TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Program pro návrh vzoru díla pro pletací stroj

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Jana Vondráčková**
Vedoucí práce: Ing. Martin Diblík, Ph.D.

Liberec 2017

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Design software for flat knitting machine

Master thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 1802T007 – Information Technology

Author: **Bc. Jana Vondráčková**
Supervisor: Ing. Martin Diblík, Ph.D.

Liberec 2017

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2016/2017**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jana Vondráčková**
Osobní číslo:	**M14000185**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Program pro návrh vzoru díla pro pletací stroj**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Definujte požadavky na funkcionalitu a uživatelské rozhraní nové verze aplikace pro návrh vzoru díla pro plochý pletací stroj.
2. Navrhněte datové struktury a algoritmy zajišťující požadovanou funkcionalitu programu. Ve zvoleném vývojovém prostředí aplikaci naprogramujte.
3. Ověřte funkcionalitu aplikace, vytvořte elektronickou nápovědu a uživatelskou příručku k programu.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **40–50 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **Petzold, Ch. Programování ve Windows. Computer Press, Praha, 1999**

[2] **DRAYTON, Peter, Ted NEWARD a Ben ALBAHARI. C# v kostce: pohotová referenční příručka. 1. vyd. Praha: Grada, 2003, xxi, 764 s. ISBN 80-247-0443-9.**

[3] **BAYER, Jürgen. C# 2005: velká kniha řešení. Přeložil Jiří. KOLÁŘ. Brno: Computer Press, 2007. ISBN 978-80-251-1620-3.**

Vedoucí diplomové práce: **Ing. Martin Diblík, Ph.D.**
Ústav mechatroniky a technické informatiky

Datum zadání diplomové práce: **10. října 2016**

Termín odevzdání diplomové práce: **15. května 2017**

Technická univerzita v Liberci

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2016

Prohlášení

Byla jsem seznámena s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědoma povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracovala samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 2. 1. 2017

Podpis: Jana Vrbková

Poděkování

Ráda bych poděkovala mému vedoucímu diplomové práce panu Ing. Martinu Diblíkovi, Ph.D. za vstřícnost a vedení. Dále mým nejbližším, kteří mě neustále podporovali a vytvářeli tu nejlepší atmosféru v průběhu celého mého studia.

Abstrakt

Výsledná aplikace umožňuje správný návrh vzoru díla pro plochý dvoulůžkový pletací stroj. Rozšiřuje původní verzi návrhového prostředí o několik funkcí a optimalizuje některé stávající nástroje. Hlavní podstatou této práce je vytvořit funkční aplikaci určenou pro novější systémy Windows. Vývojovým prostředím je Visual studio 2013, jazyk C#. Zároveň se snaží vyhovět všem kladeným požadavkům, mezi které patří uživatelská přívětivost. Z toho důvodu vznikla uživatelská nápověda pomocí programu HelpNDoc 4, která je spustitelná přímo v běžící aplikaci. Zajímavé části programu jsou uvedeny ve vybraných ukázkách.

Klíčová slova

Pletací stroj, návrh pleteného vzoru, vlastní komponenta, vývoj aplikace.

Abstract

Final application allows the correct design of product pattern for double flat knitting machine. Extends the original version of the design environment of several functions and optimizes some tools. The main purpose of this thesis is to create a functional application designed for the newer Windows systems. Chosen development environment is Visual Studio 2013 C# language. It also seeks to satisfy all the imposed requirements, including user-friendliness. For that reason, it was created user help by using HelpNDoc 4, which is executable in the running application. Interesting parts of the program are listed in the selections.

Keywords

Knitting machine, design of knitted pattern, user component, development of application.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Úvod

Tato práce vznikla za účelem vyřešení problémů se současnou verzí programu pro návrh vzoru díla pro pletací stroj firmy Tonak a.s., kde v úseku designu probíhá návrh pletených polotovarů v několika softwarových aplikacích, které jsou určeny pro různé pletací stroje. Nejdůležitějším iniciátorem potřeby k vytvoření nové aplikace byla možnost správného běhu aktuální verze pouze na operačních systémech Windows XP, což bylo po firemních aktualizacích nepraktické. Po konzultacích s uživateli a po zhlédnutí výrobního procesu byly definovány další požadavky.

V následujícím textu jsou nejprve sepsány informace o pletacím stroji a výsledném textovém souboru, který je určen pro definici postupu pletení stroje. Následují stručné informace o nové aplikaci ELCAP Designer 2.0 a uvedení požadavků nutných pro její realizaci. Dále je podrobně uveden postup řešení a jsou popsány důležité objekty programu. Nakonec jsou přiblíženy prvky uživatelské přívětivosti, které se vyskytují v celé aplikaci.

1 Stroj, pletení

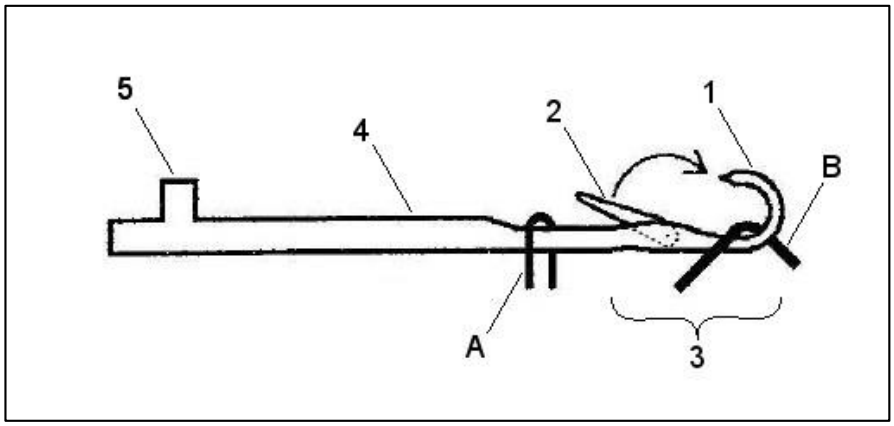
V této části bude obecně vysvětlen chod pletacího stroje, pro nějž je výsledek této práce určen, dále některé jeho důležité části a možnosti pletení. Nakonec je popsán soubor, podle kterého je výrobek pleten a který definuje výstup návrhové aplikace.

1.1 Pletací stroj

Pletená díla ve firmě Tonak a.s. jsou realizována mimo jiné na pletacích strojích Elektrocap, které byly vyrobeny v 90. letech 20. století v Brně soukromou firmou. Zpracování modernizace jejich řídící jednotky, která byla původně na bázi jednočipového mikroprocesoru, proběhla na Katedře textilních a jednoúčelových strojů, Fakultě strojní Technické univerzity v Liberci v letech 2013-2014. Jedná se o plochý dvoulůžkový pletací stroj (viz Obr. 1) s jazýčkovým typem jehel, kterých je 160. To znamená, že má dvě synchronizované pletací hlavy, plete dva výrobky současně. Stroj je schopen využívat maximálně 3 příze najednou (3 barvy současně na jeden produkt). Pomocí paměti Flash je soubor se vzorem pletení přesunut do paměti řídicího systému. Pokud je vzor bez chyby a obsahuje všechny nezbytné informace, je možné jej spustit.

Obr. 1: Pletací stroj po modernizaci (vlevo nová řídicí jednotka)

Výsledkem pletení stroje je zátažný typ pletenin, vznikající postupným proplétáním příze na jednotlivých jehlách ve směru řádku. Je to tzv. plošná textilie, což znamená, že dva rozměry výrobku jsou podstatně větší než třetí. Pletení probíhá tak, že se přes jehelní lůžka pohybují saně s tzv. zámky, které při pohybu vytahují a zatahují jehly, přes které projedou. Zároveň je tažena vodičem nit, kterou jehly během chvilkového vysunutí zachytí a během pohybu zpět vytvoří nová oka. Odhození oka minulého a zároveň zachycení nového, probíhá prostřednictvím speciální části jehly – jazýčku, následovně. Jehla je v základní poloze, kdy drží visící očko, probíhá vysunutí jehly nahoru do chytové polohy a následně do uzavírací polohy. Při přechodu do druhého stavu je pomocí starého očka odklopen jazýček a očko pak přepadne přes něj na stvol jehly. Při zpětném pohybu jehly dojde ke kladení a zachycení nové nitě. Jehla stále klesá a staré očko zvedá jazýček, což je poloha kladení. Když je háček pod odhazovací rovinou dojde k odhození starého očka a nové je vytvořeno. Všechny části jehly jsou viditelné na Obr. 2: háček (1), jazýček (2), hlava jehly (3), stvol (4) a kolénko (5). Dále zde vidíme staré očko (A), které bude odhozeno, a nově nabrané (B).



Obr. 2: Jazýčková jehla [5]

1.2 Soubor se vzorem

Instrukce pro pletení jsou zapsány v textovém souboru, který vzniká při kompilaci nakresleného návrhu v aplikaci. Soubor musí obsahovat všechny části a nesmí obsahovat chyby, jinak jej nebude možné ve stroji spustit. Struktura tohoto souboru je daná a není předmětem řešení této práce. Obsahuje dvě hlavní části „Hlavičku“ a „Tělo“, které jsou následně lépe popsány.

- **Header (Hlavička)**

  Toto je především informační část, díky níž je možné identifikovat základní vlastnosti a nastavení pletení vzoru. Je uvedena klíčovým slovem *[Header]*, poté následuje 7 řádků s informacemi (viz Zdrojový kód 1), jako je například číslo návrhu, nebo první a poslední použitá jehla v celém vzoru (detailnější popis níže). Na prvním místě řádku je vždy číslo s předepsaným počtem cifer, následuje oddělovací středník a po něm mohou následovat jakékoliv pomocné informace, protože položky za středníkem nejsou v řídicím systému stroje zpracovávány. Na posledním řádku je místo čísla text v jednoduchých uvozovkách, který je využíván jako stručný popis vzoru.

```
;[Header]
000001;pattern number
00531;pattern row number off = 109
001;pattern zaplet = 0/1 = 104
010;pattern rychlost off = 105
001;pattern min jehla off = 106
112;pattern max jehla off = 107
'Vzor 11_5';pattern name
```

Zdrojový kód 1: Hlavička výsledného souboru

Na konci některých řádků se nacházejí části označené uvozujícím slovem *off*, které značí příslušný offset. Tyto hodnoty byly užitečné při vývoji programu řídicího systému stroje a bylo by možné je v současnosti vynechat. Detailnější popis významu jednotlivých řádků je následující.

