Střední průmyslová škola Ostrov, příspěvková organizace

ROČNÍKOVÁ PRÁCE

Sudoku

(desktopová aplikace)

Studijní obor	**Informační technologie**	
Třída	**I3**	Kubaf
Školní rok	**2015/2016**	**Jakub Jankovec**

Prohlášení autora

„Prohlašuji, že jsem tuto práci vypracoval samostatně a použil jsem literárních pramenů a informací, které uvádím v seznamu použité literatury a zdrojů informací.“

V Ostrově dne …… …….

Podpis autora

Střední průmyslová škola Ostrov, p. o.

Klínovecká 11197, Ostrov 363 01, IČ: 70845425

ROČNÍKOVÁ PRÁCE

Obor: Informační technologie

Třída: I3

Jméno: Jakub Jankovec

Název práce: Sudoku (Desktopová aplikace)

Zadání:

Cílem je vytvořit program, který vygeneruje náhodný Sudoku zadání, včetně výběru obtížnosti (lehká, střední, těžká). Vygenerované sudoku si uživatel může následně zahrát. Program bude ověřovat správnost hráčem vloženého čísla do pole a při chybném vložení ukáže, kde jsou shody s jinými čísly. Pokud uživatel bude chtít, program vypíše možná čísla, které lze na místo dosadit.
Program bude odladěný, bez chyb, napsaný v jazyce C#. Práce bude kromě programu obsahovat analytickou a programátorskou dokumentaci a uživatelský manuál.

Zadavatel: Mgr. Kateřina Fexová, Mgr. Zuzana Štočková, Ing. Roman Stark

Závazné termíny pro zpracování:

Zadání práce	31. 3. 2016
Odevzdání kompletní práce	31. 5. 2016
Obhajoba	13. 6. 2016

Pokyny:

- Práce bude v rozsahu hlavní části (jádro vlastní práce) 10 – 15 stran.
- Práce bude mít všechny požadované části a budou dodrženy normy pro úpravu písemností a citací (viz soubor *Normalizovaná úprava písemností*).
- Práce bude obsahovat obrázky nebo tabulky, příp. grafy, přílohu i vysvětlení principu/fungování programu.
- V práci možná bude použito doslovné i parafrázované citace.
- Bude odevzdána v jednom výtisku (kroužková vazba) a v elektronické podobě (na přiloženém CD) ke stanovenému datu.
- Součástí práce bude prezentace v PowerPointu.

…….. ……..

autor práce zadavatel práce

Anotace

Moje práce je zaměřená na vygenerování náhodného Sudoku zadání, které lze následně bez problému hrát na PC. Pro generování jsou na výběr obtížnosti „lehká“, „střední“ a „těžká“. Existuje ještě jedna obtížnost, která se nazývá „libovolná“. V libovolné obtížnosti může uživatel určit kolik políček v Sudoku bude smazáno po vygenerovaní.V dokumentaci je aplikace popsána jak z uživatelský části, tak i z programové části, ve který je popsán algoritmus generování.

Anotation

My work is focused on generating of random Sudoku solution, which can user play then on PC with no problem. For generating, there is for choice difficulty „easy“, „medium“ and „hard“. There is still another one difficulty, which called „custom“. In custom difficulty can user define how much squares in Sudoku will be cleared after generating. In documentation is application described both from user part and program part, where is described algorithm of generating.

Obsah

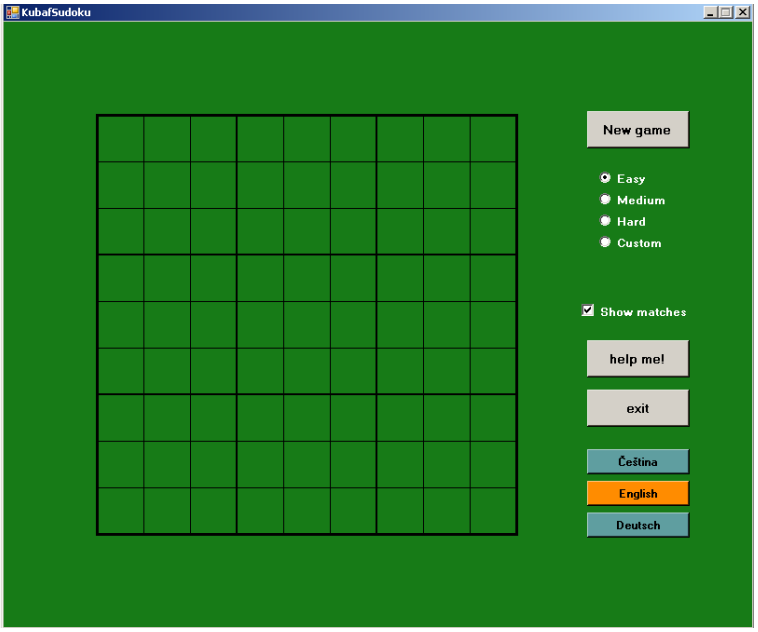
1 Úvod

Tento program Vám vygeneruje náhodný Sudoku zadání, které si můžete zahrát. Na výběr jsou různý obtížnosti od lehké, přes střední až po těžkou. Dále je tu na výběr libovolná „Custom“ obtížnost, ve který si sami vyberete kolik políček s čísly chcete v zadání vymazat. Nově přibyl herní mód „LevelMode“, ve kterým začínáte v Sudoku s 1 vynechaným políčkem a při dokončení postupujete dál do další úrovně, s více vymazanými čísly. Postup v tomto herním režimu si můžete kdykoliv uložit či načíst pomocí tlačítek „Save“ a „Load“. Když už si s řešením Sudoku nevíte rady, v programu je tlačítko „Poraď mi“, které Vám poradí jaké číslo na současné pozici můžete dosadit. Aplikaci jsem naprogramoval v jazyku C# za pomocí vývojářského programu Visual Studio ve Windows Form Application .NET 4.0, takže minimální požadavkem na systém je OS Windows XP.

2 Rozbor aplikace z uživatelského pohledu

2.1 Spuštění programu

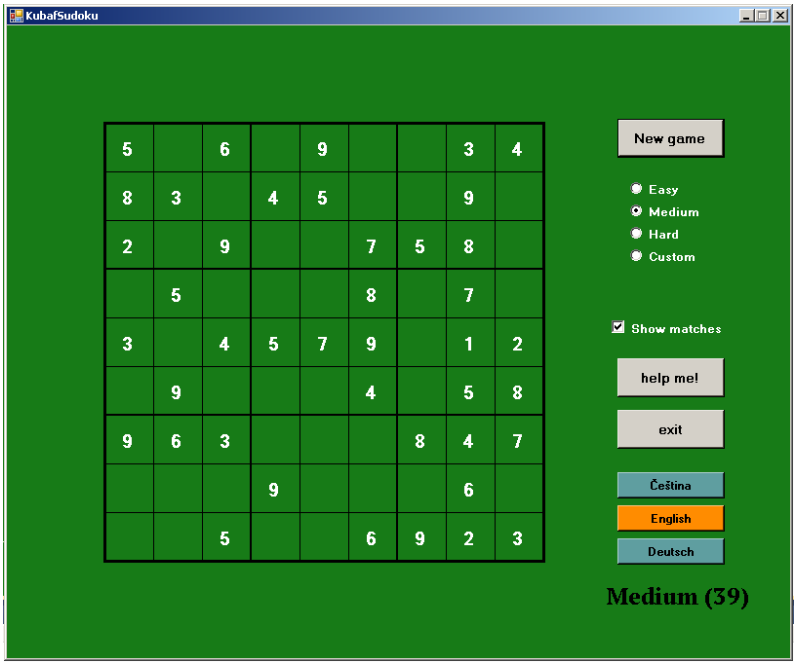
Program spustíte pomocí jednoduchého binárního (.exe) souboru „Sudoku.exe“, který zabírá cca 23 Kib. V úvodu se otevře okénko s již připravenou hrací deskou, ale ještě nevygenerovaným sudoku. Jako původní jazyk je nastavená angličtina. Program beží na rozhrání Windows Form Application .NET 4.0, takže je spustitelný i na Windows XP.



