**TECHNICKÁ ČÁST**

POPIS ALGORITMU

**Generování Sudoku tabulky**

*Algoritmus pro řešení Sudoku*

K vyřešení Sudoku použijeme algoritmus prohledávání stromu do hloubky (tedy DFS – algoritmus).

Pro určování v konstantním čase, zda je hodnota v určitém poli vhodná (podle pravidel Sudoku), si vytvoříme seznamy o velikosti 10 pro každý řádek, sloupec a čtverec (3×3 na hrací ploše Sudoku) v tabulce (9×9). Seznamy postupně vyplníme hodnotami *True* (v této oblasti se hodnota indexu ještě nevyskytuje) a *False* (v této oblasti se hodnota indexu již objevila, tedy už ji zde nemůžeme použít). Hodnota na indexu 0 bude libovolná.

Pro nalezení zatím nevyplněných polí v tabulce v konstantním čase využijeme zásobník, kam vložíme všechna volná políčka.

Nyní budeme postupně odebírat prázdná pole ze zásobníku. Po odebrání jednoho pole postupně pro každou hodnotu 1-9 určíme, zda je pro toto pole vhodná. To určíme pomocí seznamů hodnot řádku, sloupce a čtverce, kde se pole nachází. Pokud narazíme na vhodnou hodnotu, tak jí vložíme na prázdné pole.

Pokud je po tomto kroku tabulka plně zaplněna (zásobník je prázdný), 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 polem. Upravíme také hodnoty v seznamech hodnot daného řádku, sloupce a čtverce. 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 pole opět necháme prázdné a upravíme seznamy řádku, sloupce a čtverce. Následně přidáme toto pole do zásobníku prázdných polí.

Pokud jsme nenašli zadaný počet řešení po vyzkoušení všech možností, vracíme hodnotu *False,* jelikož jsme již do tohoto pole 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 polí, které řešitel musí vyplnit. Vychází z faktu, že čím více polí je vyplněno, tím snadněji se určí hodnoty nevyplněných polí. (Jelikož čím více polí je vyplněno, tím více hodnot pro jednotlivá prázdná pole 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 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 algoritmu výše). To možné je, ale pro jednoduchost tento aspekt obtížnosti vynecháme.

*Algoritmus generování Sudoku tabulky*

Nejprve si vytvoříme prázdnou tabulku, zásobník prázdných polí (nyní jsou zde obsažena všechna pole) a vhodné seznamy hodnot pro řádky, sloupce a čtverce. Zásobník prázdných polí budeme naplňovat v náhodném pořadí (v programu pomocí random.randint()), abychom vždy vygenerovali jiné zadání. Na tabulku nyní zavoláme funkci na řešení Sudoku, kde budeme požadovat nalezení jednoho řešení. Pro procházení hodnot 1-9 u každého prázdného pole (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 polí v zadání. Nyní si založíme počítadlo prázdných míst. Dokud nemá počítadlo danou hodnotu, tak náhodně volíme souřadnice jednoho pole a určíme, zda je prázdné. Pokud je pole prázdné, tak volíme znovu. Pokud je pole neprázdné, tak jeho hodnotu vymažeme, upravíme seznam řádku, sloupce a čtverce a vložíme toto pole do zásobníku prázdných polí. Pro případ že toto pole 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 pole (a tedy dále upravit seznamy a zásobník).

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 pole, 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 poli odsazeném hodnotou ze zadání, tak hráči neumožníme hodnotu měnit.

Pokud se nachází na poli, 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 na 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.