- *pattern number* – označuje šestimístné číslo pro pořadí vzoru
- *pattern row number* – celkový počet řádků těla vzoru
- *pattern zaplet* – tento příznak nabývá pouze hodnot „000“ a „001“, kdy jednička značí, že začátek a konec výrobku se na konci splétají, při nule se nesplétají
- *pattern rychlost* – výchozí rychlost pletení stroje
- *pattern min jehla* – nejmenší číslo jehly, které je v celém návrhu použito
- *pattern max jehla* – nejvyšší číslo jehly, které je v celém návrhu použito
- *pattern name* – slovní označení/komentář vzoru, slouží především pro jeho identifikaci při omezeném prohlížení přímo na stroji

- **Body (Tělo vzoru)**

  Zde se po klíčovém slově *[Body]* nachází samotný vzor, který je určen k pletení na stroji. Každý jednotlivý řádek textu odpovídá řádku v pletenině (viz Zdrojový kód 2). Na jeho začátku jsou uvedeny následující informace: číslo řádku, rychlost pletení řádku, první použitá jehla, poslední použitá jehla. Jednotlivá čísla jsou oddělena čárkou a za poslední je v jednoduchých uvozovkách instrukce pro pletení. Každá jehla je znázorněna jedním znakem, a tak, jak již bylo řečeno, znaků je celkem 160 (maximální počet použitelných jehel stroje). Na konci souboru se nevyskytuje žádný speciální znak pro ukončení, pouze v hlavičce je uveden celkový počet řádků.

```
[Body]
00001,002,001,111,'555555555055555555550555555555555555555555...'
00002,002,001,110,'555555555055555555550555555555555555555555...'
00003,010,001,111,'555555555055555555550555555555555555555555...'
00004,010,001,111,'555555555055555555550555555555555555555555...'
00005,010,001,051,'555555555055555555550555555555555555555555...'
00006,010,042,050,'000000000000000000000000000000000000000005...'
00007,010,041,053,'000000000000000000000000000000000000000075...'
00008,010,001,052,'555555555055555555550555555555555555555555...'
```

Zdrojový kód 2: Ukázka těla vzoru výsledného souboru

Aby bylo možné tento soubor spolehlivě a bez chyb generovat, je třeba znát také zakódování jednotlivých akcí pro jehly, které se nacházejí v těle souboru se vzorem. Jak již bylo uvedeno, je možné použít maximálně 3 barvy, od čehož se odvíjí potřebné možnosti k zakódování. U každé z barev je také možno vykonat „chyt“, neboli „chytovou kličku“. Přičteme-li znak pro žádnou akci (jehla není použita), je to v součtu 7 různých znaků pro jednu jehlu.

Bitově jsou tyto akce v současnosti zakódovány následovně. Znak je reprezentován od nejvyššího bitu takto: $[b_3\ b_2\ b_1\ b_0]$. Výsledná hodnota se do generovaného souboru zapisuje jako znak hexadecimální soustavy.

- $\boldsymbol{b_0}$ – '0' pro jehlu pasivní, '1' pro jehlu aktivní (volenou)
- $\boldsymbol{b_1}$ – '0' jehla do základní polohy, '1' jehla do chytové polohy
- $\boldsymbol{b_3\,b_2}$ – '00' žádný vodič, '01' první vodič, '10' druhý vodič, '11' třetí vodič

2 Aplikace

V této kapitole je uvedeno základní seznámení s aplikací ELCAP Designer 2.0, jejím vzhledem, ale i strukturou programu. Tomuto návrhu předcházela analýza stávající aplikace, jejího použití a jejích nedostatků, které jsou popsány v Semestrálním projektu [11] a budou v následující podkapitole *Požadavky na novou aplikaci* stručně shrnuty.

2.1 Požadavky na novou aplikaci

Mezi hlavní cíle této práce patří zachování veškerých důležitých funkcí z předchozího návrhového prostředí, včetně podobnosti vzhledu (rychlejší zaškolení zaměstnanců). Dále také doplnění chybějících navrhovaných funkcí (např. export nakresleného vzoru), případně zavedení i těch doplňujících, které práci na návrhu usnadní. Původní aplikace je vytvořena ve vývojovém prostředí Delphi 7 a její největší nevýhodou je funkčnost pouze na operačních systémech Windows XP. Díky tomu také vznikl požadavek na aplikaci novou, která by byla funkční na systémech Windows 7 a vyšších. Na základě toho se autorka rozhodla aplikaci vytvořit v návrhovém prostředí Visual Studio 2013 v programovacím jazyce C#.

Na základě konzultace s paní Ing. Chumanovou a paní Ing. Červinkovou z oddělení designu firmy Tonak a.s. byly shromážděny požadavky a návrhy na dodatečné funkce v nové aplikaci v porovnání se současně používanou. Jejich přehled je stručně vypsán v následujících bodech.

- Možnost tisku návrhu vzoru výrobku, export do obrázku
- Náhled vzoru - složený obrázek
- Optimalizace kreslících nástrojů, módy myši
- Zobrazení informace o právě zvolených nástrojích
- Zobrazení informace o vybrané buňce
- Možnost opakování nakresleného vzoru
- Krok zpět a vpřed
- Globální změna barvy
- Uživatelsky nastavitelné zkratky úkonů

Vysvětlení těchto pojmů je uvedeno v Semestrálním projektu [11]. Blíže budou také popsány v dalších kapitolách, kde bude uveden postup při jejich řešení.

2.2 Stručný popis postupu práce

Díky zmíněné předchozí analýze, byla návrhová tabulka určena jako ústřední komponenta celého programu. Zastává nejdůležitější úkol, jímž je zprostředkování intuitivního a jednoduchého navrhování (nebo také kreslení) vzoru určeného k pletení, jehož výsledek bude možné zkompilovat do souboru určeného pro pletací stroj a také exportovat v takovém formátu, který bude jednoduché vytisknout. Po průzkumu několika volně dostupných komponent, které byly buď příliš pomalé, nebo nesplňovaly požadované funkce, se autorka rozhodla pro vlastní zpracování této komponenty. Tím začal celý postup návrhu. K lepší orientaci v síti tabulky bylo potřeba zobrazit okrajové pravítko. Během jeho vývoje byly řešeny problémy především s jeho posouváním dle scrollování v návrhové tabulce.

Následně byly k tomuto základu přidávány doplňkové funkce, jako je zobrazování informace o zvolené buňce, volba vodiče, ale také implementace různých akcí myši, a to především akce výběru. Tato akce je v této aplikaci implementována odlišně oproti původní verzi. Ke změně bylo přistoupeno z důvodu intuitivnějšího použití pravého tlačítka myši (jak je tomu v mnoha nejen návrhových prostředích). Dále bylo řešeno zobrazení informační tabulky vlevo od editační, která je také součástí výsledného souboru (úvodní informace na každém řádku). Vnitřní struktura dat této postranní tabulky byla dále rozšířena a použita pro držení několika dalších informací využitelných pro další funkce aplikace.

Po kompletaci zobrazení všech dat, které jsou obsaženy ve strojovém souboru, byla implementována jeho kompilace (ukládání vzoru) a také otevírání již existujících souborů. Díky těmto funkcím bylo možné otestovat a případně upravit editor pro reálná a objemnější data. To se stalo především u postranní informační tabulky, která neobstála svou rychlostí reakcí na změnu dat přesto, že obsahuje pouze 4 sloupce. Součástí tohoto vývoje bylo také zavedení několika kontrolních metod a funkcí, včetně vytvoření zprávy o nalezených chybách. Mezi nové funkce patří možnost exportovat vzor jako obrázek a náhled obrázku, který je užitečný především při navrhování vzoru s více vodiči.

Další novinkou, která byla přidána, je možnost vracet poslední změny v návrhu. To je provedeno prostřednictvím možností *Krok zpět* a *Krok vpřed*, které mají omezenou paměť počtu úkonů na 10. K tomu je použitý návrhový vzor Memento, který po testování na reálných datech dosahuje dostatečné rychlosti a při omezení paměti úkonů ani závažně nezvyšuje paměťové nároky.

Následně byly implementovány i původní speciální funkce nacházející se nyní v oblasti *Rozšířené*. Jedná se o nastavení výměny vodiče mezipohybem a funkce vytvoření doplňku řádku. Obě volby jsou užitečné při používání více vodičů.

