V praxi to znamená, že algoritmus vybere prázdné pole, náhodně vybere jednu kandidátní číslici, tu dosadí do desky a pokračuje v řešení. Pokud dojde k místu, kdy je sudoku dále neřešitelné, vrátí chybu, do původního políčka dosadí další kandidátní číslici a řešení probíhá znovu. Algoritmus využívá rekurze, jeho průběh je zobrazen na obrázku 5.2.

Tento řešitel bude na rozdíl od klasického zpětného navracení vybírat pole pro dosazování ne podle jejich pořadí, ale podle počtu kandidátů. Dojde tím k úspoře řešícího času, jelikož u pole s nejmenším počtem kandidátů je největší pravděpodobnost, že bude vybrané číslo správné a že se algoritmus nebude muset vracet.



Obrázek 5.2: Návrh algoritmu řešícího sudoku metodou zpětného navracení

5.1.3 Řešitel 3

Třetí řešící algoritmus by měl vhodně zkombinovat předchozí dva. Sudoku bude tedy řešit logikou a teprve v případě, že nelze dále logicky pokračovat, dosadí číslici náhodně. Poté se sudoku řeší znovu logicky. Řešitel bude pracovat rekurzivně, na podobném principu jako předchozí algoritmus. V případě úspěšného vyřešení vrátí *true*, v případě chyby v sudoku vrátí *false*. Algoritmus by měl zvládnout vyřešit libovolné sudoku v lepším čase, než pokud by bylo použito pouze zpětné navracení.

5.1.4 Podrobný návrh vybraných logických metod

Jelikož implementace metod pro přímé doplnění kandidáta je triviální, bude v této podkapitole uveden pouze návrh algoritmů pro eliminaci kandidátů. Jednotlivé algoritmy probíhají následovně:

Pointing

1. Pro každý čtverec ber postupně číslice od 1 do 9 a pro každou z nich zkontroluj, zda je již ve čtverci vyplněná.
2. Pokud není, ulož pozice výskytu kandidáta s vybraným číslem a zároveň počítej, kolikrát se ve čtverci kandidát vyskytuje.
3. Pokud se kandidát ve čtverci vyskytuje dvakrát nebo třikrát, zkontroluj, zda se všichni vyskytují v jednom řádku nebo sloupci.
4. Pokud ano, eliminuj tohoto kandidáta ve zbytku řádku nebo sloupce a vrať *true*.
5. Pokud ne, pokračuj v kontrole dalšího čtverce.
6. Pokud byly zkontrolovány všechny čtverce a k žádné eliminaci nedošlo, vrať *false*.

Claiming

1. Pro každý řádek ber postupně číslice od 1 do 9 a pro každou z nich zkontroluj, zda je již v oblasti vyplněná.
2. Pokud není, ulož pozice výskytu kandidáta s vybraným číslem a zároveň počítej, kolikrát se v řádku kandidát vyskytuje.
3. Pokud se kandidát v řádku vyskytuje dvakrát nebo třikrát, zkontroluj, zda se všichni vyskytují v jednom čtverci.
4. Pokud ano, eliminuj tohoto kandidáta ve zbytku čtverce a vrať *true*.
5. Pokud ne, pokračuj v kontrole dalšího řádku.
6. Po kontrole všech řádků, pokračuj kontrolou všech sloupců.
7. Pokud byly zkontrolovány všechny řádky i sloupce a k žádné eliminaci nedošlo, vrať *false*.

Naked subsets

1. V každé oblasti procházej postupně jednotlivá pole, a pokud pole není vyplněné, uchovej počet jeho kandidátů a jeho souřadnice.
2. Pokud bylo prázdné pole nalezeno, vyhledej v oblasti ostatní pole se stejným počtem kandidátů.

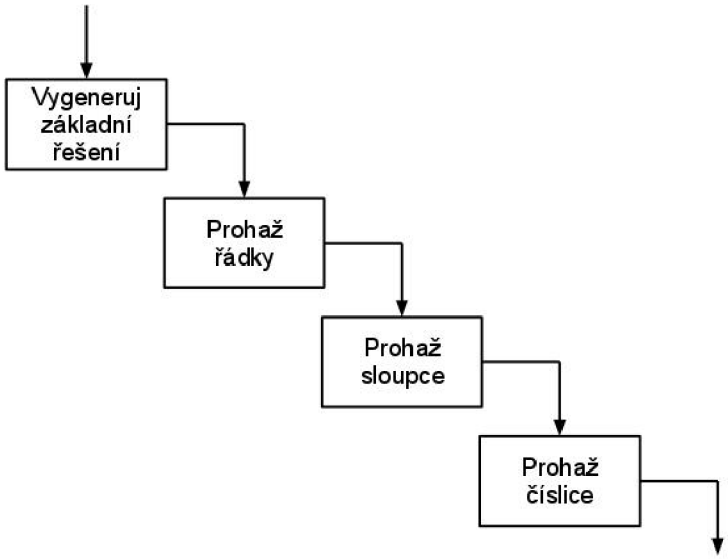
3. Porovnej kandidáty v jednotlivých polích, a pokud nalezneš více polí se shodnými kandidáty, jejichž počet se rovná počtu nalezených shodných polí, eliminuj tyto kandidáty ve zbytku aktuální prohledávané oblasti.
4. Pokud byl nějaký kandidát eliminován, vrať *true*. Pokud k eliminaci nedošlo, pokračuj v kontrole dalších polí v oblasti.
5. Pokud byli všechny oblasti zkontrolovány a žádný kandidát nebyl eliminován, vrať *false*.