Obrázek 1 - Spuštění aplikace

2.2 Nová hra

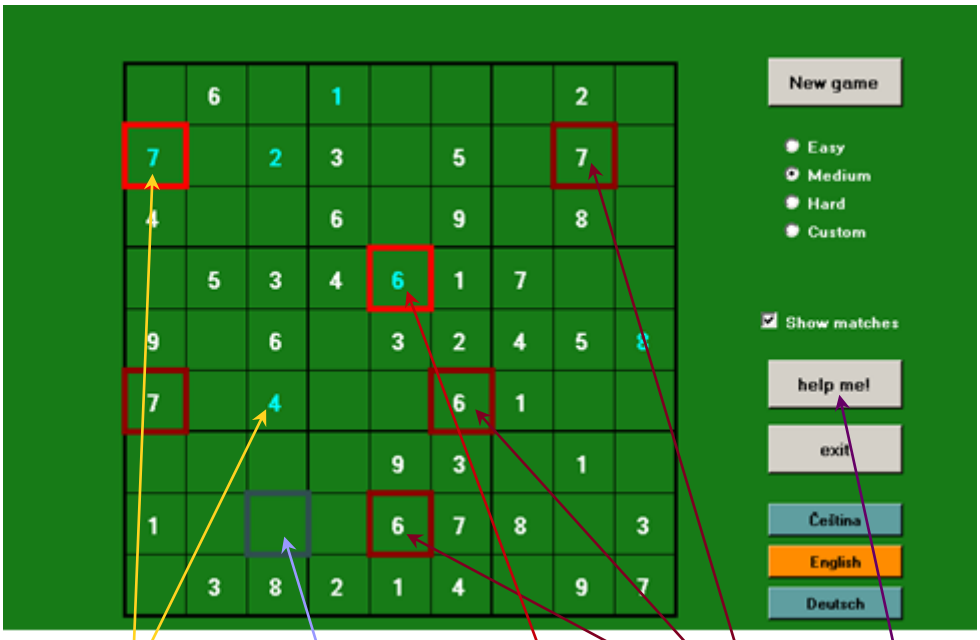
Program se skládá z herního pole a ovládacího panelu s tlačítky, který je po pravé straně herního pole.Vygenerování Sudoku zadání se jednoduše spustí tlačítkem „New game“, které se nachází v úplně horní části ovládacího panelu.



Obrázek 2 - Nová hra: vygenerované Sudoku zadání

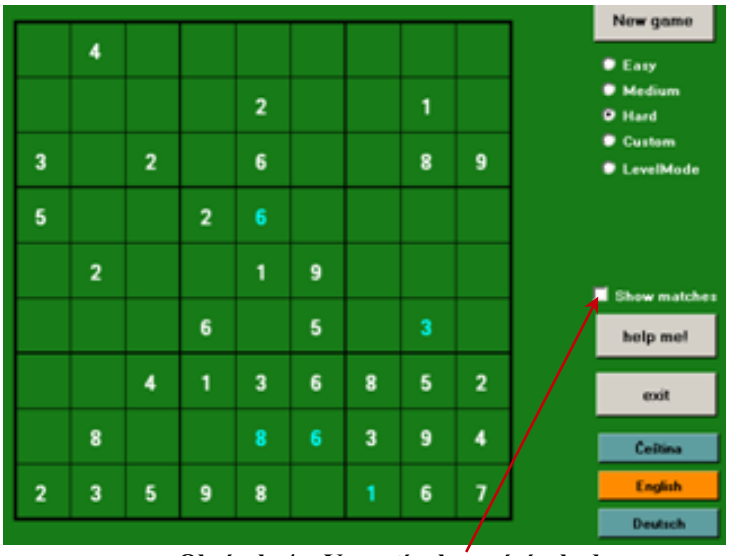
2.3 Základní ovládání

Políčko si jednoduše vyberete myší, potom se můžete mezi políčky pohybovat šipkami. Generovaná čísla mají bílou barvu a jsou dána na pevno, tudíž je nemůžete přepisovat. Číslice můžete vkládat pouze do prázdných políček nebo tam kde jste je již sami vložili, za pomocí numerické klávesnice (NumPad) nebo prostřednictvím tlačítek v horní řadě klávesnice, nad písmeny. Tlačítkem „0“ nebo pravým tlačítkem myši vymažete svoji číslici z políčka úplně. Uživatelem vložená čísla mají barvu do modra („aqua“). Při vložení číslice, která nesedí do podmínek Sudoku, tzn. že je někde v řadě, sloupci či čtverci shoda, program ukáže kde ty shody jsou čtvercovým označením.



Obrázek 3 – Základní ovládání a ukazování shod

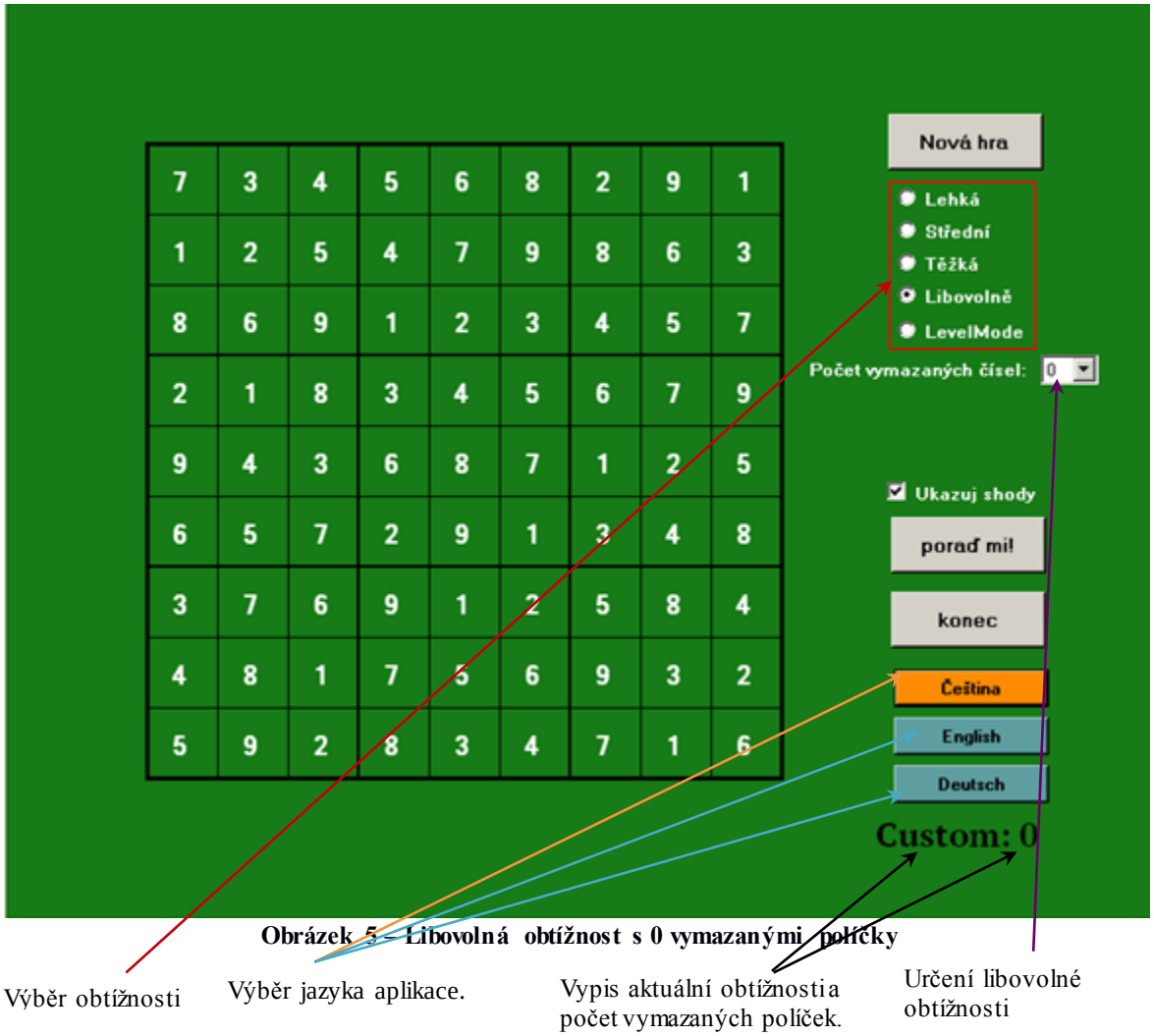
Jestli se Vám zdá, že Sudoku s označováním shod budete mít příliš snadný, v ovládacím panelu je tuto funkci možný vypnout odškrtnutím čtverečku „Show matches“. Když už si nevíte rady, jakou číslici do prázdného políčka vložit, můžete využít tlačítka „help me!“



Obrázek 4 – Vypnutí ukazování shod

2.4 Výběr obtížnosti a základní ovládací prvky

Obtížnosti fungují tak, že se z celého zadání náhodně odebere vždy určitý počet čísel v políčkách. Lehká obtížnost má rozmezí 23 - 32 vynechaných čísel, střední 35 - 44, těžká 48 - 58 a libovolná 0 - 81. Libovolná obtížnost záleží pouze na volbě uživatele.