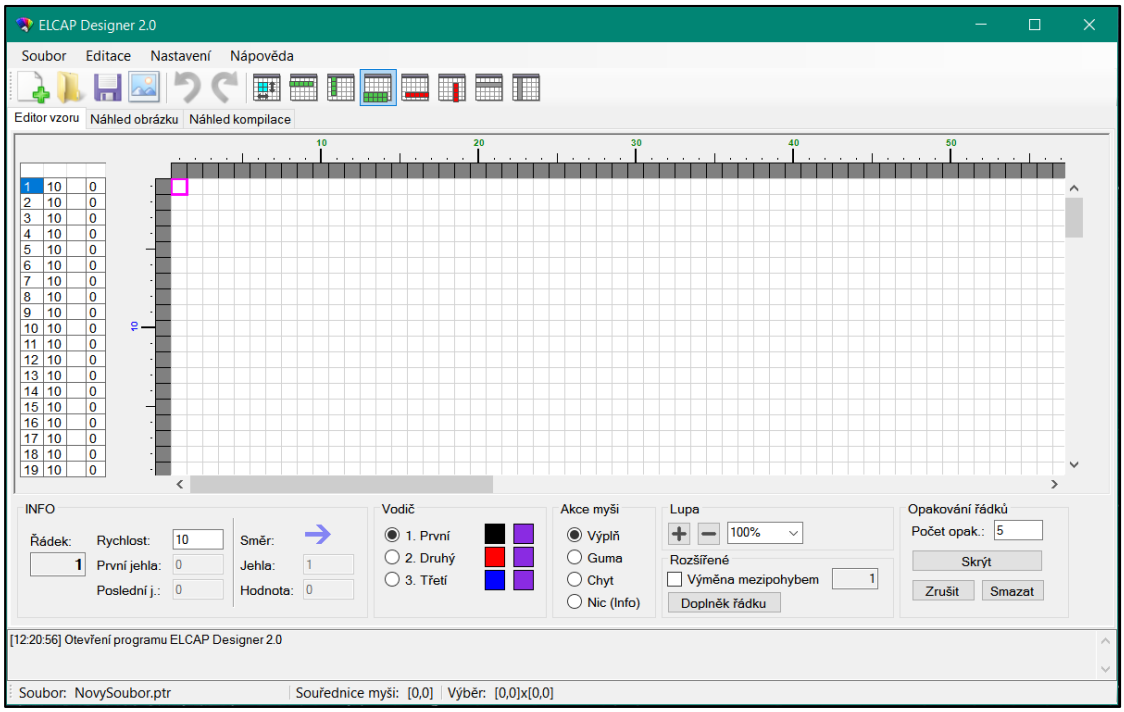
Jedním z nejnáročnějších byl návrh možnosti nastavení opakování vzoru, jelikož bylo nutné vymyslet smysluplné a uživatelsky přívětivé ovládání, zobrazení ale i chování. Jak se má návrh chovat v případě, že v něm existuje nastavené opakování a jak jej jednoduše znázornit v tabulce zabalené a rozbalené. Jaké chování je očekáváno, bude-li opakování zrušeno, nebo smazáno. Především také jak tuto informaci o opakování uložím do kompilovaného souboru, aniž bych narušila dosavadní strukturu. To bylo nakonec vyřešeno umístěním této informace ve formě tří čísel na konec uživatelského komentáře, za znak „|“.

V dalším kroku byly řešeny záložky pro *Náhled obrázku* a *Náhled kompilace,* které uživateli umožňují především kontrolu v zobrazených datech. Na konci veškerého vývoje byla aplikace rozšířena o zobrazování informací ve spodních panelech. Především bylo nutné zprostředkovat zobrazování výsledků proběhlých kontrol na datech a potvrzování některých úspěšně dokončených úprav. Dále je zobrazena informace o aktuálních souřadnicích myši a existujícím výběru v tabulce.

V průběhu navrhování probíhalo také testování aplikace na reálných i nereálných datech. Díky tomu bylo odhaleno několik neošetřených chyb a nedokonalostí v chování aplikace. Na závěr byla vytvořena uživatelská nápověda v programu HelpNDoc 4 [3].

2.3 Struktura vzhledu aplikace

Na Obr. 3 můžete vidět současný vzhled aplikace. Je vidět zachování umístění ovládacích prvků a tedy i struktury programu. Nejzásadnější změny se nacházejí v položkách hlavního horního menu, kde však díky slovnímu označení je použití snadné.



Obr. 3: Náhled výsledné aplikace

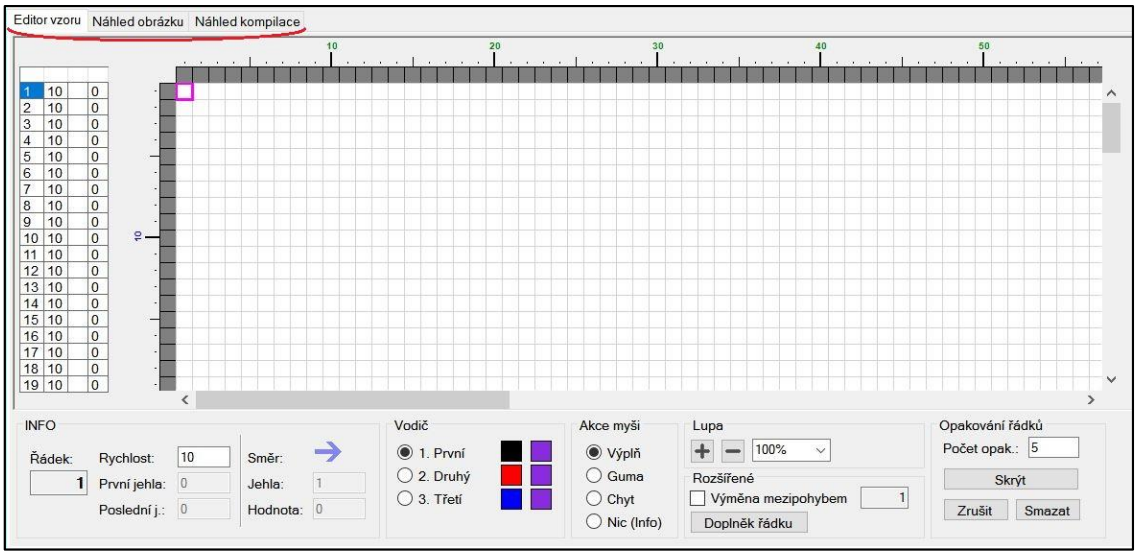
Struktura vzhledu aplikace by se dala rozdělit na 3 části – horní část se základními ovládacími prvky, střední část s návrhovými prvky a dolní část s informačními. Jejich umístění vychází z rozložení vzhledu běžných aplikací a také z aplikace původní. Jednotlivé oblasti jsou popsány následně.

Obr. 4: Pohled na horní ovládací prvky

V horní části (viz Obr. 4) je umístěno menu s položkami *Soubor*, *Editace*, *Nastavení* a *Nápověda*. Nacházejí se zde položky manipulující se soubory, které jsou v tomto případě pouze dvojího typu – *.ptr* (textové soubory se zkompilovaným vzorem) a *.jpg* (pro export grafického návrhu vzoru). v editaci je možné najít možnosti *Krok zpět* a *Krok vpřed*, následně několik úprav tabulky týkající se sloupců a řádků. Dále pod volbou nastavení se skrývají tři možnosti, a to *Možnosti vzoru, Nastavení adresářů* a *Výchozí hodnoty*. Nakonec je možné zobrazit pomocnou *Nápovědu* k programu. U některých položek je uvedena klávesová zkratka, kterou lze k jejímu vykonání použít.

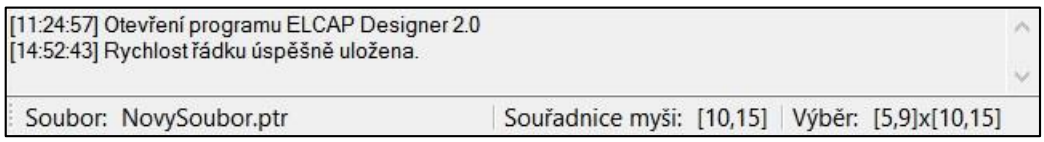
Pod tímto menu se nachází ovládací panel s tlačítky, která umožňují téměř totožné akce, ale s rychlejším přístupem.

Ve střední oblasti aplikace jsou umístěny všechny prvky, které umožňují návrh či editaci vzoru a podpůrné funkce k jeho kontrole. Pomocí *TabControlu* je možné přepínat mezi různými stránkami. Pod volbou *Editor vzoru* (viz Obr. 5) se nachází ústřední návrhová komponenta – tabulka. Ta je doplněna ve spodní části o několik ovládacích a informačních komponent. Další záložkou je *Náhled obrázku*, kde je možné v případě barevných vzorů nechat složit dvoj-řádky a podívat se tak na výsledný obrázek. *Náhled kompilace* umožňuje nastavit některé hlavičkové informace výsledného souboru a také jej celý zobrazit a tedy také zkontrolovat.



Obr. 5: Pohled na střední oblast aplikace

Spodní část slouží především jako informační (viz Obr. 6). Nachází se zde informační řádek, ve kterém se vypisují základní události včetně času, které aplikace úspěšně vykonala, případně zjistila nějakou chybu. V řádcích je možné se posouvat pomocí kolečka myši a šipek na panelu. Úplně posledním prvkem aplikace je řádek s několika vybranými informacemi, jako je např. souřadnice vybrané buňky, název souboru.



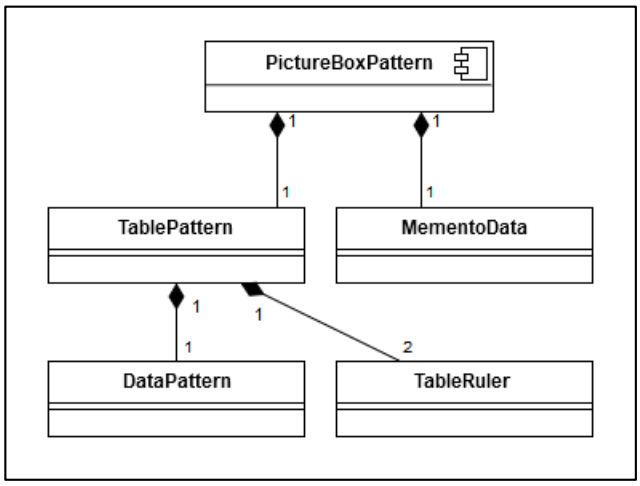
Obr. 6: Pohled na spodní informační oblast aplikace

2.4 Struktura programu

Program je rozdělen do několika tříd. Je složen z 5 formulářů (jeden hlavní), 2 vlastních komponent a 5 souborů tříd (včetně hlavního programu). Strukturu vlastní komponenty a jejích částí je možné vidět na zjednodušeném diagramu tříd na Obr. 7.

Nejdůležitější části jsou vidět pod komponentou *PicturBoxPattern*, kde jsou veškeré datové struktury vzoru. Tato komponenta zprostředkovává hlavní návrhovou tabulku, její měřítko a postranní informační tabulku. Je oddělena část dat, nesoucí informace a část pro jejich zobrazování do tabulky. To jsou třídy se jménem *DataPattern* a *TablePattern*. Aby byl uživatel schopen se v návrhové tabulce orientovat, bylo potřeba vytvořit ještě třídu *TableRuler*, která na základě poskytnutých informací ze třídy *TablePattern* vykreslí měřítko příslušné velikosti nad tabulkou a vlevo od tabulky. Dále komponenta *PicturBoxPattern* obstarává paměť vykonaných změn nad tabulkou prostřednictvím třídy *MementoData*, která, jak již název říká, je vytvořena na základě návrhového vzoru Memento pro kroky zpět a vpřed.

