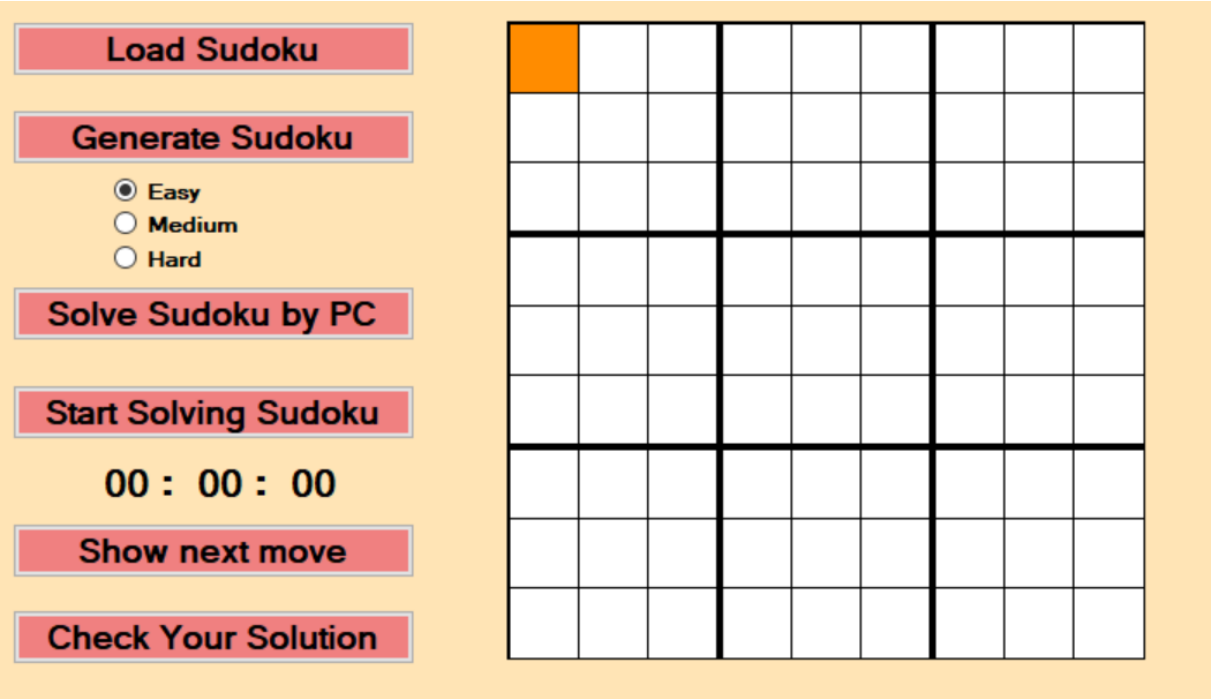
Sudoku aplikace – dokumentace

Úvod

Cílem bylo vytvořit aplikaci pro populární celosvětový hlavolam pocházející ze Země vycházejícího slunce zvaný **Sudoku**. Jedná se o jednoduchou logickou hru, kdy se do tabulky 9x9 snažíme umístit čísla od 1 do 9, tak aby v žádném sloupci, řádku a čtverci (3x3, kterých je na hrací ploše celkem 9) neležela dvě stejná čísla.

Program **SudokuApp** představuje lehce přívětivější grafické rozhraní pro uživatele s funkcemi jako nápověda dalšího tahu, vygenerování si Sudoku počítačem, vyřešení si Sudoku počítačem nebo nahrání si vlastní Sudoku.

Uživatelská příručka



Na obrázku je grafické uživatelské rozhraní. Rozhraní má následující **featury**:

- **Tlačítko "Load Sudoku"**
  - načte Sudoku z textového souboru (.txt) v následujícím formátu:
  - na každé řádce je právě 9 čísel vzájemně oddělené jednou mezerou
  - pokud chceme vyjádřit nevyplněné políčko, tak píšeme číslo 0
  - takto zapíšeme všechny řádky Sudoku na samostatný řádek

- př.

