**TECHNICKÁ ČÁST**

POPIS ALGORITMU

**Generování Sudoku tabulky**

*Algoritmus pro řešení Sudoku*

K vyřešení Sudoku použijeme algoritmus prohledávání do hloubky (tedy DFS – algoritmus) a s tím související heuristiku.

Pro určování, zda je hodnota pro dané políčko vhodná (podle pravidel Sudoku), v konstantním čase si vytvoříme tabulku, kde pro každé políčko udržujeme seznam délky 10, kde budeme označovat, zda hodnotu indexu v seznamu lze do políčka korektně vložit. Seznamy postupně vyplníme hodnotami *True* (hodnotu indexu můžeme do políčka vložit) a *False* (hodnotu indexu již pro toto políčko nelze použít). Hodnota na indexu 0 bude libovolná, jelikož pracujeme s hodnotami 1 až 9.

Dále si vytvoříme tabulku všech políček, kde pro každé políčko označíme, zde je prázdné (*True/False*) a kolik existuje hodnot, které lze do políčka vložit. (Iniciace těchto tabulek je snadná, jelikož algoritmus začíná s prázdnou Sudoku tabulkou.) Tyto hodnoty využijeme při implementaci **heuristiky**, která spočívá v následujícím. Při vybírání prázdných políček si vybere vždy to, které má nejmenší počet možných hodnot, které do něj můžeme vložit.

Při řešení Sudoku si tedy vždy vybereme políčko s nejmenším počtem možných hodnot. Nyní postupně pro každou hodnotu 1-9 určíme, zda je pro toto políčko vhodná pomocí vytvořené tabulky. Pokud narazíme na vhodnou hodnotu, vložíme jí do prázdné políčka.

Pokud je po tomto kroku tabulka plně zaplněna (udržujeme si proměnnou s počtem volných políček), pak pokud jsme měli nalézt pouze jedno řešení, tak algoritmus skončí (jako funkce vrací hodnotu *True)*. Pokud jsme měli nalézt více řešení, tak zvýšíme hodnotu počítadla počtu nalezených řešení o jedna a porovnáme se zadaným počtem řešení. Pokud se hodnoty rovnají, algoritmus skončí (jako funkce vrací hodnotu *True)*, jinak pokračuje.

Jestliže tabulka není plně zaplněna, tak rekurzivně voláme funkci pro řešení Sudoku na tabulku s vyplněným políčkem. Upravíme také hodnoty v tabulce možných hodnot. Pokud volaná funkce vrátí hodnotu *True*, tak jsme již hotovy, tedy nalezli jsme řešení (jako funkce také vracíme hodnotu *True* pro vynoření z rekurze). Jinak upravované políčko necháme prázdné a upravíme tabulku možných hodnot. Upravíme i tabulku pro určování prázdných políček.

Pokud jsme nenašli zadaný počet řešení po vyzkoušení všech možností, vracíme hodnotu *False,* jelikož jsme již do tohoto políčka zkusili vložit veškeré vhodné hodnoty.

Pokud jsme našli daný počet řešení, pak je nyní tabulka jedním z nich. Pokud jsme nenašli daný počet řešení, tak je tabulka v původním stavu.

*Škála obtížnosti*

Škála obtížnosti je určena počtem políček, které řešitel musí vyplnit. Vychází z faktu, že čím více políček je vyplněno, tím snadněji se určí hodnoty nevyplněných políček. (Jelikož čím více políček je vyplněno, tím více hodnot pro jednotlivá prázdná políčka můžeme eliminovat.)

Uživatel zadá hodnotu 1 (nejjednodušší) až 5 (nejnáročnější), kde platí:

(1) V zadání bude 61-70 vyplněných polí

(2) V zadání bude 51-60 vyplněných polí

(3) V zadání bude 41-50 vyplněných polí

(4) V zadání bude 31-40 vyplněných polí

(5) V zadání bude 21-30 vyplněných polí

Zatím nejmenší nalezený počet prvků v zadání Sudoku tabulky tak, že má jednoznačné řešení, je 17. Tedy víme, že bude existovat i jednoznačné zadání s 21 a více zadanými hodnotami. Tudíž tuto škálu obtížnosti jistě můžeme zvolit.

Tato škála obtížnosti je stále jistě jen orientační, jelikož obtížnost výrazně závisí na rozmístění hodnot v tabulce. Pro určení obtížnosti celkově by bylo potřeba zpracovat algoritmus řešení Sudoku „lidským způsobem“ (nikoli „zkoušením všech možností“ jako algoritmus výše). To možné je, ale pro jednoduchost tento aspekt obtížnosti vynecháme.