Hlavní formulář s názvem *Form1* vytváří vzhled celého programu. Další 4 formuláře s názvy *FormDirectoriesSettings*, *FormPatternSettings*, *FormValuesSettings* a *FormSpecifySelect* doplňují hlavní formulář především o různé nastavitelné možnosti vzoru ale i aplikace, díky nimž je možné práci lehce přizpůsobit uživateli. Poslední ze jmenovaných formulářů poskytuje možnost manuální (číselné) definice výběru. Poslední nejmenovanou komponentou je *ViewFinalPicture*, která umožňuje uživatelský náhled složeného obrázku vzoru. *Form1* také vlastní komponentu *PicturBoxPattern*.



Obr. 7: Zjednodušený diagram tříd vlastní komponnty PictureBoxPattern

3 Jednotlivé části programu

Tato kapitola se věnuje detailněji samotnému návrhu jednotlivých částí aplikace a postupu při jejich řešení. Vysvětleny budou vybrané třídy a jejich části, případně zdůvodněno jejich použití. Jsou zde uvedeny zdrojové kódy některých částí. V popisech jsou používána skutečná jmena proměnných a dalších datových struktur použitých přímo v kódu programu.

3.1 Vlastnosti nastavení projektu

V souboru určeném pro nastavení celého projektu (*Settings.settings*) jsou vytvořeny položky, které se využívají v celém programu pro různé účely. Všechny položky jsou vázány na uživatele, aby bylo možné je měnit dle jeho potřeby, nebo potřeby programu. Vyskytují se zde nejen uživatelská nastavení, ale i nastavení určená pro jednotlivé soubory vzorů, se kterými se právě pracuje. Jejich pojmenování se odvíjí od oblasti použití a dále konkrétního významu – např. *Dir_Opening*, *Patt_Number*.

3.2 Data vzoru

Třída, která se stará o veškeré manipulace s daty určenými pro pletací stroj, se jmenuje *DataPattern*. Existují dva druhy konstruktoru. Jeden pro případ vytváření čistých výchozích dat s parametrem počtu řádků a druhý pro otevírání dat z existujícího souboru s parametrem adresářové cesty k souboru. V případě nahrávání dat ze souboru je volena metoda *LoadDataPatern(string path)*, která pomocí *StreamReaderu* čte dokument po řádcích a orientuje se pomocí uvozujících slov [Header] a [Body]. Důležitá hlavičková data (číslo vzoru, komentář, …) jsou načtena a uložena do vybraných položek v souboru nastavení projektu.

Po detailní analýze a poznání všech částí výsledného kompilovaného souboru došlo k návrhu struktur jeho jednotlivých částí. Data v tělové části souboru byla rozdělena na dvě struktury – *dataPattern* (hodnoty jehel) a *prefixDataPattern* (předvolby jednotlivých řádků). Jako obalový typ pro jednotlivé řádky byl zvolen *List<T>*, a to především z důvodu indexového přístupu k řádkům a jednoduchému vkládání nových řádků kamkoli v listu bez nutnosti dynamického vytváření polí. Řádky proměnné *dataPattern* jsou tvořeny z polí typu *char* kvůli výslednému zápisu akcí pro jednotlivé jehly – pro každou jehlu jeden znak vybraný z daných 7 možností. Pro řádky uchovávané

v *prefixDataPattern* je vytvořena jednoduchá třída s názvem *TPrefixData*. Její část je možné vidět ve Zdrojovém kódu 3. Tato struktura vnitřních dat byla zvolena především pro jednoduché použití v kódu programu (kompatibilita typů) a kvůli orientaci v textu.

```
public class TPrefixData
{
    public int lineNumber;
    public int speed;
    public int firstNeedle;
    public int lastNeedle;
    public char yarnUse;
    public int repetition;
    public bool changeMovement;

    public TPrefixData(int lineNum)
    {
        this.lineNumber = lineNum ;
        this.speed = Properties.Settings.Default.Patt_DefaultSpeed;
        this.firstNeedle = 0;
        this.lastNeedle = 0;
        this.yarnUse = '0';
        this.repetition = 0;
        this.changeMovement = false;
    }
...
}
```

Zdrojový kód 3: Třída pro manipulaci s daty v proměnné prefixDataPattern

Informace o řádcích v proměnné prefixDataPattern se využívají v mnoha funkcích a metodách. Jedním z nejdůležitějších úkonů je změna statusu pro jednotlivé buňky (jehly) pomocí metody *ChangeStatus(int row, int col, char status)*, kde při jejich změně je kontrolováno a vyplňováno jedno důležité pravidlo. Řádek, ve kterém dochází ke změně statusu buňky, musí tuto změnu povolovat. Každý lichý a sudý řádek (uživatelsky)[1] ihned po něm následující, musí být vyplněn stejným vodičem, jeho chytovým znakem, nebo „nulou“. Při této akci se využívá položka *prefixDataPattern.yarnUse*, díky které je kontrola prováděna. K těmto účelům se zde nachází slovník *lineStatusAllow*, díky kterému je možné jednoduše zjistit, jaký chytový znak přísluší kterému vodiči. Část vyplňovací metody je uvedena ve Zdrojovém kódu 4.

[1] Je nutno poznamenat, že v celém programu bylo třeba rozlišovat dvojí indexování – uživatelské začínající od 1 a programové začínající od 0. Vysvětleno v podkapitole Indexování.

```
public void ChangeStatus(int row, int col, char status)
{
    if (lineStatusAllow[status] == this.prefixDataPattern[row].yarnUse
        || this.prefixDataPattern[row].yarnUse == '0') //pro vodice
    {
        this.dataPattern[row][col] = status;
        //zmena statusu + nastaveni prislusne barvy dvojradku
        if (dataPattern.Count % 2 == 1 && row == (dataPattern.Count - 1))
            return;

        if (this.prefixDataPattern[row].yarnUse == '0' && (row % 2) == 0)
        {
            this.prefixDataPattern[row].yarnUse = lineStatusAllow[status];
            this.prefixDataPattern[row + 1].yarnUse = lineStatusAllow[status];

        }
        else if (this.prefixDataPattern[row].yarnUse == '0' && (row % 2) == 1)
        {
            this.prefixDataPattern[row].yarnUse = lineStatusAllow[status];
            this.prefixDataPattern[row - 1].yarnUse = lineStatusAllow[status];
        }
    }
    else if (status == '0') //pro gumu
    ...
}
```

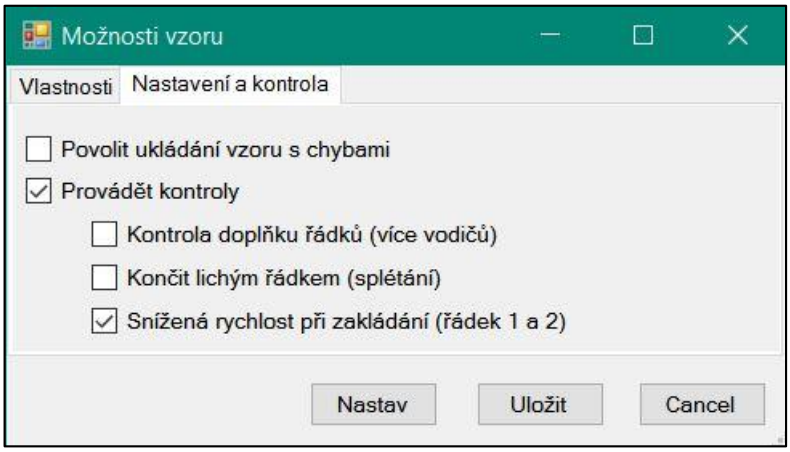
Zdrojový kód 4: Ukázka metody pro změnu statusu buňky dat

3.2.1 Kontrola dat

Součástí této třídy musí být také kontrola dat, která je v prvním případě založená na možnostech pletacího stroje. Jedná se o kontrolu zakončení lichým řádkem a kontrolu doplňku řádku v případě použití více vodičů (barevný vzor). Lichým řádkem může vzor končit v případě, je-li výsledné dílo určené ke konečnému splétání konců – stroj na konci pletení spojí začátek a konec produktu. Pokud je zvolena kontrola doplňku řádků, probíhá složitější procházení dat po dvou řádcích, kdy se kontroluje sudý řádek s následujícím lichým (uživatelsky), zda jsou k sobě doplňkem – v místech nul jednoho řádku se nachází výplň vodičem řádku druhého. To celé pouze v případě, že tyto řádky používají rozdílný vodič. Další jednoduchou kontrolou je snížená rychlost prvních dvou řádků, která musí být v tomto případě nastavena na rychlost 2. Pro každou z těchto voleb existuje samostatná metoda, která v případě nalezení chyby vytvoří o ní krátkou zprávu a ta je zobrazena ve spodním informačním panelu.

Tyto kontroly se povolují v hlavním menu aplikace pod položkou *Nastavení* → *Možnosti vzoru*. Vzhled tohoto dialogu je vidět na Obr. 8. Důležité jsou první dvě volby,

kdy zaškrtnutí položky *Povolit ukládání vzoru s chybami* umožní ukládání souboru i přesto, že chyby obsahuje (předpokládá se při průběhu návrhu). Nicméně, pokud je navíc zaškrtnuta i následující položka *Provádět kontroly*, výpis informací o chybách se zobrazí vždy. Pokud první položka není zaškrtnuta, uložení souboru vůbec neproběhne, pokud chyby existují.