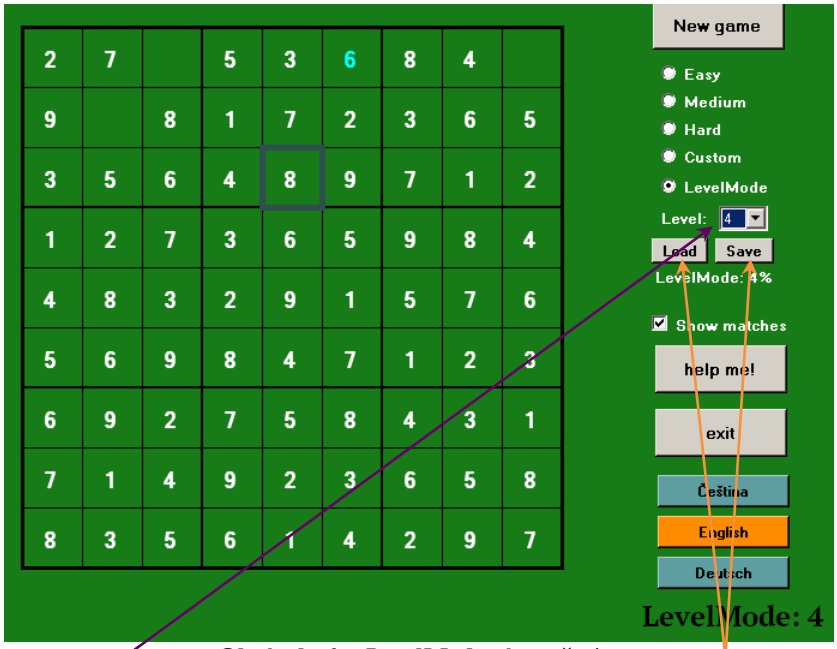


Obrázek 5 – Libovolná obtížnost s 0 vymazanými políčky

Nově přibyla obtížnost „LevelMode“, kterou popíšu v následující kapitole. Obtížnost si vyberete zvolením příslušné možnosti pod tlačítkem „Nová hra“. Mezi základní ovládací prvky hry též patří přepínání jazyku aplikace. Na výběr máte český, anglický a německý jazyk, aktuálně zvolený jazyk září oranžovou barvou. Pod panely jazyků si můžete přečíst jakou hrajete obtížnost a kolik je prázdných políček v zadání.

2.5 LevelMode

Levelmode jsem nazval nový herní mód, který mě napadlo vytvořit. Začínáte na úrovni 1, která obsahuje Sudoku s 1 vymazaným herní políčkem. S každým dokončeným levelem se dostáváte dál do další úrovně, s jedním vynechaným políčkem navíc. Poslední úroveň je 81, která má vynecháno 9x9 polí. Tímto stylem by se dalo říct „Zahrajte si celé Sudoku od začátku do konce“. Postup v tomto herním režimu si můžete kdykoliv uložit či načíst.

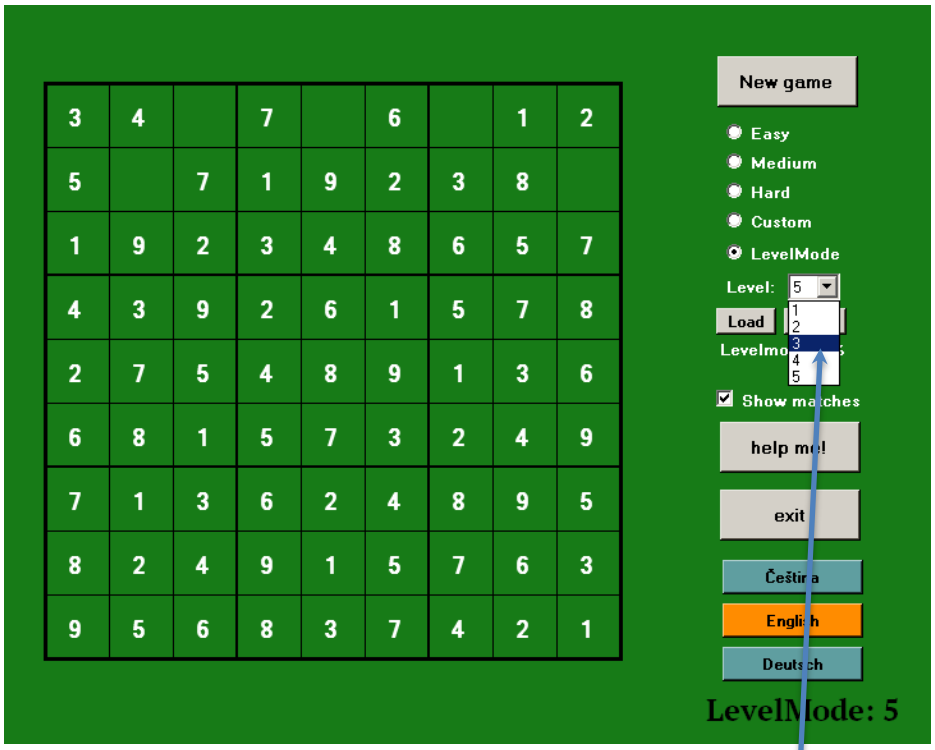


Aktuální úroveň

Obrázek 6 - LevelMode úroveň 4

Tlačítka pro načtení a uložení

Jakoukoliv úroveň, přes kterou jste již dostali si kdykoliv můžete znovu zahrát, ale abyste se dostali do nové úrovně, musíte dohrát level, který se nachází před ní.

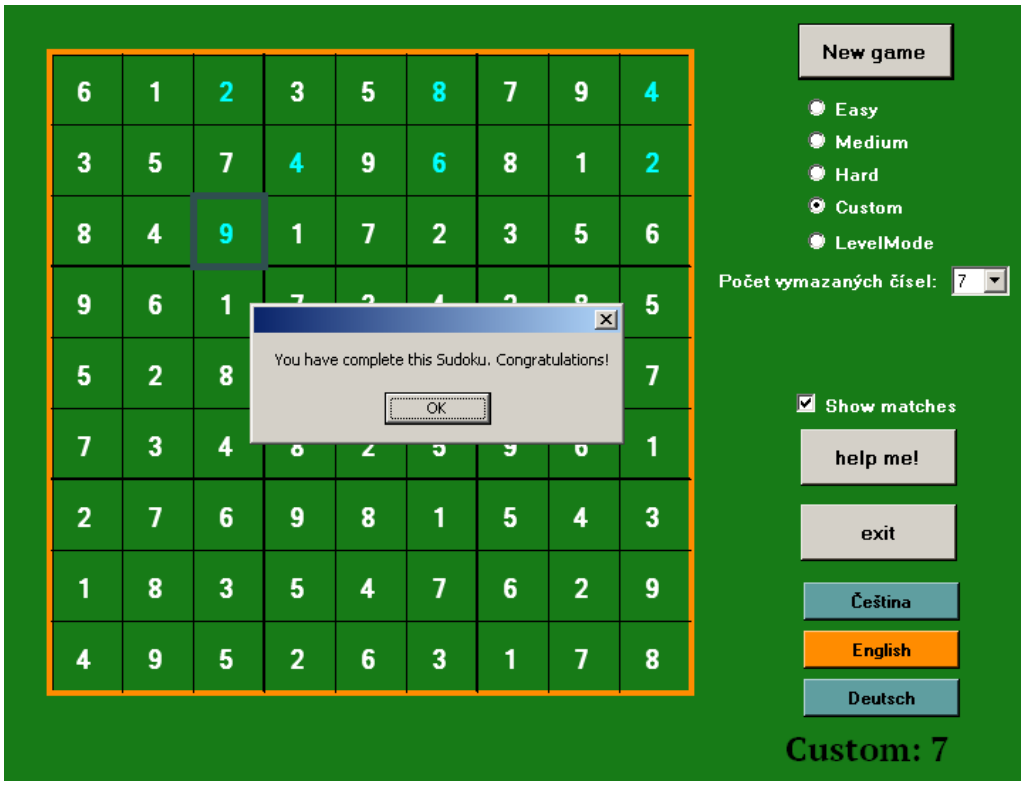


Obrázek 7 – LevelMode úroveň 5 a výběr úrovně

Výběr úrovně

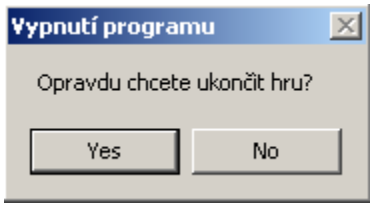
2.6 Konec hry

Jakmile dořešíte celé Sudoku zadání, program to detekuje, zahlásí výhru a poblahopřeje Vám. Obrys zadání se změní na vítěznou zlato-oranžovou barvu.