  0 0 0 0 2 0 0 8 9
  0 0 0 3 0 0 4 5 0
  4 0 0 8 0 6 2 3 0
  0 0 0 4 0 1 6 2 0
  8 2 9 0 6 0 7 1 4
  0 6 4 2 0 8 0 0 0
  0 7 2 9 0 5 0 0 1
  0 4 8 0 0 7 0 0 0
  9 1 0 0 4 0 0 0 0

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

- **Tlačítko “Generate Sudoku”**
  - vygeneruje Sudoku zadané obtížnosti
- **Tlačítko “Solve Sudoku by PC”**
  - Vyplní celou Sudoku
- **Tlačítko “Start Solving Sudoku”**
  - Odstartuje stopky pod tlačítkem – uživatel soutěží sám se sebou
- **Tlačítko “Show Next Move”**
  - Vyplní a na chvíli označí další možný krok při řešení
- **Tlačítko “Check Your Solution”**
  - Rozhodne, zdali je řešení luštitele správné
- **Hrací plocha**
  - Do ní může uživatel postupně vyplňovat a sledovat vývoj hry

Uživatel může tlačítka využívat dle libosti, ale upozorníme ho, že při načtení či vygenerování nové Sudoku se jeho progres ztratí a hrací plocha se **automaticky přepíše**.

Testovací data

Fungování programu - co se rozhraní týče - musí uživatel vyzkoušet sám, stejně tak generování Sudoku. Doporučuji zkusit vyluštit vygenerované Sudoku a přesvědčit se tak o jeho správném fungování. Co lze otestovat je fungování “řešičky” Sudoku. Připravil jsem do složky “testingSudoku” **testovací Sudoku**, které potřebují malý komentář, abychom se mohli přesvědčit o **korektních výsledcích**:

- “sudoku-lehke/stredni/tezke.txt” … obyčejné řešitelné Sudoku z knihy
- “sudoku-insane.txt” … sudoku, kdy lidský algoritmus selže
- “sudoku-neresitelne.txt” … neřešitelná Sudoku
- “sudoku-spatnyFormat.txt” … chybně formátovaná Sudoku

Popis programu

V této části se více z blízka podíváme na jednotilvé **části programu** a na reprezentaci dat v program, abychom mu mohli lépe porozumět či ho o něco doplnit. Celý zdrojový kód je hojně okomentován a měl by dobře dovysvětlil, co bude popsáno níže. Zdrojový kód obsahuje značné množství tříd – některé je vhodné zde zmínit, protože jsou stěžejní. Naopak některé jsou velmi doplňující a jejich smysl je očividný, proto jimi zde nebudu čtenáře zatěžovat.

Třídy a jejich popis:

1. **AppHandler**
   Řídí chod celé aplikace. Propojuje grafické rozhraní s vnitřní strukturou reprezentace Sudoku a její fungování. Cílem této třídy je inicializovat potřebné třídy, volat správné funkce s korektními parametry a dobře vyhodnocovat jejich výsledky.
2. **Cell**
   Reprezentuje vnitřní strukturu jedné buňky v Sudoku. Uchovává si číslo, možnosti (pole booleanu), počet možností a údaj, zdali je původně v zadání a nelze tedy změnit.
3. **Globals**
   Tato třída sdružuje některé statické proměnné, které se opakují v celém programu. Hlavní konstantou je *Size*, která říká kolik polí je na šířku, tedy u nás 9. Program je napsaný tak, aby např. při čísle 16 generoval Sudoku 16x16 s čísly od 1 do 16.
4. **GridHandler**
   Přímo ovládá hrací plochu a její zobrazení. Hrací plocha je reprezentovaná jako utilita DataGridView a *GridHandler* má příhodné funkce jako *PrintNumberInGrid* či *ClearGrid*.
5. **Sudoku**
   Sudoku si drží svoji vlastní plochu, tedy dvourozměrné pole buněk - *Cell[,] Grid*. Na její instanci můžeme volat funkce jako *Solve* a *TryHintNextStep*.
6. **SudokuParser**
   Parser pro načtení Sudoku z textového souboru se správně inicializovanými vlastnostmi jednotlivých buněk.
7. **SudokuGenerator**
   Stará se o celé vygenerování nové Sudoku
8. **SudokuSolver**
   Největší a nejpodstatnější třída, která se stará o celé řešení Sudoku stejně tak jako napovídání kroků. Vždy si při volání funkce drží private Sudoku sudoku a s ní v celém průběhu práce operuje. Má **3 hlavní funkce**:
   - void Solve(Cell[,] sudokuGrid)