Obr. 8: Nastavení kontroly nad daty

3.2.2 Akce s řádky a sloupci

Původní aplikace zahrnovala také základní úkony prováděné s celými sloupci a řádky návrhové tabulky. Jedná se o akce vkládání čistých, odstraňování a mazání obsahu řádků a sloupců. Protože tabulka manipuluje se svojí velikostí rozdílně, než v předchozí aplikaci, byla přidána ještě funkce přidání zadaného počtu řádků na konec tabulky.

```
public void InsertNewRows(int startIndex, int count)
{
    TranslatePrefRows(startIndex, count);

    for (int i = 0; i < count; i++)
    {
        char[] row = new char[160];
        PopulateRowStatus(row, '0');
        this.dataPattern.Insert(startIndex,row);

        TPrefixData prefixRow = new TPrefixData(startIndex+1);
        this.prefixDataPattern.Insert(startIndex, prefixRow);
        startIndex++;
    }
}
```

Zdrojový kód 5: Metoda pro přidávání řádků v datech

Akce prováděné s řádky jsou relativně jednoduchými úkony, je však potřeba je vždy provádět zároveň pro hlavní i prefixová data. Před vykonáním příslušné akce je dále

nutné provést „posunutí“ prefixových dat. Ty v sobě obsahují informaci o čísle řádku, která by již déle neplatila. K tomu slouží metoda *TranslatePrefRows*, která posune toto číslo o zadanou hodnotu (kladnou i zápornou) pro všechny řádky od počátečního indexu. Poté už je možné jednoduše vložit (odebrat) řádky na příslušnou pozici, jak je vidět ve Zdrojovém kódu 5.

Přidávání a odebírání sloupců se provádí jiným způsobem, a to nejen proto, že se nejedná o základní funkci listu, ale i proto, že sloupce v podstatě nelze přidávat a odebírat – jejich počet musí být vždy roven 160 (počet jehel stroje). Akce musí být provedeny pomocí posunu dat dopředu (odebírání), nebo dozadu (přidávání). Z toho plyne, že pokud existují vyplněné buňky na konci tabulky z pohledu sloupců, budou tato data ztracena. Na druhou stranu se tato akce provádí pouze s hlavními daty, prefixová data je třeba pouze aktualizovat. Ve Zdrojovém kódu 6 vidíte metodu pro odebírání sloupce.

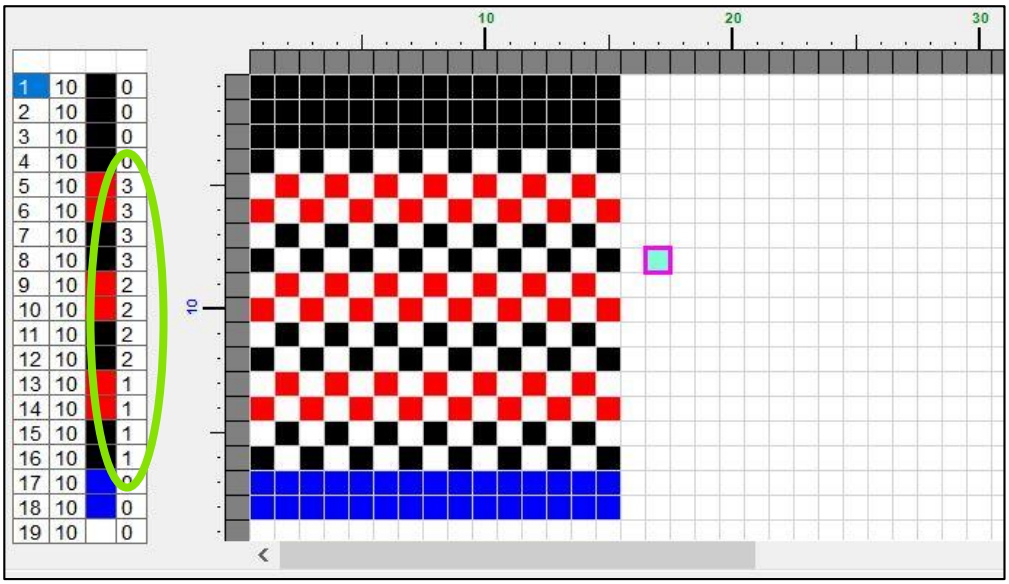
```
public void DeleteOneColumn(int colIndx)
{
    colIndx--;
    for (int row = 0; row < this.dataPattern.Count; row++)
    {
        for (int col = colIndx; col <= this.dataPattern[0].Length - 1; col++)
        {
            if (col == this.dataPattern[0].Length - 1)
               this.dataPattern[row][col] = '0';
            else
               this.dataPattern[row][col] = this.dataPattern[row][col+1];
        }
    }
    SetNeedlePrefixData();
    SetPrefixDataLineColor();
}
```

Zdrojový kód 6: Metoda pro odebírání sloupce v datech

3.2.3 Opakování vzoru

Funkce opakování vzoru je velikým ulehčením práce pro uživatele během navrhování. Téměř ve všech již existujících vzorech se totiž nějaké opakování vyskytuje, což je předurčeno typem výrobku, pro který je pletací stroj určen – pokrývky hlavy. Jak bylo v úvodu vysvětleno, stroj plete výrobky „ze strany“, neboli lem čepice je na levém okraji a špička na pravém. Tvarování čepice musí být v průběhu pletení souměrné, pokud má produkt pravidelný tvar. Výjimky existují pouze u barevných vzorů, které se na čepici neopakují (komplexní obrázek), nebo se z nějakého důvodu jedná o nepravidelný tvar.

Pro účely nastavování opakování byla rozšířena struktura třídy *TPrefixData o* položku *repetition* typu *integer*. Informace o opakování nějakého úseku se ukládá pomocí několika čísel. Pokud se opakování nevyskytuje, je vyplněna nulou. Pokud se ale vyskytuje, nastavují se hodnoty následovně. Úsek, který je určený k opakování, je označený velkým číslem – počáteční řádek hodnota 1000, meziřádky 2000, konečný řádek 3000, k těmto číslům je ještě přičtený celkový počet opakování. Tato čísla byla zvolena z důvodu volby maximálního počtu opakování – 999. Dále se sekvencím řádků sestupně od začátku přiřazuje hodnota jejich opakování, jednoduchý příklad na Obr. 9. Toto číslo se také následně zobrazuje v postranní informační tabulce. Je tedy rozlišena výchozí sekvence řádků od těch dalších. Jelikož při vytváření opakování jsou řádky vytvářené referencí na vybraný úsek, výsledné nastavení má pak řádky svázané a jakékoli změny se tak promítají do všech úseků. Před samotným vytvořením opakování je třeba provést několik úkonů. Kontrolu počtu řádků dat – v případě, že jich je málo je třeba vytvořit dostatečně dlouhý list. Dále pokud se pod úsekem, který byl určen k opakování, nacházejí vyplněné řádky, je třeba všechny posunout o správnou hodnotu, aby byly zachovány. To vše obstará metoda pro vkládání nových řádků popsaná výše. Po úspěšném vytvoření opakování je také nastavena položka *Rep_Exists* na *true* v souboru nastavení.



Obr. 9: Ukázka opakování

Aby bylo možné tuto funkci pohodlně používat, bylo třeba doplnit několik voleb, které s opakováním manipulují. Funkce skrývání je užitečná především pro následný export obrázku (o tom dále v textu). Po stisknutí tohoto tlačítka se skryjí všechny opakované řádky, kromě původních a je nastavena položka nast. *Rep_Hidden* na hodnotu

true. Protože zobrazování dat do tabulky probíhá na základě aktuálního obsahu, je skrývání vyřešeno přesunem dat do paměti opakování (proměnné stejného typu jako *dataPattern* a *prefixDataPattern*) a z originálních dat jsou odstraněny. Nastavení opakování pro řádky je však neporušeno.

Tato funkce je ale také dost omezující a pro její použití je třeba znát její pravidla. Během návrhu může dojít k potřebě změnit délku opakovaného úseku, nebo dokonce jej zrušit. K tomu jsou určená tlačítka zrušení a smazání. Volba zrušení slouží k „rozvázání“ závislosti opakovaných řádků a také ke zrušení nastavení opakování. Všechny řádky tedy zůstanou vyplněné, je vytvořena jejich vlastní instance dat a je díky tomu možné je opět upravovat jednotlivě. Smazání však nejen zruší nastavení opakování, ale také všechny řádky, kromě prvních, odstraní a posune tak celý zbytek návrhu opět nahoru. S řádky je teď možné manipulovat jako dříve. Opakování je vždy možné opět vytvořit, může však existovat v celém návrhu pouze jednou.

```
public void ResetRepetitionForRows()
{
    int[] info = FindRepetition();
    if (info == null) //musi existovat
        return;

    int start = info[0];
    int end = info[1];
    int count = end - start + 1;
    int repetitionNum = prefixDataPattern[start].repetition % 1000;

    for (int repeat = 1; repeat < repetitionNum; repeat++)
    {
        for (int rowToCopy = start; rowToCopy < (start + count); rowToCopy++)
        {
            int pos = rowToCopy + repeat * count;

            prefixDataPattern[pos].repetition = repetitionNum - repeat;
            prefixDataPattern[pos].speed = prefixDataPattern[rowToCopy].speed;
            prefixDataPattern[pos].changeMovement =
               prefixDataPattern[rowToCopy].changeMovement;

            dataPattern[pos] = dataPattern[rowToCopy];
        }
    }
    SetPrefixDataLineColor();
    SetNeedlePrefixData();
    Properties.Settings.Default.Rep_Exists = true;
}
```

Zdrojový kód 7: Metoda pro znovu-svázání řádků po kroku zpět a vpřed

Existuje ještě několik podpůrných a kontrolních funkcí a metod využitých pro správné fungování a zacházení v programu s opakováním. Například funkce pro nalezení opakování, které se využívá v mnoha případech pro kontrolu existence, nebo nalezení jeho začátku. Dále zde je metoda pro obnovení „svázanosti“ řádků, což bylo nutné využít u možnosti kroku zpět a vpřed, kvůli způsobu, jakým se data obnovují (více popsáno dále). Ta je uvedena ve Zdrojovém kódu 7.

```
public class TCellCoords
{
    public int rowNum; //cisla pro uzivatele a stroj
    public int colNum;
    public int rowIndx; //indexy pro pristup do dat
    public int colIndx;

    public TCellCoords()
    {
        this.rowIndx = -1;
        this.colIndx = -1;
        this.rowNum = 0;
        this.colNum = 0;
    }
    public TCellCoords(int newRow, int newCol, bool isIndx)
    {
        if (isIndx)
        {
            this.rowIndx = newRow;
            this.colIndx = newCol;
            SetNumFromIndxs(newRow, newCol);
        }
        else
        {
            this.rowNum = newRow;
            this.colNum = newCol;
            SetIndxFromNumbers(newRow, newCol);
        }
    }
    private void SetIndxFromNumbers(int rowNum, int colNum)
    {
        this.rowIndx = rowNum - 1;
        this.colIndx = colNum - 1;
    }
    public void SetNewNumbers(int newRowNum, int newColNum)
    {
        this.rowNum = newRowNum;
        this.colNum = newColNum;
        SetIndxFromNumbers(newRowNum, newColNum);
    }
...
}
```

Zdrojový kód 8: Část třídy pro dvoje souřadnice buňky

3.2.4 Indexování

V programu se pracuje se dvěma druhy indexování řádků a sloupců dat. Jazyk C# indexuje své více položkové typy od 0, avšak pletací stroj a uživatelské zobrazení pracuje s číslováním od 1. V pozdějším stádiu programování docházelo v tomto směru k častým orientačním problémům a tak za tímto účelem vznikla pomocná třída se jménem *TCellCoords*. Ta se využívá především v místech, kde se předávají informace o konkrétní buňce (např. změna jejího statusu), jelikož z grafického rozhraní často přichází číslování uživatelské. Tato třída obsahuje informace o souřadnicích buňky v obou formách. Označení *rowNum, colNum* je pro číslování od 1 a označení *rowIndx, colIndx* pro indexování v datech od 0. Tyto hodnoty se při vzniku proměnné tohoto typu vždy nastaví najednou. Dle potřeb vzniklo několik metod pro závislé nastavování hodnot. Část definice této třídy vidíte ve Zdrojovém kódu 8.