Obrázek 8 – Dokončení Sudoku zadání

Vypnout hru můžete buď klasickým Windows křížkem nebo alternativním tlačítkem „exit“ (či „konec“ v češtině). Když chcete ukončit hru, program se Vás pro jistotu zeptá, zdali doopravdy chcete vypnout program, pro případ, že byste se překlikli.

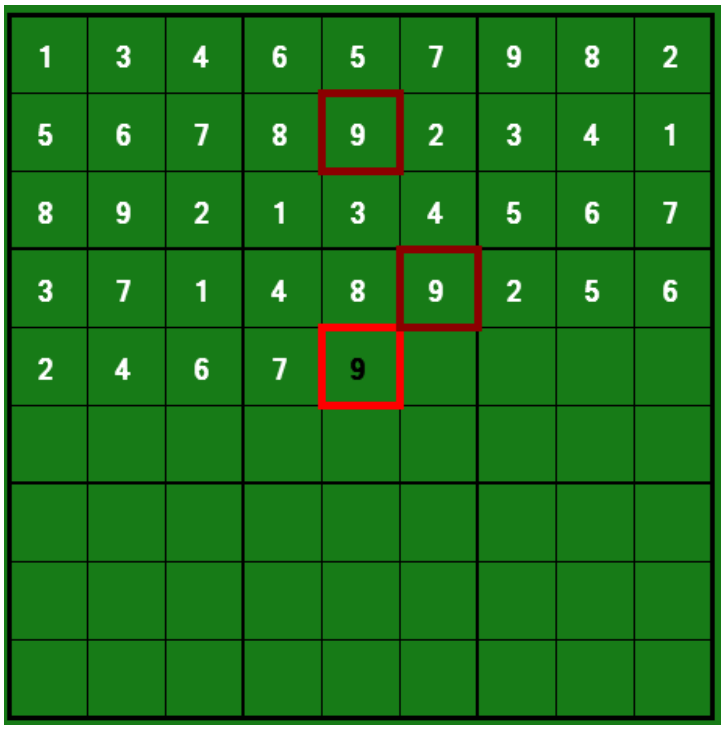


Obrázek 9 – Ukončení hry

3 Analytická dokumentace

3.1 Stručný popis generování Sudoku

Program začíná na souřadnicích x = 1 a y = 1, to znamená na prvním místě v řadě v prvním sloupečku. Při prázdném políčku nejdříve vyzkouší zda lze nějaké náhodné číslo od 1 do 9 dosadit na tuhle pozici podle pravidel Sudoku. Pokud číslo nejde dosadit, algoritmus začne automaticky vyhledávat další čísla v pořadí, zdali jdou dosadit na konkrétní pozici.

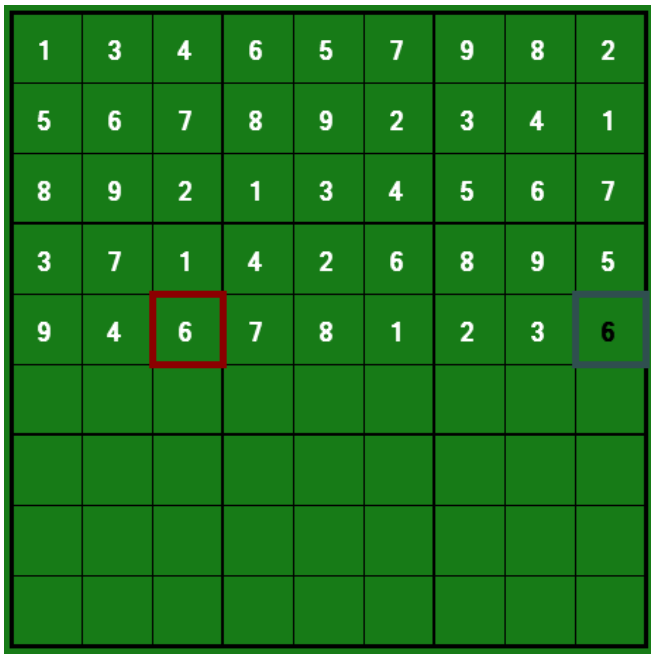


Obrázek 10 – Shoda s generovaným číslem

Jakmile najde číslo, které lze na tuto pozici dosadit, posune se řadová souřadnice x o 1 (dále pouze „x“) dopředu a když algoritmus dojde na konec řady, připočte se sloupcová souřadnice y (dále jen „y“) o 1 a x se obnoví na 1, to znamená na začátek řady .

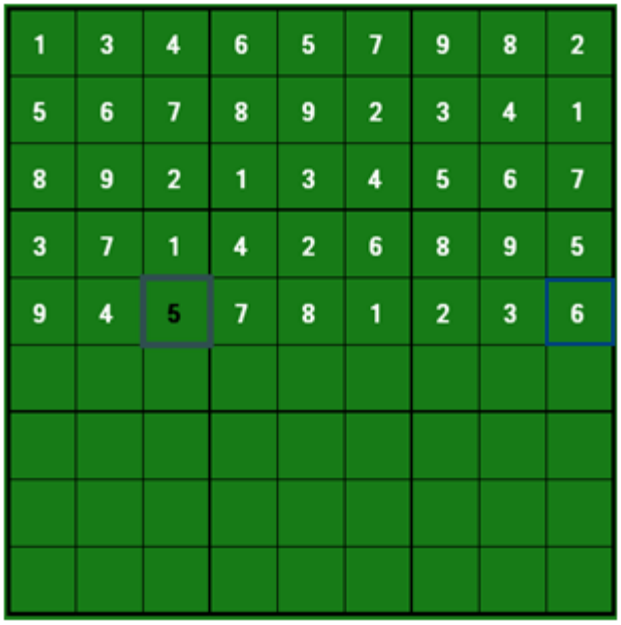
3.1.1 Horizontální oprava

Když na prázdné pole už nic podle pravidel Sudoku nelze dosadit, algoritmus začne hledat jiné číslo z řady, které je možný dosadit na momentální pozici (x, y). Jakmile najde číslo z řady, které tam dát jde, ihned začne hledat nové číslo, které půjde dosadit na původní pozici čísla z řady.



Obrázek 11 – Vyhledání čísla z řady při horizontální opravě

Hned jak to číslo najde, automaticky ho tam dosadí a na současnou pozici x a y vloží tohle číslo z řady, pak pokračuje dál v dalším zkoušení nových náhodných čísel na další souřadnici x a y.

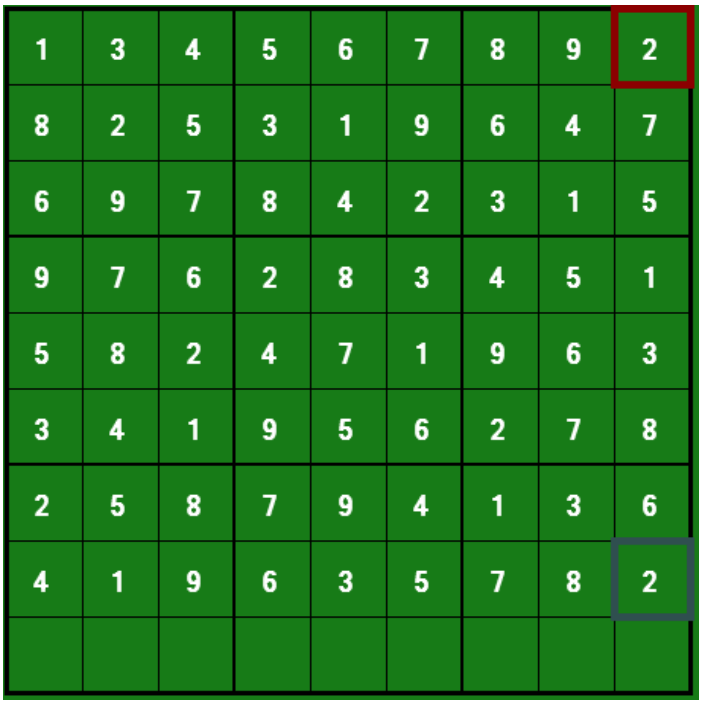


Obrázek 12 – Nalezení nového čísla na původní pozici z řady při horizontální opravě

Pomocí této metody se algoritmu ve většině případů podaří vygenerovat Sudoku až do konce. Jakmile už nelze použít žádné číslo z řady, což znamená, že algoritmus došel se souřadnicí „Px“ na začátek řady. Vyzkouší se jiný styl prohazování čísel.