    - Vyřeší celé Sudoku. Do *private sudoku* přiřadí *sudokuGrid* a díky referenci se při volání (z AppHandler) *sudoku.Solve()* celá *sudoku* přepíše. Už musí vědet, že je Sudoku řešitelná!
- bool TrySolve(Cell[,] sudokuGrid)
    - Zkusí vyřešit Sudoku a vrací bool, zdali je Sudoku vyřešitelná. Rozdíl se Solve je v návratové hodnotě a také v tom, že TrySolve si vytvoří pouze kopii *sudokuGrid* a uloží ji do *private sudoku*, a proto se Sudoku nepřepíše při volání TrySolve.
- bool TryHintNextStep(Cell[,] grid, out int num , out Position position)
    - Zkusí najít další krok, který vrátí v out parametrech, pokud to lze. AppHandler si out parametry již zpracuje sám.

Všechny tyto tři funkce využívají **dva hlavní algoritmy** na řešení:

- **SolveByBacktracking**
- **SolveLikeHuman**

Oba vrací bool, který značí, zdali je to pomocí těchto algoritmů vyřešitelné. O těchto algoritmech se podrobněji zmíním níže.

Algoritmus generování Sudoku

Je vyřešen triviálně (viz Diskuse). Backtrackingový algoritmus nám vyplní celou hrací plochu a postupně odstraňuji čísla z plochy. Po každém odstranění čísla z náhodné pozice na ploše musím zkontrolovat, že je Sudoku stále vyřešitelná a nedostaneme se do neřešitelné pozice. Složitost Sudoku se promítne čistě počtem odstraněných políček. Čím více políček odstraníme, tím je Sudoku těžší (viz Diskuse).

Algoritmus řešení Sudoku – Backtracking

Začínáme v levém horním rohu a postupně dosazujeme do každého volného políčka. Dosazujeme postupně čísla 1 až 9 v náhodném pořadí (normálně se dělá for cyklus od 1 do 9, random order je naimplementován z důvodu toho, že když se backtrackingem teď budeme snažit vyplnit naprosto prázdné Sudoku, tak dostaneme náhodně vypadající celou vyplněnou Sudoku, čehož využíváme v generování). Pro každé takové číslo se podívám do herní plochy, zdali neporušuje pravidla Sudoku, tedy jestli je jediné v řádku, sloupci a čtverci. Pokud ne, tak zkouším další. Pokud ano, tak ho tam doplním a rekurzivně se ptám, jestli toto číslo vedlo k řešení. Pokud ne, tak ho opět vymažu a zkouším číslo další.

Algoritmus řešení Sudoku – „Solve Like Human“

Tento algoritmus využívá metody, které používá i člověk, nespoléhá na náhodu. Nejrychlejší řešitelé Sudoku využívají toho, že ze začátku v každém nevyplněném políčku mohou býti čísla 1-9, ale pak se podíváme do řádku/sloupce/čtverce a můžeme vyškrtat všechna čísla, která na příslušné místo rozhodně doplnit nemůžeme. Takto postupujeme u

každého pole. Pokud v poli zbyde jen jedna možnost, pak můžeme toto číslo doplnit. Dalším stylem doplnění je, pokud je číslo X v daném sloupci/řádku/čtverci v možnostech jen v jednom políčku. Toto jsou dva hlavní způsoby (viz Diskuze). Tyto dvě vlastnosti opakujeme, dokud to lze, ideálně než není celá Sudoku vyřešena.

Diskuse

Na diskusi je toho opravdu hodně, proto věnuji každému aspektu jeden odstavec.

První část zamyšlení pojednává o **rozšiřitelnosti** programu. Co jsou dva hlavní okruhy, kde lze přidat je grafické uživatelské rozhraní a generátor Sudoku. Rozhraní je navrhnuté co nejjednodušeji, aby uživatel Sudoku opravdu viděl a s designem + rozšířením funkčnosti si lze vyhrát další stovky hodin. Rozhraní zároveň není dokonalé z hlediska nastavení pořadí tabulátorů a podobných detailů, které by ve finální distribuované aplikaci neměly chybět.