X-Wings

1. Procházej postupně řádky (sloupce). Aktuální řádek (sloupec) označ jako řádek 1 (sloupec 1) a zjisti, kolikrát se v něm vyskytují jednotliví kandidáti.
2. Pokud se některý kandidát v řádku (sloupci) vyskytuje jen dvakrát, zjisti, v jakých sloupcích (řadách) se tento kandidát vyskytuje.
3. Nyní vyber jeden ze zjištěných sloupců (řad) a označ ho jako sloupec 1 (řada 1). Procházej zbytek sloupce 1 (řady 1) a hledej pole, kde se aktuální kandidát také vyskytuje.
4. Pokud takové pole nalezneš, zkontroluj, jestli se na souřadnicích [řada 1][sloupec 2] ([řada 2][sloupec 1]) nevyskytuje hledaný kandidát.
5. Pokud ano, zkontroluj zbytek sloupce 2 (řady 2), jestli se v ní kandidát vyskytuje jen dvakrát, a pokud ano, eliminuj kandidáta ve zbytku sloupců 1 a 2 (řádků 1 a 2).
6. Pokud byl nějaký kandidát eliminován, vrať *true*. Pokud ne, pokračuje v kontrole dalšího řádku (sloupce).
7. Pokud byly všechny řádky (sloupce) zkontrolovány a nedošlo k eliminaci, vrať *false*.

5.2 Generátor řešení

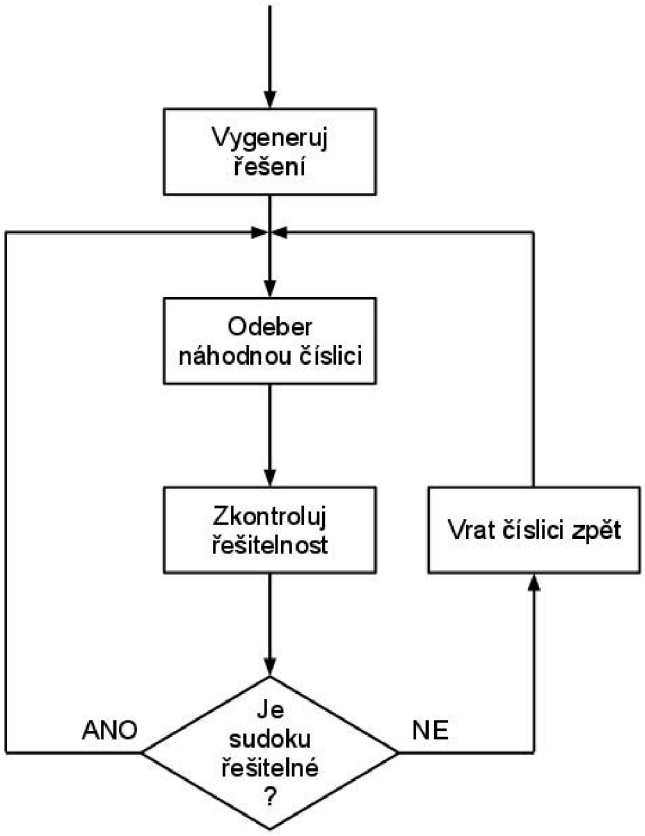
Generátor bude využívat tzv. základního řešení (viz kapitola 3.1) a generování bude probíhat podle algoritmu na obrázku 5.3.



Obrázek 5.3: Návrh generátoru řešení

5.3 Generátor zadání

Pro generování sudoku byla zvolena varianta vycházející z vyplněného sudoku. Generátor bude tedy nejprve generovat zadání a poté z něj náhodně odebírat číslice. Po každém odebrání provede kontrolu řešitelnosti sudoku. Pokud je řešitelné, zůstane číslo odebrané, pokud ne, vrátí se číslo zpět na původní pozici (viz obrázek 5.4). K ukončení mazání číslic dojde tehdy, když se buď vymaže maximální možný počet polí (tzn. zbude sudoku pouze s 17 číslicemi), nebo když se v určitém množství po sobě jdoucích kroků nepodaří vymazat žádná číslice.



Obrázek 5.4: Návrh generátoru zadání

Nakonci generování se sudoku otestuje na obtížnost. Pokud je shodná s požadovanou, tak se generování ukončí, pokud ne, probíhá generování znovu.

5.4 Třída sudoku

Hra sudoku bude v programu reprezentována třídou `Sudoku`, která bude obsahovat třídu představující řešení sudoku – `SudokuSolution` a třídu představující zadání sudoku – `SudokuProblem`.