3.3 Návrhová tabulka

Pro zobrazování a navrhování dat slouží návrhová tabulka, která je vykreslována a obsluhována prostřednictvím třídy *TablePattern*. Jak již bylo řečeno, toto řešení bylo zvoleno kvůli specifickému zadání. Tabulka totiž nikdy nemůže dosáhnout nekonečných nebo příliš velkých rozměrů. Počet sloupců je omezen počtem jehel na pletacím lůžku na 160 a maximální počet řádků je definován řídící jednotkou pletacího stroje na 1500. Buňky tabulky slouží pouze k zobrazování různých barev (vodičů), kterých může být maximálně 7. Konstruktory této třídy jsou opět dva pro stejné případy – jeden pro výchozí čistou tabulku, druhý pro otevírání souboru.

Nejdůležitější objekty držené a vlastněné touto třídou jsou data a měřítka. Umístěny jsou zde, jelikož vykreslování tabulky přímo závisí na datech, bez kterých by neexistovala. Okrajová měřítka je třeba nastavovat podle aktuálních vlastností tabulky, proto je jejich hlavní správa umístěna zde, i když jejich vykreslování je spravováno z jiného místa a to díky tomu, že jsou umístěna v samostatné komponentě typu *pictureBox*. Více v kapitole Měřítka.

3.3.1 Převodní systém

Z důvodu častého výskytu převádění hodnot mezi barvami tabulky a statusu dat, vzniklo několik slovníků, které tuto práci zjednodušují. Pomocí definovaného výčtu *TColors* jsou navíc hodnoty srozumitelně pojmenované. Slovníky, které nastavují barvu

na základě jména barvy/vodiče normálních buněk a buněk výběru se nazývají *colorTable* a *colorTableSelect*. Barvy ve slovníku *colorTableSelect* jsou stálé a není možné je měnit. K zobrazení výběru na prázdných plochách byla zvolena *Color.Aquamarine* a pro výplňové barvy pouze 3 různé stupně šedi, z důvodu barevné estetičnosti. V prvním konceptu byly výběrové barvy vodičů vypočítávány jako inverzní, ale po odzkoušení jejich barevnosti od toho bylo odstoupeno. Naopak barvy v prvním slovníku se mění na základě nastavení uživatele.

Dalším slovníkem je *statusFromColorName*, který umožňuje získání statusu (znaku) příslušného k nějaké barvě. Toho je využíváno především při akci kreslení, kde se pomocí právě zvoleného názvu barvy/vodiče přímo nastavuje status buňky uvnitř dat. Tento slovník existuje i pro přesně opačný převod *colorNameFromStatus*, který se nejvíce využívá při zobrazování dat do grafické tabulky. V ukázce ve Zdrojovém kódu 9 je vidět výše zmíněný výčet a definice všech slovníků. Může se zdát, že množství slovníků je zbytečné a že je možné převádět znaky rovnou na barvy. Nicméně, zejména pro lepší orientaci a pro použití v různých částech programu, byl zvolen tento způsob převodu.

```
public enum TColors
        {
            Clear,
            Color1,
            Color1H,
            Color2,
            Color2H,
            Color3,
            Color3H
        }
TColors colorInUse;
public static Dictionary<TColors, SolidBrush> colorTable;
Dictionary<TColors, SolidBrush> colorTableSelect;
Dictionary<TColors, char> statusFromColorName;
public static Dictionary<char, TablePattern.TColors> colorNameFromStatus;
```

Zdrojový kód 9: Výčet názvů barev a převodní slovníky

3.3.2 Výběr oblasti

Pro umožnění vybírání oblasti myší bylo nutné vytvořit proměnnou, která bude kopírovat rozměry tabulky dat a umožní jednoduché nastavování výběru nad buňkou, nezávisle na jejím obsahu. Proto vznikl *List<bool[]> selectTable*, kde hodnota *true* nad „buňkou“ znázorňuje výběr oblasti. Vyplňování této tabulky probíhá pomocí určené

počáteční a konečné buňky, ty se však mění v závislosti na tom, jakým směrem je tažena myš. Toto správné nastavování obstarává metoda *SetContinousSelect(int mouseX, int mouseY)*, která vždy nejdříve vymaže předchozí oblast a poté nastaví novou aktuální. Vyplňování hodnot *true* probíhá v metodě *FillSelect(int startRow, int startCol, int lastRow, int lastCol)* velice jednoduchým způsobem pomocí *for* cyklů.

S již vytvořeným výběrem je možné provést několik akcí – vyplnit, zrušit výplň, kopírovat a vložit. První dvě akce jsou zprostředkovány pomocí stejné metody *SetSelectStatus(bool erase)*, kde se pouze podle potřeby změní parametr *erase*. Kopírování je prováděno na jiných úrovních, avšak pro získání definice kopírované oblasti jsou použity funkce *GetStartingSelectCell* a *GetEndingSelectCell*.

3.3.3 Vykreslování tabulky

Existují dva případy, pro které je třeba vykreslit celou grafickou podobu tabulky, pro editor vzoru a pro export obrázku. Pro obě tyto situace jsou vytvořeny rozdílné metody a to z důvodu potřeby rozdílného vzhledu zobrazení. Pro exportovaný obrázek je vytvořena přehlednější mřížka a také jsou nastaveny větší rozměry všech prvků (buňky, linie), z důvodu lepšího výstupu na tiskárně (jednobodové linie mohou být nesprávnou komprimací snáze ztraceny).

```
public void OnMouseMove(int mouseX, int mouseY)
{
    if (IsChangeFromLast(mouseX,mouseY)) //kontroluje souradnice nad bunkou
    {
        DataPattern.TCellCoords cell = GetCellIndex(mouseX,mouseY);
        if (cell.rowIndx > -1 && cell.colIndx > -1
         && cell.colIndx < data.GetColumnsCount()
         && cell.rowIndx < data.GetRowsCount())
            data.ChangeStatus(cell.rowIndx, cell.colIndx,
                              statusFromColorName[colorInUse]);
    }
}
```

Zdrojový kód 10: Metoda pro vyplňování jednotlivých buněk

Při všech akcích myši, které mění obsah buněk, je volána stejná metoda, která vyplňování a změnu statusu obstarává. Při přepínání akcí myši je aktualizována proměnná *colorInUse*, díky tomu je pomocí slovníku vybrán a nastaven správný status. Aby nastavování neprobíhalo několikrát po sobě (při každém pohybu myši nad stejnou buňkou), jsou vždy při prvním nastavení souřadnice uloženy a porovnávány s novými.

Teprve pokud se liší, proběhne nové zapamatování a nová změna statusu v datech. Tuto důležitou metodu je možné vidět ve Zdrojovém kódu 10.

```
public void DrawTable(Graphics g, int startRow, int lastRow, int startCol,
                      int lastCol)
{
   Brush brush;
   gridLine = Pens.LightGray;

   //cykly pro sloupce a radky
   for (int row = startRow; row < lastRow; row++)
   {
      for (int col = startCol; col < lastCol; col++)
      {
         TColors colorToDraw = colorNameFromStatus[data.GetStatus(row, col)];
         if (selectTable[row][col])
             brush = colorTableSelect[colorToDraw];
         else
             brush = colorTable[colorToDraw];

         //vykresleni ctvercu a mrizky
         g.FillRectangle(brush, col * myCellSize , row * myCellSize,
                         myCellSize, myCellSize);
         g.DrawRectangle(gridLine, col * myCellSize , row * myCellSize,
                         myCellSize, myCellSize);
      }
   }
   //zobrazeni opakovani
   int[] repetInfo = data.FindRepetition();
   if (Properties.Settings.Default.Rep_Hidden && repetInfo != null)
   {
      Pen gridLine2 = new Pen(Brushes.LightGreen, 2);
      int start = repetInfo[0];
      int end = repetInfo[1];
      g.DrawLine(gridLine2,
               new Point(0, (start - 1) * myCellSize + myCellSize),
               new Point((data.GetColumnsCount() - 1)*myCellSize + myCellSize,
               (start - 1) * myCellSize + myCellSize));
      g.DrawLine(gridLine2,
               new Point(0, end * myCellSize + myCellSize),
               new Point((data.GetColumnsCount() - 1)*myCellSize + myCellSize,
               end * myCellSize + myCellSize));
   }
}
```

Zdrojový kód 11: Metoda pro vykreslování návrhové tabulky

Vykreslování probíhá procházením každé položky obsažené v datech a podle jejího obsahu je volena barevná výplň čtverce. Ovšem kontrolována je i hodnota výběru, která má při vykreslování přednost. Velikost vykreslovaných čtverců ovládá uživatel pomocí volby Lupa. Linka čtverců je jednobodová s barvou *Pens.LightGray*, protože díky tomu jsou vidět okraje buněk s výplní černou i bílou. Nakonec po vykreslení celé tabulky se kontroluje výskyt opakování, které pokud je v režimu „skryté“, tak se navíc zobrazují

jeho hranice pomocí zelených linek. Celou metodu pro vykreslování návrhové tabulky je možné vidět ve Zdrojovém kódu 11.