*Algoritmus generování Sudoku tabulky*

Nejprve si vytvoříme prázdnou tabulku, tabulku s možnými hodnotami pro jednotlivá políčka a tabulku pro určování prázdných políček. Na tabulku nyní zavoláme funkci na řešení Sudoku, kde budeme požadovat nalezení jednoho řešení. Pro procházení hodnot 1-9 u každého prázdného políčka (v algoritmu řešení Sudoku) si vytvoříme seznam hodnot 1-9 v náhodném pořadí, abychom dosáhli co největší rozmanitosti zadání.

Toto řešení si uložíme, abychom ho měli k dispozici v další části programu.

Podle míry obtížnosti zvolené uživatelem určíme náhodně hodnotu v rozmezí daném škálou obtížnosti. Tato hodnota bude počet prázdných políček v zadání. Nyní si založíme počítadlo prázdných míst. Dokud nemá počítadlo danou hodnotu, tak náhodně volíme souřadnice jednoho políčka a určíme, zda je prázdné. Pokud je políčko prázdné, tak volíme znovu. Pokud je políčko neprázdné, tak jeho hodnotu vymažeme, upravíme tabulku možných hodnot a tabulku pro určení prázdných políček. Pro případ že toto políčko budeme chtít zpátky vyplnit, musíme do pomocných proměnných vložit všechny původní hodnoty měněných proměnných. Nyní zkusíme nalézt pomocí funkce na řešení Sudoku 2 různá řešení. Pokud taková najdeme, tak po odebrání této hodnoty již nemá tabulka jednoznačné řešení a hodnotu musíme vložit zpátky do tabulky (a tedy dále upravit pomocné tabulky).

Pokud nenajdeme 2 řešení, tak má upravená tabulka právě jedno řešení. (Víme, že určitě jedno řešení existuje, jelikož vycházíme z plně zaplněné tabulky.) Tedy tuto hodnotu opravdu můžeme odebrat. Zvýšíme počítadlo prázdných míst a s touto tabulkou dále pokračujeme.

Po konečně mnoha krocích dostaneme tabulku s daným počtem prázdných míst a právě jedním možným řešením.

Tento proces může být velice rychlý, ale také velice pomalý vzhledem k náhodnému generování souřadnic a hodnot v algoritmu. Z teorie pravděpodobnosti víme, že tento cyklus jistě po konečně mnoha krocích skončí, ale pro ošetření velmi časově náročných případů budeme počítat celkový počet zavolání funkce na řešení Sudoku a pokud přesáhne daný limit, tak se tabulka začne generovat od začátku.

**Algoritmus hry Sudoku (Pygame)**

Pro hru je potřeba vytvořit tabulku, umožnit hráči vpisovat do tabulky hodnoty 1-9 a zároveň měnit a mazat vepsané hodnoty. Poté je potřeba vyhodnotit správnost tohoto řešení.

Vytvoříme tedy tabulku a tlačítko na vyhodnocení tabulky. Do tabulky postupně vepíšeme hodnoty vygenerovaného zadání.

Uživatel do tabulky vloží hodnotu pomocí kliknutí levého tlačítka myši na políčko, kam chce hodnotu vložit. Tedy ošetříme případ, že uživatel klikne levým tlačítkem myši. Dála ověříme, že se kurzor nachází v oblasti tabulky.

Pokud se nachází v tabulce, ale na políčku odsazeném hodnotou ze zadání, tak hráči neumožníme hodnotu měnit.

Pokud se nachází na políčku, které nemá hodnotu ze zadání, potom ošetříme možnost, že uživatel stiskne tlačítko na klávesnici. Pokud se jedná o klávesu s číslem od 1 do 9, potom tuto hodnotu vložíme do tabulky (vložením bílého čtverce přes původní hodnotu v poli a vepsáním nové) a upravíme tak i tabulku se zadáním. (V tuto chvíli již musíme mít kopii původního zadání, abychom rozeznali, které hodnoty lze měnit.) Hodnotu klávesy určujeme díky klíčům v modulu pygame.

Jestliže uživatel stiskne tlačítko „backspace“, vložíme bílý čtverec přes původní hodnotu v tabulce a vymažeme hodnotu v tabulce, kam ukládáme hodnoty upravované uživatelem.

Když uživatel klikne levým tlačítkem myši na tlačítko pro určení správnosti řešení, tak pokud je tabulka plná (tedy při průchodu tabulkou jsme nenarazili na prázdné pole), tak ověříme, zda se rovná tabulka s řešením z algoritmu generování Sudoku tabulky a aktuální tabulka upravená uživatelem. Pokud se rovnají hodnoty na všech pozicích, potom uživateli gratulujeme ke zdárnému vyřešení Sudoku. Jestliže se hodnoty nerovnají, nabídneme možnost pokračovat, nebo Sudoku začít řešit od začátku.