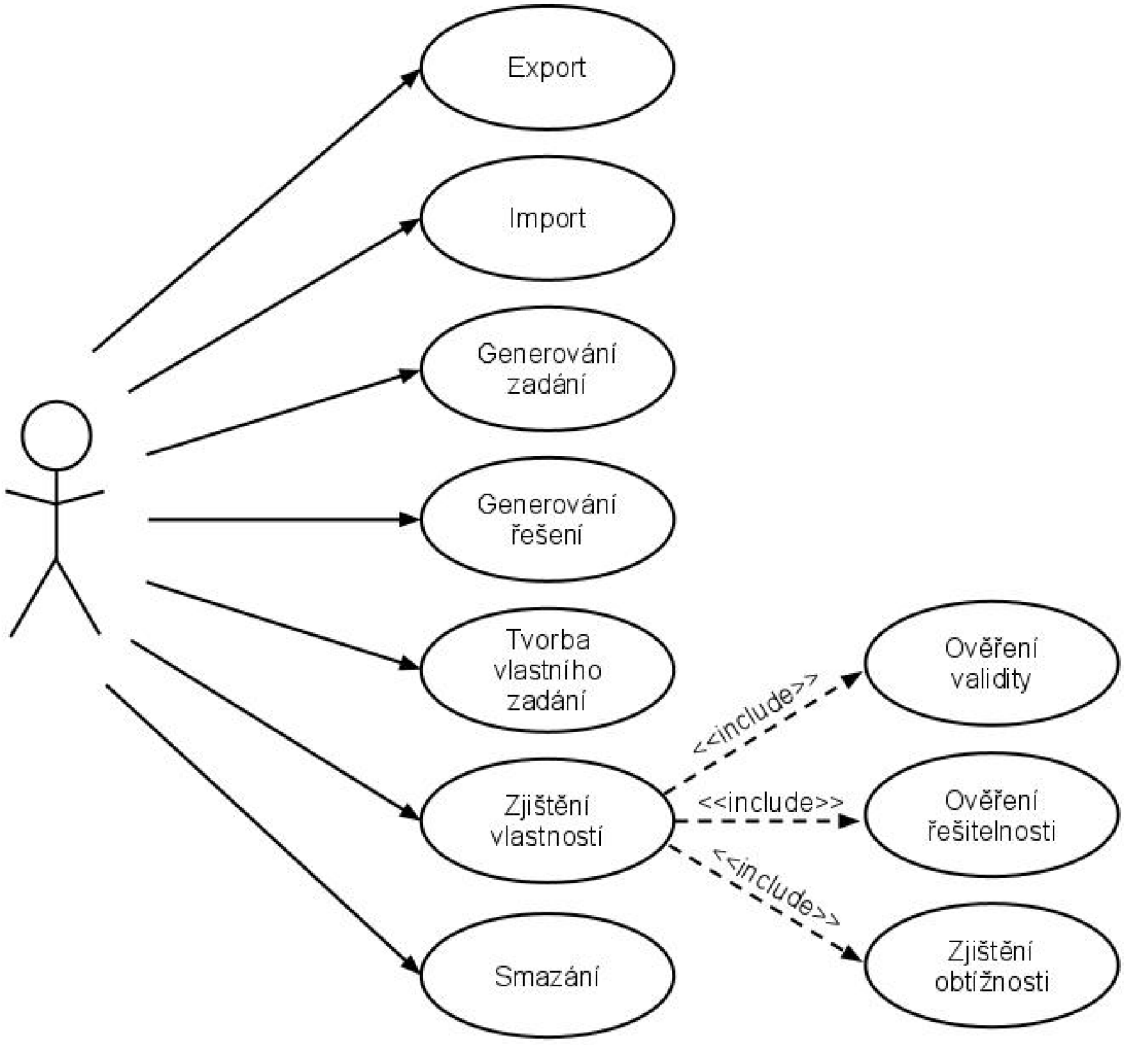
Třídu `SudokuProblem` bude tvořit dvourozměrné pole pro uložení zadání, dále bude obsahovat informaci o obtížnosti zadání a pole, které umožní práci s kandidáty v jednotlivých polích – s jejich počtem a jednotlivými hodnotami. Třída `SudokuSolution` bude reprezentovat vyřešené sudoku.

5.5 Aplikace

Výsledná aplikace by měla uživateli poskytovat jednoduchý nástroj jednak pro pohodlné generování a upravování, tak pro vytváření vlastních sudoku. Dále má uživateli poskytnou možnost exportování a importování her.

5.5.1 Use-case diagram

Diagram případů užití, z něhož následně vyhází návrh uživatelského rozhraní aplikace, je uveden na obrázku 5.5.