V dalším kroku vykreslování je použita metoda určená pro zobrazení „informační buňky“, ta je určena posledním místem, kde bylo uvolněno tlačítko myši. Tato buňka musí být nějak zvýrazněna a to z důvodu jasného určení místa, pro které se zobrazují veškeré informace těsně pod návrhovou tabulkou, dále slouží pro mnohá nastavení. K tomu slouží metoda *DrawCell(Graphics g, int row, int col)*, která funguje na stejném principu jako předchozí metoda, avšak její linka ohraničení má tloušťku 3 bodů a barvu *Brushes.Magenta*.

3.3.4 Manipulace s řádky

Protože je nutné promítat změnu počtu řádků i do proměnné *selectTable*, musely být tyto metody posunuty i do úrovně této třídy a ve všech velikost správně aktualizovat. S tím souvisí i aktualizace levého postranního měřítka tabulky. Ve Zdrojovém kódu 12 je ukázána jedna z těchto metod.

```
public void DeleteRows(int startIndx, int count)
{
    DeleteSelect();
    data.DeleteRows(startIndx, count);
    if ((startIndx + count) <= data.GetRowsCount())
    {
        for (int row = startIndx; row < startIndx + count; row++)
        {
            selectTable.RemoveRange(startIndx-1,count);
        }
    }
    rulerLeft.UpdateNumbRows(data.GetRowsCount(), GetTableHight());
}
```

Zdrojový kód 12: Metoda pro odstranění řádků na úrovni třídy TablePattern

3.3.5 Informační buňka

Nedílnou součástí programu je zobrazování informací o právě vybrané buňce. Získávání těchto údajů bylo umístěno do této třídy, protože je prostředníkem mezi souborovými daty a grafickou úrovní programu. Podle souřadnic myši je třeba zprostředkovat informace o řádku umístěné ve třídě *DataPattern* v proměnné *prefixDataPattern* a k nim navíc přidat souřadnici sloupce a také status výplně buňky (viz Zdrojový kód 13). Konstrukce informací byla zvolena tímto způsobem, protože většina z nich se již vyskytuje ve zmíněných prefixových datech. Proto byla maximálně využita

a pouze rozšířena o další dvě položky. Další manipulace se odehrává ve vlastní komponentě *PictureBoxPattern*.

```
public DataPattern.TPrefixData GetInfoCell(int mouseX, int mouseY,
                                           out string status,out int cellNum)
{
    DataPattern.TCellCoords cell = GetCellIndex(mouseX, mouseY);
    if (cell.colIndx < 0 || cell.rowIndx < 0)
    {
        status = "0";
        cellNum = 0;
        return new DataPattern.TPrefixData();
    }
    cellNum = cell.colNum;
    DataPattern.TPrefixData info = data.GetInfoRow(cell.rowIndx);
    status = data.GetStatus(cell.rowIndx, cell.colIndx).ToString();
    return info;
}
```

Zdrojový kód 13: Funkce pro složení informací o zvolné buňce

3.4 Měřítka tabulky

Měřítka jsou vytvářena pomocí třídy *TableRuler*, která má jeden konstruktor, prostřednictvím kterého se nastaví výchozí rozměry a typ pravítka. Existují dvě a vykreslují se podle typu určeného při vytváření instance. Díky tomu je možná jejich samostatná aktualizace. Protože se může měnit pouze počet řádků v tabulce, je zde vytvořena metoda pouze pro tuto změnu proměnných.

Vzhled měřítek je tvořen okrajovými buňkami a pravítkem složeným z úseček a čísel. Čísla jsou psána v násobcích 10 spolu s nejvýraznější úsečkou, mezi nimi v násobcích 5 jsou úsečky slabší a zbylá čísla jsou označena pouze velice krátkými úsečkami odsazeními od kraje.

3.4.1 Vykreslování měřítka

Měřítka je třeba zobrazovat opět ve dvou případech a tím je zobrazení v Editoru vzoru a dále při exportu obrázku. Pro export je vytvořeno lehce odlišné zobrazení a to především čísel, která jsou vykreslena podstatně větším a tučným fontem. Vykreslení určené pro Editor vzoru provádí metoda *DrawRuler(Graphics graph)*, která by se dala rozdělit na dvě oblasti – vykreslování okrajových buněk a dále vykreslení čísel s úsečkami. Zajímavější a také náročnější na vývoj je oblast druhá. Vykreslování probíhá po jednotlivých buňkách, kdy je nejprve nutné určit přesnou polohu úsečky. Ta má být

vykreslena na středu buňky, stačí tedy odečíst polovinu velikosti jedné buňky. Pozice čísel je nutné vypočítat pomocí jejich opravdové šířky a počáteční bod posunout o polovinu této hodnoty. Metoda, která provádí vykreslování samotných čísel, je uvedena ve Zdrojovém kódu 14.

```
private void DrawMyNumbers(string which, Graphics gr, int number,
                           int centerPosition)
{
    Font myFont = new Font("Arial", 6, FontStyle.Bold);
    SizeF textSize = gr.MeasureString(number.ToString(), myFont);
    int transC = (int)(textSize.Width / 2);

    if (which == "TOP")
    {
        gr.DrawString(number.ToString(), myFont, Brushes.Green,
                      new Point(centerPosition - transC, 2));
    }
    else if (which == "LEFT")
    {
        gr.DrawString(number.ToString(), myFont, Brushes.Blue,
                      new Point(centerPosition - transC, 2));
    }
}
```

Zdrojový kód 14: Metoda pro vykreslení čísel měřítka

Složitost, která byla při vykreslování měřítek řešena, byla především transformace plátna pro měřítko u levé strany tabulky. Transformace byla nutná kvůli otočení čísel, k tomu bylo přistoupeno v zájmu šetření místa – kdyby se čísla neotáčela, bylo by nutné je vypisovat do řádku a při vyšších hodnotách (nad 1000) by šířka měřítka značně narostla. Pro provedení správného způsobu otočení plátna bylo nutné vypočítat správné umístění jednotlivých úseček, protože nejvyšší hodnota pozice se momentálně nacházela v místě, kde měřítko začíná od hodnoty 1. Ve Zdrojovém kódu 15 je uvedena část z metody *DrawRuler(Graphics graph)*, kde je tento problém řešen.

3.5 Kroky zpět a vpřed

Třída, pomocí které jsou realizovány funkce kroku zpět a vpřed, má název *MementoData*. K realizování těchto funkcí bylo potřeba datové struktury podobné zásobníku s konečnou délkou, který po přidání prvku na vrchol ve stavu, kdy je paměť plná, vymaže prvek na dně paměti, tím uvolní místo a vloží tam na vrchol prvek další. Pro to byla použita třída *LimitedSizeStack<T>* [6]. Jak napovídá název třídy, k obnovování dat je použit návrhový vzor Memento, který ukládá kompletní stav dat.

```
graph.TranslateTransform(0, tabHeight);
graph.RotateTransform(-90);
...
int num = tabCellsNum;
for (int col = 1; col <= tabCellsNum; col++, num--)
{
    position = (col + 1) * cellSize - (cellSize / 2);

    if ((num % 5) == 0)
    {
        if ((num % 10) == 0)
        {
            pen = new Pen(Brushes.Black, 2);
            graph.DrawLine(pen,new Point(position,15),new Point(position,30));
            DrawMyNumbers("LEFT", graph, num, position);
            continue;
        }
        pen = new Pen(Brushes.Black, 1);
        graph.DrawLine(pen, new Point(position, 20), new Point(position, 30));
        continue;
    }
    pen = new Pen(Brushes.Black, 1);
    graph.DrawLine(pen, new Point(position, 25), new Point(position, 26));
}
```

Zdrojový kód 15: Úsek s řešením vykreslování levého měřítka tabulky

Objekty zapamatování jsou však pouze dva – *List<char[]> dataPattern* a *List<DataPattern.TPrefixData> prefixDataPattern*. Ve třídě *MementoData* tedy existují celkem 4 limitované zásobníky, 2 pro akci zpět a 2 pro akci vpřed. Jejich kapacita je nastavena na 10 úkonů.

Kdykoli proběhne změna v návrhové tabulce, je nutné volat metodu *SaveMyData*, která si kompletně uloží jejich současný stav do zásobníku typu „undo“. Protože se v obou případech jedná o listy obsahující položky složených datových typů, je při jejich ukládání nutné vytvořit novou instanci i vnitřních položek. K tomu byly vytvořeny pomocné funkce *CreateNewDataPattern* a *CreateNewDataPrefix*. Pro správné fungování tohoto systému bylo třeba třídu doplnit o několik pomocných proměnných typu *bool*. V prvním případě je nutné v určitých situacích nejprve přehodit vrchol zásobníku na druhý a až poté vykonat jeho akci. Jsou to situace, které by se daly nazvat „změna směru“, tedy všechny první čtení z daného zásobníku (např. první Krok zpět po kreslících akcích). Ke značení těchto situací slouží proměnné *doUndoPop* a *doRedoPop*. Druhá dvojice proměnných nesoucí název *canUndo* a *canRedo* slouží k nastavování dostupnosti tlačítek s touto akcí. Pro tuto indikaci totiž nestačí pouze počet obsažených položek v příslušných zásobnících. Např. při otevření nového souboru je potřeba si uložit jeho stav při nahrávání, zásobník

„undo“ obsahuje jednu položku, ale ta zatím neslouží k vykonání akce (souvisí s předchozími proměnnými). Ve Zdrojovém kódu 16 je vidět způsob ukládání dat do paměti zásobníků. Nakonec je pro jistotu konzistence dat vytvořena proměnná *undoRedoLock*, která pokud je hodnoty *true*, není možné zároveň data ukládat.

```
public void SaveMyData(List<char[]> data,
                       List<DataPattern.TPrefixData> prefixData)
{
    while (undoRedoLock)
        System.Threading.Thread.Sleep(100);

    if (doRedoPop)
    {
        dataPatternUndo.Push(CreateNewDataPattern(dataPatternRedo.Pop()));
        dataPrefixUndo.Push(CreateNewDataPrefix(dataPrefixRedo.Pop()));
        doRedoPop = false;
    }

    this.dataPatternRedo.Clear();
    this.dataPrefixRedo.Clear();
    this.dataPatternUndo.Push(CreateNewDataPattern(data));
    this.dataPrefixUndo.Push(CreateNewDataPrefix(prefixData));
    canRedo = false;
    if (dataPatternUndo.Count > 1)
        canUndo = true;
    doUndoPop = true;
}
```

Zdrojový kód 16: Ukládání dat do paměti zásobníků ve třídě MementoData

Pro akce zpět a vpřed slouží funkce *UndoDataChange* a *RedoDataChange*. Obě fungují téměř totožně, jen používají příslušné zásobníky a proměnné. Na začátku a na konci se nastavuje proměnná *undoRedoLock*, dále, pokud je potřeba, je nejprve přemístěn vrchol zásobníku na druhý. Pak se již pracuje se samotnými daty, která jsou nahrána do výchozí proměnné, tím i odstraněna z vrcholu a přesunuta na zásobník druhý. Nakonec se ještě správně nastaví proměnné, o kterých je hovořeno výše.