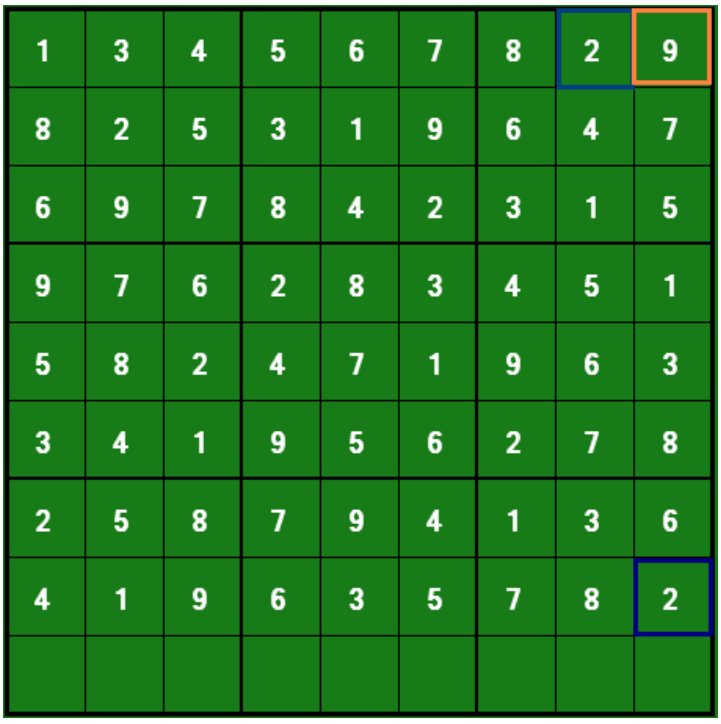
3.1.2 Vertikální oprava

Pro tuto metodu algoritmus vyhledá číslo, které má na současném prázdném políčku shodu pouze ve sloupečku (y).



Obrázek 13 – Nalezní čísla shodného pouze v Y při vertikální opravě

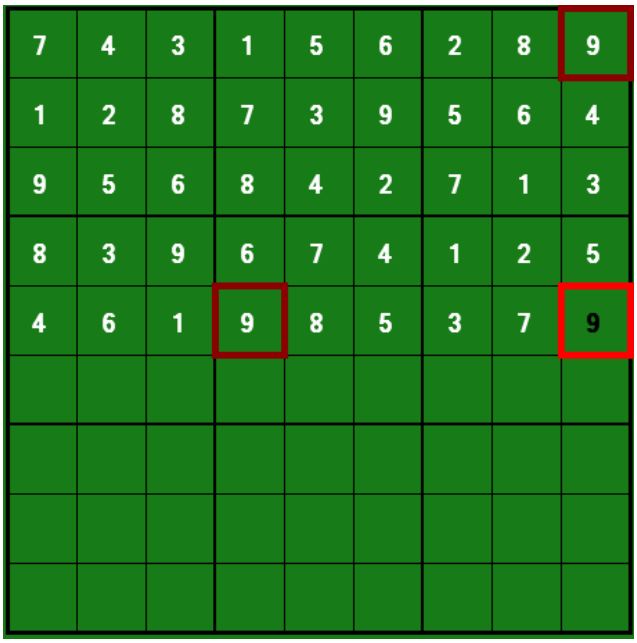
V tomto sloupci zkusí shodné číslo prohodit s jiným číslem v řadě a čtverci, ve kterém se nachází. Díky tomu, že se tímto eliminuje ověřování shody s čísly v řadě a ve čtverci, musí se ověřovat pouze shoda ve sloupci (y). Jakmile ani jedno z čísel nemá shodu v sloupečku, čísla se mohou prohodit.



Obrázek 14 – Prohození čísla ve stejný řadě a čtverci

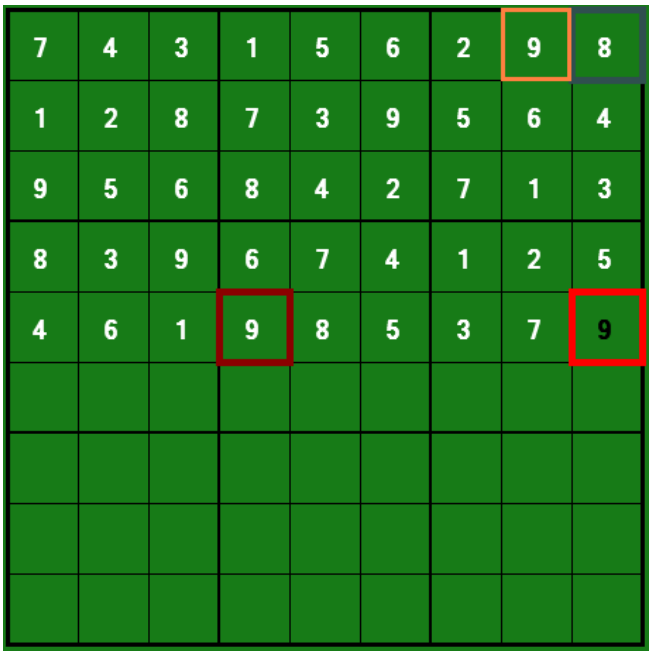
Takto se zkouší další čísla, která mají na momentální pozici x a y shodu pouze ve sloupci, dokud se nevyhledá číslo, které lze nad tímto číslemt. Jakmile se čísla prohodí, na prázdné políčko se dosadí nové číslo, které mělo shodu pouze ve sloupci a generování může pokračovat dál.

3.1.3 Horizontálně-vertikální oprava



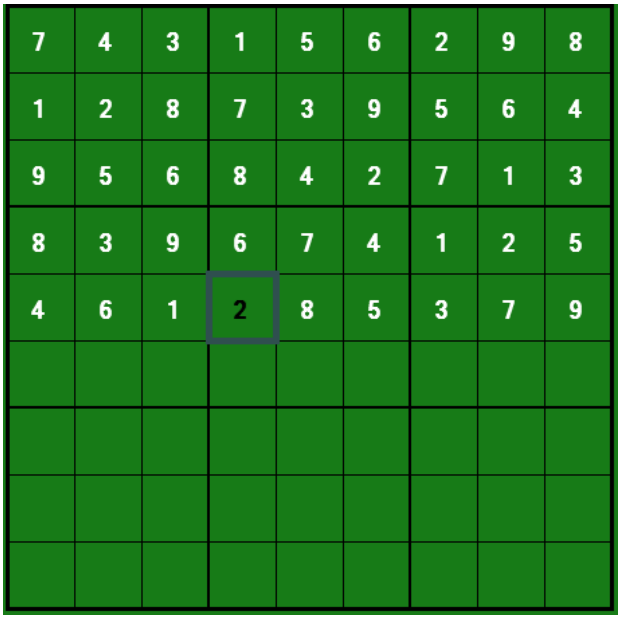
Obrázek 15 – Nalezení čísla z řady, které má shodu pouze v sloupci (y)

Pokud už nejde aplikovat ani tento způsob prohazování na žádné z čísel od 1 do 9, program zkusí vyhledat číslo z řady, které na současné pozici má shodu pouze ve sloupci (y). Tohle shodné číslo ve sloupci se opět zkusí prohodit s jiným číslem ve své řadě a čtverci.



Obrázek 16 - Prohození čísel ve stejný řadě a čtverci v rámci horiontálně-vertikální opravy

Když jde prohodit, postup je stejný, ale ještě se musí jako při horizontální opravě najít nové číslo, které jde použít na původní pozici dosazovaného čísla z řady.