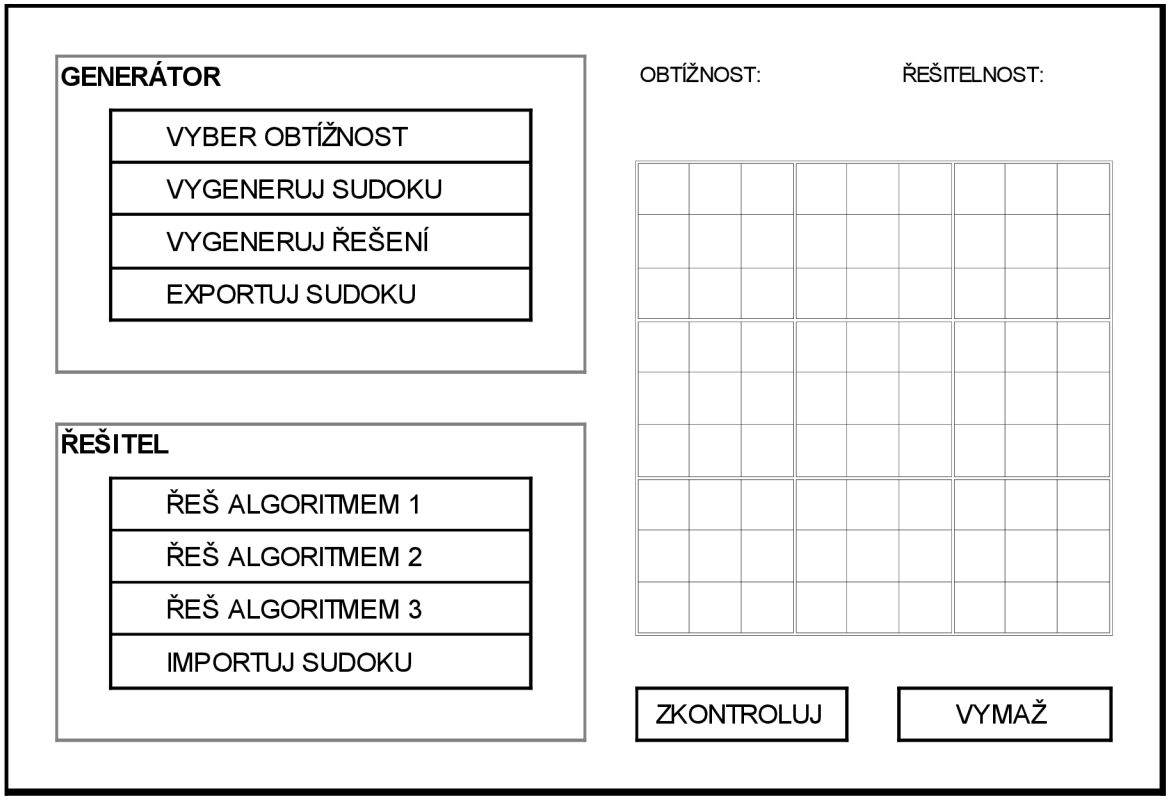
Obrázek 5.5: Diagram případů užití

Popis jednotlivých případů užití:
Export – exportování sudoku do souboru xml
Import – importování sudoku ze souboru xml
Generovaní zadání – vygenerování hry zadané obtížnosti

Generovaní řešení – vygenerování řešení hry
Tvorba vlastního zadání – ruční vytvoření vlastního sudoku
Zjištění vlastností – zahrnuje zjištění obtížnosti a řešitelnosti a ověření validity sudoku
Smazání – smazání sudoku

5.5.2 Uživatelské rozhraní

Návrh hlavního okna aplikace je uveden na obrázku 5.6. Tvoří ho sada tlačítek pro generování a řešení sudoku, tabulka pro zobrazení sudoku, tlačítko pro zjištění vlastností sudoku a tlačítko pro jeho vymazání.



Obrázek 5.6: Návrh vzhledu aplikace

5.6 Import a export

Jelikož aplikace sama neslouží jako hra, ale pouze jako generátor validních sudoku zadání, je důležité, aby podporovala export sudoku do vhodného formátu tak, aby mohly být hry dále použity. Pro tento účel byl zvolen export do formátu xml, který je obecně rozšířený, přenostitelný a v případě potřeby snadno editovatelný i běžným uživatelem počítače.

Xml soubor se sudoku bude obsahovat jak zadání, tak i výsledné řešení, aby mohly být soubory použity v další aplikaci bez nutnosti v ní implementovat řešitele sudoku. Dále bude soubor obsahovat obtížnost sudoku (pokud je známá). Struktura ukázkového xml souboru ve uvedana v příloze 1.

6 Implementace

Aplikace byla implementována za použití nástroje Qt, jehož součástí je QtDesigner určený pro tvorbu programů s grafickým uživatelským rozhraním, a dále poskytuje celou řadu knihovních funkcí, např. pro práci s xml soubory, které umožňují jejich efektivní tvorbu, čtení a úpravu.

6.1 Uživatelské rozhraní

Pro tvorbu uživatelského rozhraní byl využit výše zmíněný QtDesigner. Pro zobrazení sudoku byla využita třída QTableWidgetItem.

6.2 Generátor a řešitelé

Generátor řešení generuje sudoku ve třech obtížnostech – lehké, střední a těžké. Lehké sudoku jdou vyřešit pouze za použití metod *Naked* a *Hidden singles,* středně těžké sudoku vyžaduje použití metod *Pointing* a *Claiming* a pro vyřešení těžkého sudoku je nutné ovládat metody *Naked subsets* a *X- Wings*. Schopností řešit sudoku těmito vyjmenovanými metodami také disponuje logický řešitel.

6.3 Import a export

Pro implementaci importování a exportování souborů byl využit modul QtXml, který umožňuje zpracovávat xml soubory pomocí rozhraní DOM (Document Object Model). Pro tento účet existuje knihovna QDomDocument, která poskytuje funkce pro pohodlnou editaci souborů.

7 Testování algoritmů

Testování algoritmů proběhlo na množině testovacích zadání různých obtížností, které byly vygenerovány jak samotnou aplikací, tak získány z veřejně dostupných on-line generátorů.

V průběhu vývoje programu byly pro testování funkčnosti algoritmů také použity hry s pouze náhodně skrytými číslicemi, což umožnilo lépě otestovat řešitele č. 2, který využívá metodu zpětného navracení.

7.1 Testování časové náročnosti

Pro ověření a porovnání časové náročnosti jednotlivých řešících algoritmů byly použity funkce z knihovny QTime, pomocí nichž byla měřena doba jejich provádění. Jelikož nejmenší jednotka, ve které tato knihovna umožňuje měřit čas jsou milisekundy, provedlo se vždy každé řešení tisíckrát a výsledný čas se pak tísíci podělil, aby se dosáhlo přesnějšího výsledku. K testování každé obtížnosti se použilo 50 her. Důležité výsledky testování jsou uvedeny v tabulce 7.1 a kompletní hodnoty testů jsou uvedeny v příloze 2.