Generátor Sudoku je věc už algoritmicky zajímavější. Rozhodně neplatí, že čím více polí odstraníme, tím je Sudoku složitější. Záleží spíše na vazbách. Nebylo by složité navrhnout generátor, který by nějakým způsobem měřil za jak dlouho by Sudoku vyřešil počítač a zároveň zkoušel kolik člověk může vyplnit čísel na jeden pohled – takové číslo bychom chtěli minimalizovat. Nicméně generátor by se choval víceméně náhodně, ale údaje by nám řekly, jak je to tedy složité. Úlohou tedy je systematicky tvořit nějak těžké Sudoku, což je otázka právě dobře řešeného generátoru. Cílem tohoto programu bylo spíše vytvořit dobré prostředí pro vývoj takového generátoru. Zároveň změna velikosti Sudoku je možná, ale musí se upravit stránka vizuální – DataGridView by se musela přeformátovat, nicméně solver a generátor je řešen obecně.

Dlouhým pojednáním je **srovnání** backtrackingu s druhým algoritmem na řešení. Oba algoritmy mají svoje výhody a nevýhody, které rozeberu v následujících dvou odstavcích.

Backtracking je spolehlivý a vyřeší každou řešitelnou Sudoku. Nicméně je pomalý. Samozřejmě v C# nám to moc velkou roli nehraje (pohybujeme se kolem 15ms), i přes fakt, že je to zhruba 10x pomalejší než „lidský“ algoritmus. Například v Pascalu, by řešení Sudoku backtrackingem zabralo čas v řádech minut. Je dobré zmínit, že čistě z fungování algoritmu by mělo být vidět, že nejdelší čas zabere řešení u Sudoku s prázdným levým horním rohem, tedy že čísla jsou spíše ve spodní části Sudoku. Další výhodou je velmi krátká a přehledná implementace bez nutnosti zavádění speciálních tříd pro buňku – v Consoli nám stačí dvourozměrné pole čísel.

Naopak **„lidský“ algoritmus** je velmi rychlý. Avšak pochopitelně vyžaduje mnohem delší a komplikovanější implementaci. Co není na první pohled vidět je, že tento algoritmus se nám může zastavit i v případě, že má Sudoku jednoznačné řešení. Existují Sudoku, kde člověk nemá již co doplnit, aniž by si byl zcela jistý. Tedy v celém Sudoku není ani jedna jednoznačně doplnitelná situace a člověk se tedy musí rozhodnout pro jednu ze dvou možností a tím směrem pokračovat, dokud se nedostane ke sporu, nebo vyřešení. Takovéto Sudoku je řešitelné (tj. je jednoznačně doplnitelné), ale tento algoritmus by to nevyřešil. Při využití backtrackingu je jedno, kolik zbývá doplnitelných možností v políčku a pouze zkouší

náhodná čísla. Proto se z této situace dokáže bez problémů dostat. Naštěstí tyto Sudoku jsou velice ojedinělé a v běžných tiskovinách se s nimi nesetkáme (ani u těch s označením „velmi těžké“). K nalezení takto obtížných rébusů doporučuji internetový vyhledávač a využití tagů jako třeba „God level Sudoku“ či „Insane Sudoku“ a jim podobné.

V programu se primárně používá „lidský“ algoritmus a v případě, že si myslí, že se Sudoku nedá vyřešit, tak zavoláme backtracking, který nám jednoznačně určí, zdali lze, nebo nelze vyřešit. **Lepší algoritmus** jistě existuje. Jistým zlepšením by rozhodně bylo přidat více metod, kdy doplnit číslo z analýzy možností (viz tzv. X-Wing), ale je otázkou do jaké míry by to bylo urychlení či jestli bychom zajistili spolehlivé vyhodnocení řešitelnosti. Kolik takových speciálních pravidel bychom museli přidat? Zároveň nepochybuji, že existují i jiné více kombinatoricky založené přístupy, které maximalizují rychlost řešení. Samozřejmě řešit **časovou složitost** u 9x9 Sudoku je lehce zbytečné, ale u n x n Sudoku představuje značně obtížnou úlohu.