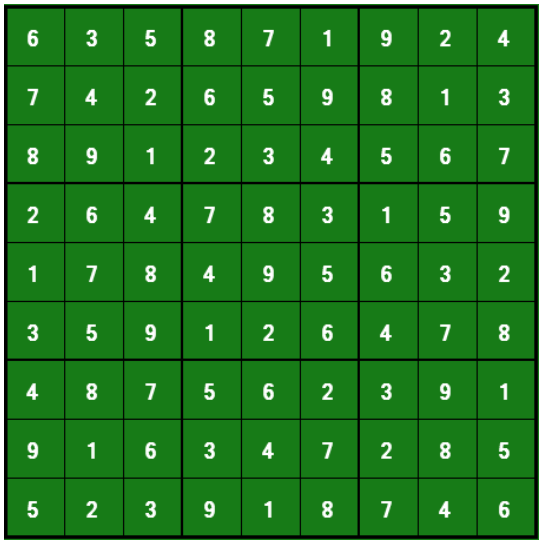


Obrázek 17 – Nalezení nového čísla na původní souřadnici z řady

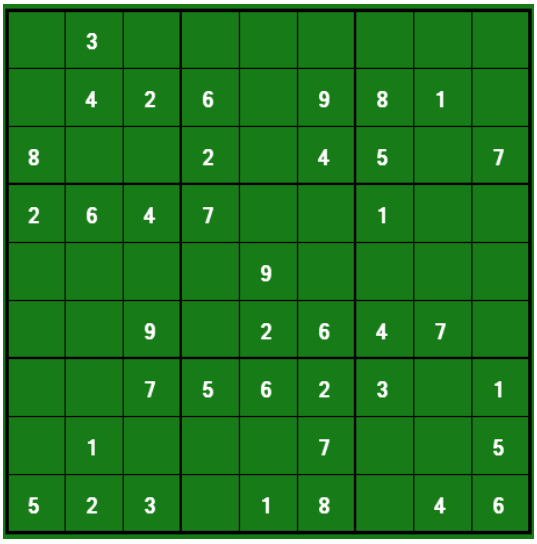
Pokud ani takto už nic prohodit nejde, program smaže celé Sudoku zadání a začne generovat odznova. Tímto způsobem se dál zkouší generovat Sudoku, dokud algoritmus nevygeneruje celé Sudoku zadání. Obvykle se to podaří do 1 až 3 pokusů, výjimečně jsem zažil případy, kdy se to podařilo vygenerovat až na 11. pokus. Vymyslel jsem ještě další jednoduché styly prohazování čísel, které by ušetřily spoustu dalších generování celého Sudoku od znova, ale na naprogramování mi už nezbyl čas.

3.1.4 Sudoku vygenerováno – vytvoření zadání pro hru

Jakmile je celé Sudoku vygenerovaný, což znamená, že program dosadil číslo i na poslední místo v řadě posledního sloupce (x = 9; y = 9), zbývá už jen náhodně promazat některá políčka v herním poli a tím z vygenerovaného Sudoku vytvořit zadání pro hru. Počet náhodně vymazaných čísel je určen obtížností či uživatelem.



Obrázek 18 – Kompletně vygenerované Sudoku



Obrázek 19 – Vytvoření Sudoku zadání

3.2 Rozbor kódu generování Sudoku

Při zmáčknutí tlačítka „New Game“ se zavolá funkce NewGame().


```
private void btnGen_Click(object sender, EventArgs e)
{
    if (pocetTahu < 5)
    { NewGame(); }
```

Obrázky 20 a 21 – Tlačítko New game

Tato funkce obnoví všechny souřadnicové proměnný na původní hodnoty, obnoví všechny hodnoty v polích na 0 a zavolá funkci generator();.

```
private void NewGame()
{
    x = 1; y = 1;
    z = 1; canna = 1;
    Px = x; Py = y;
    pocetTahu = 0;
    napovedy = 0;
    lbNapoveda.Visible = false;
    cleared = 0;

    for (var a = 1; a <= 9; a++)
    {
        for (var b = 1; b <= 9; b++)
        {
            areaXY[a, b] = 0;
            PotenXY[a, b] = 0;
            areaZ[a, b] = 0;
            vstpXY[a, b] = 0;
            ShodyXY[a, b] = 0;
        }
    }

    Win = false;
    first = true;
    pripocet = false;
    generator();
}
```

Obrázek 22 - Metoda NewGame()

3.2.1 Základní logika generování

Tato metoda nejdříve vyzkouší jestli souřadnice **x** není 9 a současně **y** souřadnice není 9, což by znamenalo, že algoritmus už došel na konec pole a Sudoku je vygenerované. Pokud nedošel, pak přiřadí proměnný generat náhodné číslo od 1 do 9 a nalezne souřadnice pro velké čtverce **z** a malé čtverečky **canna** vycházející ze souřadnic **x** a **y**.

```
if (x != 9 || y != 9)
{
    generat = random.Next(1, 9);
    hledani = true;

    z = ((x - 1) / 3) + 1 + 3 * ((y - 1) / 3);
    canna = ((x - 1) % 3) + 1 + ((y - 1) % 3) * 3;
```

Obrázek 23 – Přiřazení základních proměnných

Následně se vyzkouší vygenerované číslo **generat**, jestli má shodu v X, Y či Z souřadnici.

```
while (hledani)
{
    for (int i = 1; i <= 9; i++) //Hledání shod s generatem
    {
        if (areaXY[i, y] == generat || areaXY[x, i] == generat || areaZ[z, i] == generat) //Shoda je na X či Y či Z.
        {
            if (firstChange)
            { g = generat; firstChange = false; }
            if (generat < 9)
            { generat++; }
            else { generat = 1; }

            if (generat == g) //vrátil se na původní pozici -> Už nic nejde... repairX();
            { Px = x; hledani = false; a = 1; repairX(); break; }
            break;
        }
```

Obrázek 24 – Hledání nového čísla na souřadnice X a Y

Pokud shodu na X, Y, či Canna souřadnici má, zastaví cyklus pro vyhledávání shod (for i) a začne hledat jiné číslo místo **generatu** přípočtem. Každým přípočtem cyklus **while (hledani)** otestuje znovu nové číslo generat jestli nemá shodu podle podmínek Sudoku. Jestliže generat má po dalších přípočtech hodnotu 9, obnoví se jeho hodnota na 1 a dál se zkouší nové čísla, dokud číslo generat nedojde na jeho původní náhodnou hodnotu.

```
else if (i == 9) //Když není shoda.. (i došlo na 9)
{
    areaXY[x, y] = generat;
    areaZ[z, canna] = generat;
    PotenXY[x, y] = generat;
    vstpXY[x, y] = 1;

    hledani = false;
    nextGen();
    return;
}
```

Obrázek 25 – Číslo lze vložit

Když číslo nikde v shodu nemá, což znamená, že hodnota **i** z cyklu for došla na 9, přiřadí se vygenerované číslo na pozice herních polí areaXY[x, y], areaZ[z, canna], do pomocného pole PotenXY[x, y] a zapíše hodnotu 1 do vstupního pole vstpXY[x, y], aby bylo jasný, že číslo na tuto pozici přiřadil počítač. Zastaví hledání nového čísla a zavolá klasickou metodu **nextGen()**, která nastaví přípočet x, obnoví některé důležité spínače a zavolá **generator();**, čímž testuje nový náhodný číslo na nové pozici x a vše pokračuje dál.

```
//Poslaní dalšího generatoru();
private void nextGen()
{
    if (y < 9 || x < 9)
    {
        g = 0; //reset uložení generátu
        firstChange = true;
        pripocet = true;

        first = true;
        for (int i = 1; i <= 10; i++)
        { SaveX[i] = 0; }

        generator();
    }
```

Obrázek 26 – metoda nextGen()

Při každém přípočtu x metoda generator() ověří, zda x už není větší jak 9.

```
if (pripocet)
{
    x++; Px = x;
    Py = y;
    pripocet = false;
}
if (x > 9)
{
    x = 1; Px = x;
    y++; Py = y;

    if (y > 9)
    { return; }
}
```

Obrázek 27 – Posun na další souřadnice

Jestliže je, pak se nastaví hodnota x na 1 a sloupec (y) se přičte a všechno pokračuje dál.