Nejlepších výsledků dosáhl při testování řěšitel 1, který využívá logických postupů. Nejvýraznější je jeho převaha u řešení jednoduchých sudoku, které zvládne oproti ostatním řešitelům vyřešit v téměř polovičním čase. Střední a těžká sudoku řeší pouze trochu rychleji, než kombinovaný řešitel 3. Řešitel 2, který řeší sudoku pouze zpětným navracením, je pro jiná než lehká sudoku téměř nepoužitelný, jelikož tato sudoku řeší v průměru přibližně 7x delší dobu a často u něj dochází rapidnímu nárůstu řešící doby až na jednotky či desítky minut. Konkrétně u středních sudoku tento případ nastal 5 krát z celkových 50 a u těžkých sudoku dokonce 10 krát. Nejvyrovnanějších výsledků dosahuje u lehké a těžké obtížnosti řešitel 3 a u střední řešitel 1. U druhého řešitele lze pozorovat, jak se doba řešení výrazně mění v závislosti na řešeném sudoku a na jeho vlastnostech. Např. u středních sudoku byla nejmenší doba řešení 7,02 ms a nejdelší (s výjimkou těch extrémních) 96,65ms.

	Čas řešení - Řešitel 1 [µs]			**Čas řešení - Řešitel 2 [µs]**			**Čas řešení - Řešitel 3 [µs]**		
	Lehká	**Střední**	**Těžká**	**Lehká**	**Střední**	**Těžká**	**Lehká**	**Střední**	**Těžká**
Průměrná hodnota:	666	1203	1487	1276	30950	22019	1118	1343	1648
Minimální hodhota:	223	963	1076	980	7020	5850	926	1101	1236
Maximální hodnota:	1184	1496	3003	1706	96650	78300	1671	1748	2582
Rozptyl:	916	533	1927	726	89630	72450	745	647	1346

Tabulka 7.1: Výsledky testování časové složitosti

8 Závěr

8.1 Zhodnocení implementovaných algoritmů

8.1.1 Generárory

Implementovaný generátor zadání i generátor řešení pracují v pořádku a umožňují generovat sudoku zvolené obtížnosti, která je ale shora limitována složitostí implementovaných logických metod řešení. Jelikož generátor zadání pracuje na principu generování náhodných sudoku až do té doby, než se vygeneruje sudoku zvolené obtížnosti (viz kapitola 3.3), může generování složitějších sudoku trvat i několik vteřin. Algoritmy by nebylo obtížné upravit pro generování sudoku jiných rozměrů.

8.1.2 Řešitel 1

Logický řešitel si bez problémů poradí s běžně dostupnými sudoku lehké a střední obtížnosti. U těžších sudoku vzniká problém v rozdílném množství implementovaných složitých metod u generátorů použitých pro testování. V několika případech se i řešiteli povedlo odhalit nedostatky některých on-line generátorů, když zjistil, že sudoku vydávaná za těžká jdou vyřešit pouze za použití metod *Naked Singles* a *Hidden Singles*.

8.1.3 Řešitel 2

Řešení jen pomocí zpětného navracení se dle výsledků testování nejeví jako příliš vhodné. Síla implementovaného algoritmu je velmi liminována obtížností řešeného sudoku a počtem viditelných polí. V původním návrhu se počítalo s tím, že se vždy před tipováním doplní pouze ta pole, v nichž se vyskytuje jedinný kandidát. V tomto případě doba výpočtu rapidně vzrostla, už pokud bylo v sudoku vyplněno méně jak 40 polí. Algoritmus byl dodatečně rozšířen o část, která před tipováním vyplní všechna pole, v nichž může být určitý kandidát jako na jedinném poli v oblasti (metoda *Hidden singles*). Výkon algoritmu se poté sice zvýšil, ale ve výsledku je rychlost řešení tímto způsobem stále nedostačující.

8.1.4 Řešitel 3

Jak bylo v návrhu předpokládáno, jeví se použití kombinovaného řešitele (převážně z hlediska časové náročnosti) jako nejrozumější řešení pro sudoku, která nejsou logicky řešitelná, nebo pro taková, s jejichž obtížností si logický řešitel sám o sobě neporadí. Algoritmus vyřeší prakticky každé sudoku, ale u logicky neřešitelných sudoku je čas vyřešení velmi závislý na tom, jak jsou „zpřetrhány“ logické souvislosti mezi políčky.

8.2 Možná rozšíření

Jelikož v logickém řešiteli nejsou implementovány všechny pokročilé metody řešení sudoku, generuje aplikace hry pouze v omezeném rozsahu obtížnosti. Bylo by zajímavé implementovat i zbývající, i když méně často používané metody, a pokusit se o generování extrémě složitých sudoku nebo se zaměřit na generování minimálních sudoku pouze se 17 číslicemi.

Aplikace funguje pouze jako generátor a řešitel, ale neumožňuje pohodlné řešení sudoku uživatelem. Pro povýšení aplikace na hru by bylo nutné umožnit vpisování kandidátů pro jednotlivá políčka a implementovat funkci, která by uměla uživateli pomoci (např. vyplněním vybraného políčka nebo zobrazením chyby), pokud by již neuměl postoupit v řešení.

Literatura

[1] ROYLE, Gordon. The University of Western Australia [online]. [cit. 2010-05-15]. Minimum Sudoku. Dostupné z WWW: <http://units.maths.uwa.edu.au/~gordon/sudokumin.php>.

[2] DAVIS, Tom. The Mathematics of Sudoku. October 12, 2008, [cit. 2010-05-14]. Dostupný z WWW: <http://www.geometer.org/mathcircles/sudoku.pdf>.

[3] FELGENHAUER, Bertram; JARVIS, Frazer. Frazer Jarvis [online]. June 20, 2005 [cit. 2010-05-15]. Sudoku enumeration problems. Dostupné z WWW: <http://www.afjarvis.staff.shef.ac.uk/sudoku/>.

[4] LEWIS, Rhyd. Metaheuristics can solve sudoku puzzles. Journal of Heuristics [online]. August 2007, Volume 13, Issue 4, [cit. 2010-05-10]. Dostupný z WWW: <http://www.rhydlewis.eu/papers/META_CAN_SOLVE_SUDOKU.pdf>.

[5] Sudoku Online.us [online]. 2010 [cit. 2010-05-14]. Dostupné z WWW: <http://www.sudokuonline.us>.

[6] Sudoku Snake [online]. c2010 [cit. 2010-05-14]. Coloring. Dostupné z WWW: <http://www.sudokusnake.com/coloring.php>.

[7] ZBOŘIL, František; ZBOŘIL, František. Základy umělé inteligence : IZU. Brno : Fakulta informačních technologií, 2008. 142 s.

[8] ZBOŘIL, František; HANÁČEK, Petr. Umělá inteligence. Vydání druhé. Brno : Nakladatelství Vysokého učení technického v Brně, 1991. 125 s. ISBN 80-214-0329-2.

[9] MAŘÍK, Vladimír, et al. Umělá inteligence (1). Vydání 1. Praha : Academia, 1993. 264 s. ISBN 80-200-0496-3, ISBN 80-200-0502-1.

[10] MACEK, Jiří. Grafické animace metod řešení úloh. [s.l.], 2007. 44 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z WWW: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=5728>.

[11] SEDLÁČEK, Václav. Umělá inteligence : [Díl] 1., Úvod, metody řešení úloh, rezoluční metoda. 1. vyd. Praha : Státní pedagogické nakladatelství, 1983. 203 s.

Seznam obrázků

Seznam tabulek

Seznam příloh

Příloha 1 – ukázkový XML soubor

```
<sudoku difficulty="easy">
<problem>
<r><c>0</c><c>2</c><c>3</c><c>4</c><c>5</c><c>0</c><c>7</c><c>0</c><c>9</c>
</r>
<r><c>7</c><c>0</c><c>9</c><c>0</c><c>0</c><c>3</c><c>0</c><c>5</c><c>0</c>
</r>
<r><c>4</c><c>5</c><c>0</c><c>0</c><c>8</c><c>9</c><c>1</c><c>2</c><c>3</c>
</r>
<r><c>2</c><c>3</c><c>4</c><c>0</c><c>6</c><c>7</c><c>0</c><c>9</c><c>1</c>
</r>
<r><c>0</c><c>6</c><c>7</c><c>8</c><c>0</c><c>1</c><c>2</c><c>0</c><c>4</c>
</r>
<r><c>8</c><c>0</c><c>1</c><c>2</c><c>3</c><c>4</c><c>5</c><c>6</c><c>7</c>
</r>
<r><c>3</c><c>4</c><c>5</c><c>0</c><c>7</c><c>8</c><c>9</c><c>0</c><c>2</c>
</r>
<r><c>6</c><c>0</c><c>8</c><c>0</c><c>1</c><c>2</c><c>0</c><c>4</c><c>5</c>
</r>
<r><c>0</c><c>0</c><c>2</c><c>3</c><c>4</c><c>5</c><c>6</c><c>7</c><c>0</c>
</r>
</problem>
<solution>
<r><c>1</c><c>2</c><c>3</c><c>4</c><c>5</c><c>6</c><c>7</c><c>8</c><c>9</c>
</r>
<r><c>7</c><c>8</c><c>9</c><c>1</c><c>2</c><c>3</c><c>4</c><c>5</c><c>6</c>
</r>
<r><c>4</c><c>5</c><c>6</c><c>7</c><c>8</c><c>9</c><c>1</c><c>2</c><c>3</c>
</r>
<r><c>2</c><c>3</c><c>4</c><c>5</c><c>6</c><c>7</c><c>8</c><c>9</c><c>1</c>
</r>
<r><c>5</c><c>6</c><c>7</c><c>8</c><c>9</c><c>1</c><c>2</c><c>3</c><c>4</c>
</r>
<r><c>8</c><c>9</c><c>1</c><c>2</c><c>3</c><c>4</c><c>5</c><c>6</c><c>7</c>
</r>
<r><c>3</c><c>4</c><c>5</c><c>6</c><c>7</c><c>8</c><c>9</c><c>1</c><c>2</c>
</r>
<r><c>6</c><c>7</c><c>8</c><c>9</c><c>1</c><c>2</c><c>3</c><c>4</c><c>5</c>
</r>
<r><c>9</c><c>1</c><c>2</c><c>3</c><c>4</c><c>5</c><c>6</c><c>7</c><c>8</c>
</r>
</solution>
</sudoku>
```