3.2.2 OpravaX() - Když už žádné číslo od 1 do 9 nejde použít na novou pozici

```
if (areaXY[i, y] == generat || areaXY[x, i] == generat || areaZ[z, i] == generat) //Shoda je na X či Y či Z.
{
    if (firstChange)
    { g = generat; firstChange = false; }
    if (generat < 9)
    { generat++; }
    else { generat = 1; }

    if (generat == g) //vrátil se na původní pozici -> Už nic nejde... repairX();
    { Px = x; hledani = false; a = 1; repairX(); break; }
    break;
}
else if (i == 9) //Když není shoda.. (i došlo na 9)
{
    areaXY[x, y] = generat;
    areaZ[z, canna] = generat;
    PotenXY[x, y] = generat;
    vstpXY[x, y] = 1;

    hledani = false;
    nextGen();
    return;
}
```

Obrázek 28 – zavolání metody oprav „repairX()“

Jakmile se číslo generát dostane na jeho původní hodnotu vypne se hledání nového čísla a zavolá se metoda repairX() se základní hodnotou pro switch (a) = 1. Tato se skládá ze metoda 6 case (beden), každý představuje konkrétní postup při opravě dříve zadaného čísla, např. prohozením.

```
//oprava:
private void repairX()
{
    switch (a)
    {
        case 1: //vyhledá číslo z areaXY[Px, y], které jde dosadit na areaXY[x, y].
            {
                newNum = 0;
                if (Px > 1)
                {
                    Px--; Py = y;
                    Pz = ((Px - 1) / 3) + 1 + 3 * ((y - 1) / 3);
                    Pcanna = ((Px - 1) % 3) + 1 + ((y - 1) % 3) * 3;

                    for (int k = 1; k <= 9; k++) //vyhledá číslo z Px, y, které jde na x, y.
                    {
                        if (areaXY[x, k] == areaXY[Px, y] || areaZ[z, k] == areaXY[Px, y]) //číslo má shodu na Y či Z.
                        { repairX(); break; }
                        else if (k == 9) //číslo z řady jde dosadit na místo x, y -> posun.
                        {
                            saveNum = areaXY[Px, y];

                            a = 2;
                            repairX();
                            break;
                        }
                    }
                }
```

Obrázek 29 - metoda oprav „repairX()“, case 1: číslo z řady, které lze použít na současné souřadnice

V prvním case nalezne číslo z řady, které jde použít z hlediska sloupce a čtverce na současnou pozici. Pokud některé jde, tak si toto číslo uloží do proměnný saveNum, přepne switch (a) na 2 a znovu zavolá metodu repairX().

```
        case 2: //vyhledá nové číslo, které lze dosadit na areaXY[Px, y].
            {
                if (newNum < 9)
                {
                    newNum++;
                    for (int k = 1; k <= 9; k++)
                    {
                        if (areaXY[k, y] == newNum || areaXY[Px, k] == newNum || areaZ[Pz, k] == newNum) //číslo se shoduje
                        { repairX(); break; }
                        else if (k == 9) //vše v pořádku -> číslo tam lze dosadit.
                        {
                            areaXY[x, y] = saveNum;
                            areaZ[z, canna] = saveNum;
                            areaXY[Px, Py] = newNum;
                            areaZ[Pz, Pcanna] = newNum;

                            a = 1;
                            PotenXY[x, y] = saveNum;
                            PotenXY[Px, Py] = newNum;
                            vstpXY[x, y] = 1;
                            vstpXY[Px, Py] = 1;

                            nextGen();
                            break;
                        }
                    }
                }
                else
                { a = 1; newNum = 0; repairX(); break; } //Nic nejde dosadit.
```

Obrázek 30 – metoda oprav „repairX()“, case 2: nové číslo na pozici vyjmutého čísla z řady

Ve druhým case vyhledá nové číslo, který jde dosadit na pozici z řady. Pokud nějaký číslo lze použít, zapíšou se oboje čísla do polí a zavolá se další nextGen(). Jestliže ne přepne se switch (a) znovu na 1 a hledá se další číslo z řady, které lze dosadit na momentální pozici.

3.2.3 Žádné číslo z řady nelze použít na současnou pozici Px

Jakmile nelze žádný číslo z řady použít na současnou pozici, přepne se switch (a) na 3 a znovu se zavolá metoda opravy. Tato krabička (case) obsahuje kód s vertikální opravou.

```
case 1: //vyhledá číslo z areaXY[Px, y], které jde dosadit na areaXY[x, y].
    {
        newNum = 0;
        if (Px > 1)
        {
        }
        else //Px < 1; došel na konec -> Vertikální oprava. (prohození ve čtverci nad y)
        {
            a = 3; pos = 0;
            repairX();
        }
```

Obrázek 31 - Algoritmus došel na začátek řady, tak volá vertikální opravu

3.2.4 Shrnutí algoritmu generování Sudoku

Takto se pokračuje dál v dalších zkouškách možností prohození čísel, dohromady v 6 case, dokud nenajde čísla, které prohodit lze nebo nezjistí, že už nic podle momentálního algoritmu prohodit nejde. **Switch/case** jsem vybral, protože pro konkrétní metodu velmi jednoduchý a přehledný, narozdíl např. od podmínky **if.**

```
//Poslaní dalšího generatoru();
private void nextGen()
{
    g = 0; //reset uložení generátu
    firstChange = true;
    pripocet = true;

    first = true;
    for (int i = 1; i <= 10; i++)
    { SaveX[i] = 0; }

    generator();
}
```

Obrázek 32 – metoda NextGen()

Jakmile hodnoty prohodit jdou, přiřadí se číslice na konkrétní souřadnice v poli a zavolá se metoda **„NextGen()“**. Tato metoda obnoví všechny důležité proměnný a pole používané při hledání čísla či prohazování na původní hodnoty, nastaví přípočet x a zavolá funkci generator(), což je hledání nové náhodné číslice na nové pozici x a vše pokračuje od znova dál. Když už žádné číslo nejde použít na současnou pozici a ani nic podle momentálního kódu prohodit, spustí se časovač (timer), který hned při prvním tiku zavolá metodu „newgame()“ pro generování celého Sudoku od znova a vypne se. Tohle je výborný ošetření proti nekonečné rekurzi, která shodí celou aplikaci, více v kapitole 4.2.

4 Řešení zajímavých problémů

4.1 Divné označování shody

```
if (shodator)
{
    if (ShodyXY[x, y] == 1 && vstpXY[x, y] == 2) //DIVNÁ chyba
    { gfx.DrawRectangle(Selector, 50 + x * 50, 50 + y * 50, 50, 50); }
    else if (ShodyXY[x, y] == 1 && vstpXY[x, y] == 1) //Označení shod.
    { gfx.DrawRectangle(ShodaSelector, 50 + x * 50, 50 + y * 50, 50, 50); }

}
```

Obrázek 33 - Chybně fungující podmínky pro označování shod s vloženým číslem

V dobách, kdy byl kód napsaný tímto stylem, políčko s vloženým číslem se často chybně označovalo jako shodné s jiným číslem podle pravidel Sudoku.

```
if (shodator)
{
    if (ShodyXY[x, y] == 1) //Označení shod.
    {
        if (vstpXY[x, y] == 1)
        { gfx.DrawRectangle(ShodaSelector, 50 + x * 50, 50 + y * 50, 50, 50); }
        else
        { gfx.DrawRectangle(Selector, 50 + x * 50, 50 + y * 50, 50, 50); }
    }
}
```

Obrázek 34 - Správně fungující podmínky pro označování shod s vloženým číslem

Od doby co jsem kód jednoduše předělal na tuto variantu, už vše funguje v pořádku, i když všiml jsem si, že občas to i tak ukáže shodu špatně. Příliš nerozumím tomu, kde je chyba. Vše by mělo fungovat v pořádku jak s 1. verzí, tak s 2. verzí tohoto kódu.

4.2 Ošetření proti nekonečné rekurzi

Jakmile je podle momentálního algoritmu potřeba generovat Sudoku od znova, spustí se časovač („timer“). Tento časovač hned při prvním tiknutí zavolá metodu NewGame() pro vygenerování nového Sudoku a vypne se.

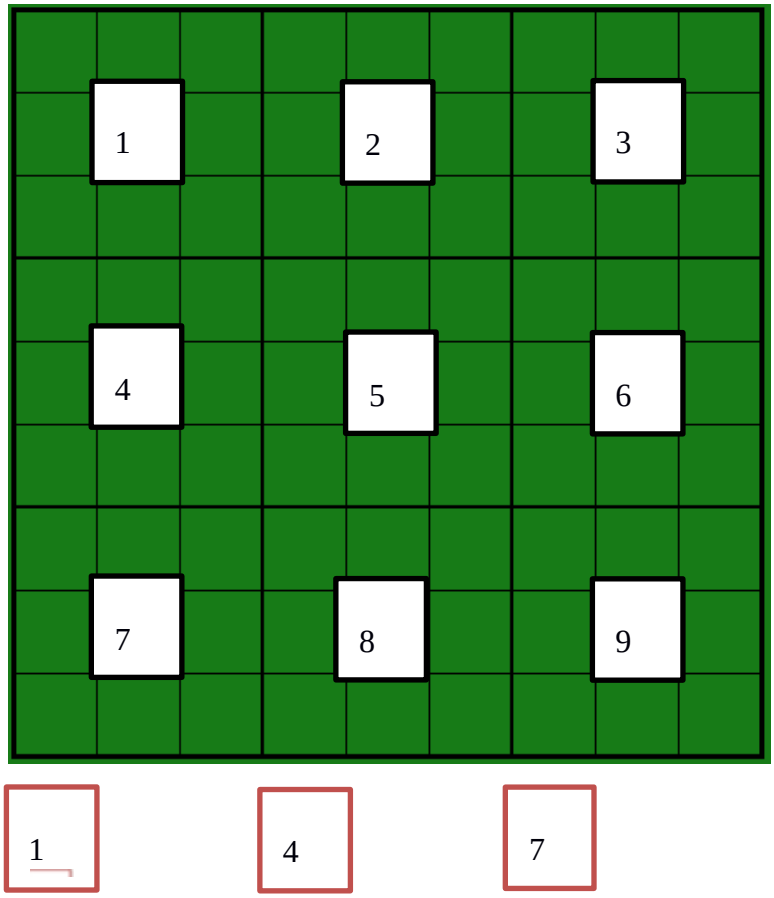
```
private void timer_Tick(object sender, EventArgs e)
{
    pc++;
    lbTimer.Text = "Počet opakování cyklu: " + pc;
    //lbTimer.Visible = true;

    timer.Enabled = false;
    NewGame();
}
```