Příloha 2 - Tabulka podrobných výsledků testu časové náročnosti

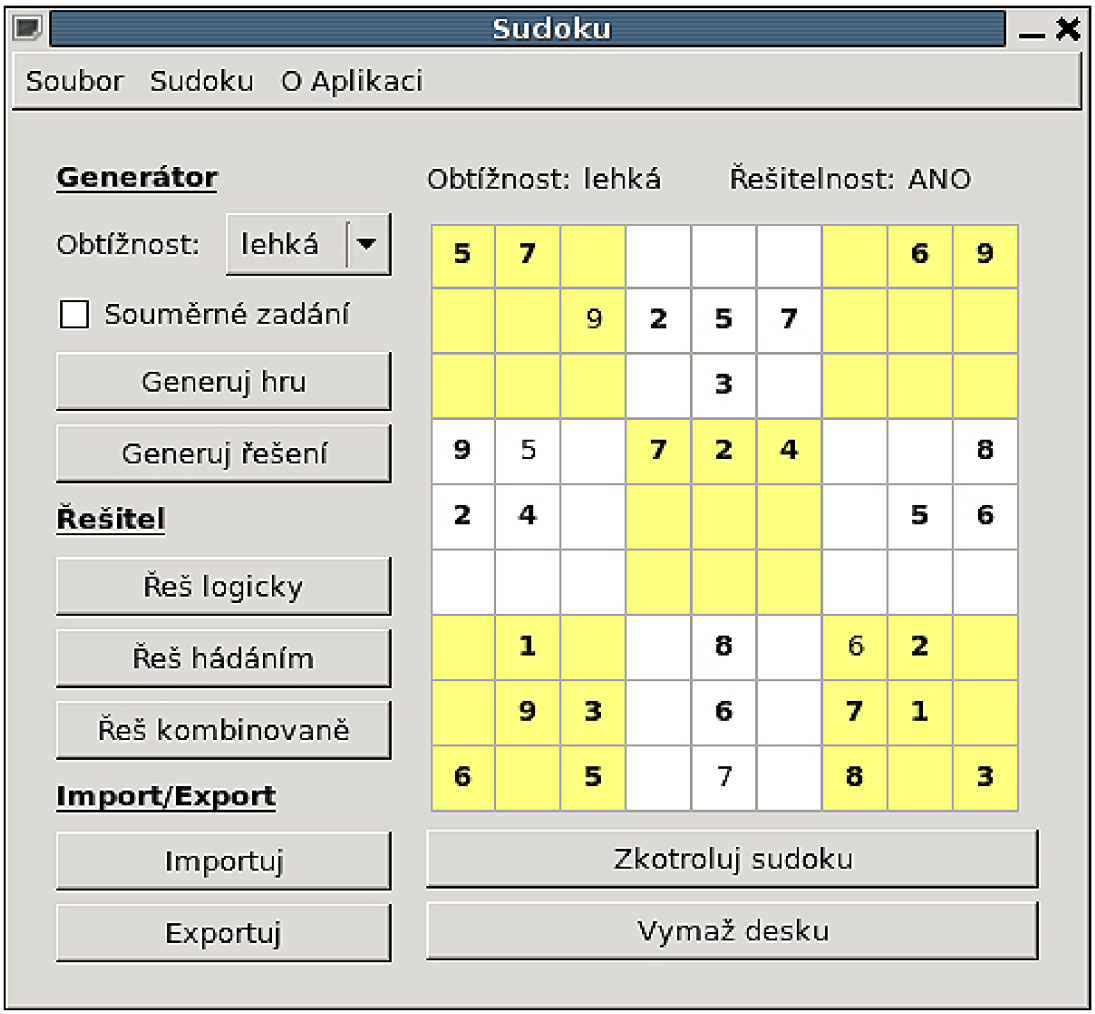
Čas řešení - Řešitel 1 [µs]			Čas řešení - Řešitel 2 [µs]			Čas řešení - Řešitel 3 [µs]		
Lehká	Střední	Těžká	Lehká	Střední	Těžká	Lehká	Střední	Těžká
977	1438	1374	1247	36950	25070	1075	1375	1688
1083	1383	1204	1434	15600	7620	1103	1343	1271
1072	1156	1589	1158	22740	34990	1025	1382	1840
903	1235	1300	1262	95830	19090	1288	1304	2174
1000	1426	1465	1414	20850	---	1191	1307	1382
1144	1022	1299	1321	11780	20110	926	1738	1473
1184	1092	1076	1283	19460	17970	1057	1748	1774
989	1141	1204	1282	17510	13220	1035	1356	1797
1106	1007	1312	1535	91830	---	1050	1261	1667
894	1266	1586	1184	49120	50660	1037	1301	1331
970	1233	1509	1349	26710	---	1031	1537	1550
986	1318	1114	1168	23990	12190	1117	1333	1986
1069	1190	1767	1461	12861	18580	1282	1342	1852
902	1047	1385	1414	27042	16450	1168	1216	1592
1056	1443	1296	1240	17360	45890	1101	1488	1679
870	1148	1275	1171	17060	---	964	1182	1490
861	1106	1314	1357	57100	42180	983	1223	1534
948	1175	1395	1343	27290	e prog	1099	1140	1518
1091	1221	1496	1700	23480	30160	1073	1474	1397
875	1035	1219	1214	26670	---	1012	1506	1417
1156	1180	1300	1511	59850	8670	1136	1213	1416
933	1191	1454	1234	7070	15260	1040	1228	1699
878	1246	1426	980	25150	---	1166	1149	1402
1020	1223	1285	1171	17100	7210	1358	1306	1312
865	1225	1330	1074	21560	8280	1190	1144	1570
1007	1214	1356	1023	7020	17250	1236	1136	1703
1016	1158	2151	1361	---	9960	1182	1321	1859
883	1031	3003	1175	32160	9450	1069	1186	1521
908	1284	2546	1078	15080	---	944	1232	1430

	931	1274	1361	1199	22110	30670	1208	1238	1797
	1028	1025	1362	1110	10462	19550	1671	1551	1737
	816	1251	1478	1233	10811	11140	1123	1148	1357
	223	1129	1739	1364	25510	44900	1160	1454	1508
	795	1444	1270	1320	---	78300	1054	1711	1591
	858	1286	1506	1210	22330	16550	1143	1540	1594
	865	1356	1113	1108	23590	8400	1238	1314	1236
	1072	1251	1371	1191	10830	6750	1085	1339	2582
	766	1224	1451	1093	13820	41150	1164	1402	1407
	776	1115	1576	1209	28840	---	950	1489	1889
	897	1269	1247	1562	78570	26740	1132	1294	1926
	992	1496	1573	1105	---	20140	1251	1297	2365
	1040	1190	1317	1309	8640	---	1106	1559	1447
	995	1137	1552	1706	31100	27880	1106	1244	1615
	992	963	1347	1182	9230	5850	1230	1439	1431
	862	1206	1620	1158	---	6060	1167	1151	1412
	772	1174	1717	1327	---	18820	1053	1198	1781
	1003	1177	1413	1071	30410	51510	941	1465	1425
	968	1062	2013	1415	70900	17830	974	1101	2101
	893	1197	1677	1270	72750	8610	949	1498	2191
	901	1108	1617	1516	96650	9670	1257	1253	1681
Průměrná hodnota:	666	1203	1487	1276	30950	22019	1118	1343	1648
Minimální hodhota:	223	963	1076	980	7020	5850	926	1101	1236
Maximální hodnota:	1184	1496	3003	1706	96650	78300	1671	1748	2582
Rozptyl:	916	533	1927	726	89630	72450	745	647	1346

Pozn.: Nevyplněná pole (resp. pole vyplněná pomlčkami) znamenají, že doba řešení tohoto sudoku byla neúměrně dlouhá (řádově desítky minut), a nebyla proto přesně změřena.

Příloha 3 – Manuál k aplikaci Sudoku

1. Ovládání aplikace



Popis jednotlivých tlačítek:

Vygeneruj zadání– vygeneruje sudoku zvolené obtížnosti a vypíše ho do tabulky.
Vygeneruj řešení – vygeneruje vyplněné sudoku a vypíše jej do tabulky.
Řeš logicky – vyřeší sudoku z tabulky za pomoci logických postupů. Pokud nejde sudoku logicky vyřešit, nabídne uživateli vypsání pouze těch polí, které logicky vyplnit jdou.
Řeš hádáním – vyřeší sudoku z tabulky pomocí optimalizované metody zpětného navracení. Umožní vyřešit i logicky neřešitelné sudoku. Pokud má sudoku málo vyplněných políček, upozorní uživatele na to, že by mohlo řešení trvat neúměrně dlouho (záleží jak jsou „vazby“ mezi vyplněnými políčky silné) a nabídne mu řešení zrušit.
Řeš kombinovaně – vyřeší sudoku kombinovanou metodou. Ta používá logických postupů a v případě nemožnosti v řešení dále postoupit logikou, vyplní nějaké pole náhodně.

Zkontroluj – zkontroluje sudoku z tabulky – všechna nově vyplněná pole přidá do zadání (po provedení kontroly budou vypsány tučně), zjistí řešitelnost sudoku, jeho validitu a obtížnost.
Vymaž – umožňuje uživateli vymazat sudoku. Nabídne mezi výběrem vymazání celé tabulky, nebo vymazání s ponecháním zadání.
Importuj – nabídne uživateli importovat zadání sudoku z xml souboru.
Exportuj – nabídně uživateli exportovat zadání do xml souboru.

Struktura horního menu

Aplikaci lze ovládat také z horního menu, jehož struktura je následující:

- Soubor
  - Konec
- Sudoku
  - Generátor
    - Generuj hru
    - Generuj řešení
  - Řešitel
    - Řeš logicky
    - Řeš hádáním
    - Řeš kombinovaně
  - Zkontroluj
  - Vymaž
  - Import
  - Export
- O aplikaci
  - O aplikaci

2. Import a export sudoku

Program umožňuje importování a exportování sudoku z/do xml souborů jak pro jejich uchovávání, tak kvůli možnosti dalšího použití, např. v jiné aplikaci. Import a export se provádí příslušnými tlačítky. Poté stačí jen vybrat umístění a název souboru, z/do kterého se má sudoku načíst/uložit. Při exportování se souboru automaticky přidá přípona .xml, aby byla zaručena přenositelnost mezi různými operačními systémy. Ukázkové xml soubory jsou přibaleny ve složce *„testovaci xml soubory“* na přiloženém CD.

3. Tvorba vlastního sudoku

Vlastní sudoku je možní vytvořit ručně (dvojklikem na příslušné políčko se umožní do něj vepsat číslo), ale doporučuje se vycházet z hotového řešení, které se dá vygenerovat přímo v aplikaci tlačítkem *„Vygeneruj řešení“*. Poté můžete libovolně odebírat číslice (kliknutím na zvolené políčko a stisknutím klávesy delete) a ideálně po každém odebrání zkontrolovat tlačítkem *„Zkontroluj sudoku“,* zda je vzniklé sudoku logicky řešitelné.

4. Použité metody a jejich obtížnost

V následující tabulce je uveden přehled metod použitých v generátoru (tzn. metody, které je nutné ovládat k řešení vygenerovaných sudoku).

Metoda	Obtížnost
Naked singles	lehká
Hidden singles	lehká
Pointing	střední
Claiming	střední
Naked subsets	těžká
X-Wings	těžká