Obrázek 35 – Timer: ošetření proti nekonečné rekurzi

Zásobník se tím z velké části vyprázdní. Dříve, když nové generování Sudoku bylo nastavený pouze na zavolání metody NewGame() nebo stisknutí tlačítka „New Game“ bylo možný rychlým klikáním tlačítka „New Game“ celý program shodit.

4.3 Zjištění začáteční X souřadnice pro prohození čísel při vertikální opravě

Jedním z mých největších problémů při vytváření generátoru Sudoku bylo vymyslet výpočet začáteční x souřadnice pro zkoušení prohození čísel ve stejné řadě a čtverci.



Obrázek 36 - Zjištění začáteční X souřadnice ve čtvercích

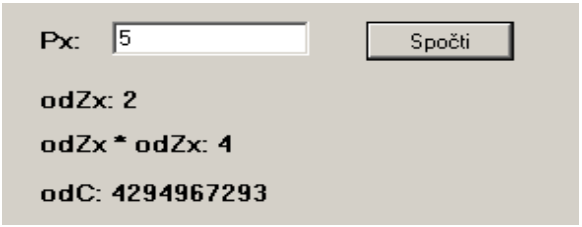
Čísla s černým ohraničením představují souřadnice čtverce (z) a číslice pod čtverci představují souřadnice pozice, které je z konkrétních čtverců třeba získat. Zkoušel jsem různý počty, ale nejlepší počet, který mě napadl bylo vzít souřadnici řady (x) , připočíst ji o 2 a vydělit 3 [odZx = (Px + 2) / 3]. Což znamená, že pokud se prohazované číslo nachází v prvním čtverci v řadě, tak odZx = 1; v druhým odZx = 2; a ve třetím odZx = 3;. Teď je ovšem potřeba 1.číslo odZx o nic nepřipočítat , druhé přičíst o 2 a třetí přičíst dokonce o 4, aby seděly počty. Na nic jsem dlouho nepřicházel a tak jsem vymyslel, že si to zjednoduším switchem v programováním, za cenu více zdrojového kódu.

```
sw = (Px + 2) / 3;
switch (sw)
{
    case 1:
        odZ = sw;
        doZ = odZ + 2;
    case 2:
        odZ = sw * 2;
        doZ = odZ + 2;
    case 3:
        odZ = sw * 2 + 1;
        doZ = odZ + 2;
```

Obrázek 37 – Switch pro výpočet začáteční X souřadnice ve čtverci

Tímto jsem měl ve zdrojovém kódu docela chaos. Pro 1 výpočet se ve **switch**i s **6 case** nalézal další **switch** s dalšími **3 case**. A tak mě napadlo vytvořit výpočet **odZx * odZx**, ten v prvním a druhým případě sedí (1 a 4), ale ve třetí je třeba odečíst 2. Pro to jsem vytvořil pomocnou nezápornou proměnnou **uint odC**, která má vzoreček **odC = odZx * odZx – 7;** a tuhle hodnotu jsem odečetl od odZx * odZx. Jelikož se jedná o nezápornou proměnnou typu **uint**, předpokládal jsem, že v prvních dvou případech bude 0 a ve třetím 2. Ovšem, jakmile se proměnná dostane do záporný hodnoty, jde od svý maximální hodnoty na nižší hodnoty, podle záporného odečtu.¨

```
int odZx = (Px + 2) / 3;
uint odc = ((uint)odZx * (uint)odZx) - 7;
int odZ = odZx * odZx + (int)odc;
int doZ = odZ + 2;
```



Obrázek 38 – Chybný výpočet začáteční X souřadnice ve čtverci a simulace tohoto výpočtu

Tudíž tohle taky nebylo možný, ale nakonec jsem to dokázal vymyslet. Stačí od odZx * odZx odečíst (odZx / 3) * 2, jelikož **odZx / 3** má pouze ve třetím čtverci v řadě hodnotu 1 a v ostatních 0. Celý výpočet je tedy ve tvaru **odZ = odZx * odZx - ((odZx / 3) * 2);**. Tímto dostanu veškeré začáteční X souřadnice ve čtverci, které potřebuji – **1, 4 a 7**.

```
case 4: //zkouška prohození čísel ve shodě Y s newNum.
    {
        if (first) //vložení základních hodnot
        {
            int odZx = (Px + 2) / 3;                //zjištění počtu čtverce v řadě.
            odZ = odZx * odZx - ((odZx / 3) * 2);   //první pozice v čtverci.
            doZ = odZ + 2;                          //poslední pozïce ve čtverci.
            Px = odZ;
            first = false;
        }
```

Obrázek 39 - Správný výpočet začáteční X souřadnice ve čtverci

5 Testování aplikace

Jediným testerem této aplikace jsem byl já, autor tohoto programu: **Jakub Jankovec**. Hru pomocí tohoto programu si zahrálo více mých kamarádů, ale nikdo z nich nenašel žádnou chybu nebo nedostatek. Aplikace by nyní měla fungovat bezvadně.

6 Závěr

Téma práce: Sudoku generátor.

Cílem práce bylo vytvořit generátor náhodného Sudoku zadání, který si bude uživatel této aplikace následně může zahrát. Program by měl umět dle pravidel této hry správně ukazovat kde jsou v řadě, sloupci či čtverci shody s vloženým číslem. Hráč si před začátkem hry může vybrat jakou bude chtít hrát obtížnost, tj. „lehká, střední, těžká a libovolná“. Jakmile se dosadí na všechny prázdná políčka správná čísla, program automaticky zahlásí výhru a poblahopřeje uživateli. Dalším cílem práce bylo vytvořit tlačítko, které při stisknutí poradí, jaká číslice se na dané políčko může hrát. Hra je lokalizovaná v angličtině, češtině a němčině.

Zhodnocení: Myslím, že se mi povedlo výborně naprogramovat to co jsem chtěl, sice bych ještě mohl vylepšit algoritmus na generování čísel, ale pro své účely to momentálně bohatě stačí. Obzvlášť radost mám z vytvoření obtížnosti „LevelMode“, která Sudoku posouvá na novou úroveň hratelnosti. Napadlo mě s každou novou úrovní změnit pozadí na nějaký jiný hezký obrázek, též vytvořit něco jako skóre za čas vyřešení Sudoku nebo jiné legrace. Do budoucna si ještě určitě trochu pohraju s algoritmem generování Sudoku a vytvořím tlačítko na posouvání historie tahů a uložení či načtení celý rozehraný hry.

7 Seznam použitých zdrojů

(1) Google (http://www.google.cz/) [online] => vyhledávání potřebných informací na internetu.
(2) Stack overflow (http://www.stackoverflow.com/) [online] => většina dostupných informacích na internetu [vyhledáno prostřednictvím služby Google search].
(3) Honza Novák, kamarád a spolužák ze třídy: Velmi užitečná rada k objektovému programování. [citováno 13.5.2016].
(4) Moje hlava a vědomosti. [citováno právě teď]

8 Seznam použitého softwaru

Microsoft Visual Studio 2010
=> Vývojářské prostředí; psaní kódu pro program a kompilování programu.
[VS 2010 je pro .NET lepší než novější verze, jelikož podporuje i systémy Windows XP]

Notepad++ v 6.9.2 (GNU public licence) => Výborný textový editor pro alternativní psaní kódu pro program.

LibreOffice 5.1.3.2 (GNU public licence) => Návrh a psaní dokumentace.

Windows Snipping Tool => Tvoření screenshotů/obrázků pro dokumentaci.

Microsoft Paint => Úprava screenshotů/obrázků pro dokumentaci.

9 Seznam obrázků