3.6 Komponenta návrhové tabulky

Bylo rozhodnuto, že tato komponenta bude zastřešovat většinu vzájemné komunikace a propojení všech předcházejících objektů. Proto je složena z několika dalších komponent, které spolu tvoří návrhové prostředí. Tato třída má název *PictureBoxPattern*. Je možné zde najít nejen editační tabulku s měřítkem, ale i postranní informační tabulku. Pomocí několika komponent panelů je vytvořen potřebný vzhled, který je třeba upravovat podle nastavení velikosti buněk. Všechny objekty, které je třeba

vykreslovat, jsou umístěné do svých vlastních pictureBoxů (tabulka a dvě měřítka). Informační tabulka je vypisována do komponenty dataGridView.

3.6.1 Tabulka

Vykreslování tabulky probíhá na základě události se jménem *pictureBox_Paint* (Zdrojový kód 17), která se volá pomocí *Refresh()* na konci všech metod a funkcí, jakkoliv manipulujících s tabulkou – tím je zajištěno okamžité zobrazení změny. Tato metoda však nespravuje pouze vykreslování dat, nýbrž také aktualizaci všech závisejících komponent a informací. Tím je hlavně scrollování v tabulce, které ovlivňuje vykreslovanou oblast, posunutí meřítek a posunutí (změna vykreslení) postranní tabulky. Dále je třeba získávat aktuální informace o prefixových datech a vybrané (informační) buňce. To vše je třeba vykonávat synchronně se změnami v tabulce, jinak by reakce komponent nebyly dostatečně přesné.

```
private void pictureBox_Paint(object sender, PaintEventArgs e)
{
    prefixData = table.data.GetPrefixData();
    RecalcPanels(); //pro okamzite zobrazeni zmeny velikosti

    startRowToView = GetStartRow(panel1.VerticalScroll.Value); //viditelna oblast
    lastRowToView = GetLastRow(panel1.VerticalScroll.Value, panel1.Height);
    startColToView = GetStartCol(panel1.HorizontalScroll.Value);
    lastColToView = GetLastCol(panel1.HorizontalScroll.Value, panel1.Width);

    PerformInfoLine(mousePos.X, mousePos.Y);

    table.DrawTable(e.Graphics, startRowToView, lastRowToView, startColToView,
                    lastColToView);
    if (Properties.Settings.Default.Rep_Hidden && infoCell.GetRowIndex() >
       (table.data.GetEndIndxOfRepetition()+table.data.GetLengthOfHiddenRepetition()))
        table.DrawCell(e.Graphics, infoCell.GetRowIndex() - 1
                        - table.data.GetLengthOfHiddenRepetition(),
                        infoCell.GetCellIndex() - 1);
    else
        table.DrawCell(e.Graphics,infoCell.GetRowIndex()-1,infoCell.GetCellIndex()-1);

    PerformDatagridScroll();

    panelRulerTop.HorizontalScroll.Value = panel1.HorizontalScroll.Value;
    panelRulerLeft.VerticalScroll.Value = panel1.VerticalScroll.Value;
}
```

Zdrojový kód 17: Akce vykonávané v události pictureBox_Paint

Protože vykreslování celé tabulky by bylo zbytečné, vznikly jednoduché metody *GetStartRow*, *GetLasttRow*, *GetStartCol* a *GetLastCol* pro výpočet zobrazované oblasti tabulky. Ty počítají s aktuálními hodnotami scrollBarů panelu, ve kterém je umístěn

hlavní pictureBox a samotnou velikostí panelu. Tyto hodnoty se pak jednoduše předají v podobě parametrů a vykresluje se pouze tato oblast.

Již zmíněná aktualizace postranní tabulky probíhá v metodě *PerformDatagridScroll*, která nejprve vymaže všechny řádky dataGridu a pak projde vybraný počet řádků prefixových dat podle zobrazení a načte některé jejich informace do nových buněk. Informace o použitém vodiči a o výměně vodiče mezipohybem se zobrazují pomocí barvy pozadí buněk. Poté je nutné nastavit odpovídající výšku řádků a velikost písma (existují pouze dvě možnosti).

Během vývoje programu byl řešen nezvyklý problém s rychlostí vykreslování změn tabulky. Dle očekávání se rychlost mohla snižovat při zobrazení menší velikosti buněk, což zapříčiní větší oblast pole dat pro procházení. Nicméně tato rychlost klesala i při zvětšování buněk. Závěr pozorování byl, že všechny velikosti buněk, kromě té výchozí (100%), razantně zpomalují vykreslování. Po dlouhém zkoumání a snažení všechny akce minimalizovat, bylo objeveno jádro problému v nevhodném nastavování velikosti řádků komponenty *dataGridView* (postranní tabulka). V následující Tab. 1 je pro zajímavost uveden nevhodný způsob kódu pro toto nastavování i s naměřenými časy trvání (výchozí velikost řádků je u počtu 36). Pod tím je uvedeno správné řešení a jeho délka trvání (pro jakýkoliv počet řádků).

Měřený kód	**Počet řádků**	**Trvání [s]**

`foreach (DataGridViewRow row in dataGridView1.Rows)`
`{`
`  row.Height = lastSizeCell;`
`}`
	29	00.0971764

	32	00.1137961
	36	00.0000043
	47	00.2419116
`dataGridView1.RowTemplate.Height = lastSizeCell;`	X	00.0000003

Tab. 1: Měření na problémovém úseku kódu

3.6.2 Kreslení v tabulce

Kreslení do tabulky probíhá pomocí tlačítek myši a jejího pohybu. Proto jsou vytvořeny metody pro tři události *pictureBox_MouseDown*, *pictureBox_MouseMove*

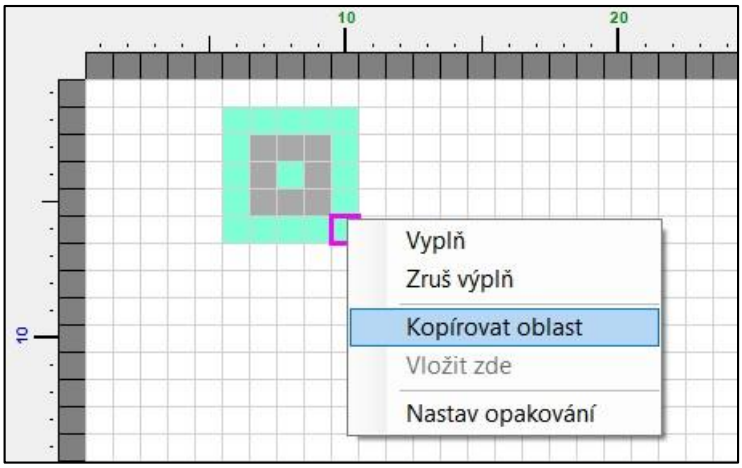
(Zdrojový kód 18) a *pictureBox_MouseUp*. Všechny mají společné příkazy pro zaznamenání současné pozice myši a pro refresh tabulky. Pro akce je třeba rozeznávat pravé tlačítko pro tvoření výběru a levé tlačítko pro kreslení. První metoda se liší především příkazy pro obnovou/zrušení některých dat, druhá hraje největší úlohu při kreslení/vybírání oblasti, třetí ukládá změny do paměti kroků.

```
private void pictureBox_MouseMove(object sender, MouseEventArgs e)
{
    if (MouseButtons == MouseButtons.Left
        && e.Y >= ((startRowToView) * lastSizeCell)
        && e.Y < ((lastRowToView) * lastSizeCell))
    {
        if (paintAllow)
        {
            table.OnMouseMove(e.X, e.Y);
            PerformDataGridColors(e.Y);
        }
        mousePos = e.Location;
        pictureBox.Refresh();
    }
    else if (MouseButtons == MouseButtons.Right
             && e.Y >= ((startRowToView) * lastSizeCell)
             && e.Y < ((lastRowToView) * lastSizeCell))
    {
        table.SetContinousSelect(e.X, e.Y);
        mousePos = e.Location;
        pictureBox.Refresh();
    }
}
```

Zdrojový kód 18: Kreslení při pohybu myši

3.6.3 Nabídka výběru

Aby bylo možné výběr oblasti jednoduše a rychle používat, bylo z tohoto důvodu vytvořeno kontextové menu, které se zobrazí po uvolnění pravého tlačítka myši nad hlavním pictureBoxem. Nabídka obsahuje 5 položek – Vyplň, Zruš výplň, Kopírovat, Vložit zde, Nastav opakování (viz Obr. 10). Tyto položky je tedy možné vykonat pouze po vytvoření výběru, jinde nejsou dostupné. Detailní postupy těchto akcí jsou vysvětleny v předchozí kapitole Data vzoru, zde se metody pouze volají a téměř vždy jsou následovány dvojící příkazů pro refresh tabulky a pro uložení do paměti. Položka Vložit zde je dostupná od chvíle, kdy je nějaká oblast ke kopírování zvolena.



Obr. 10: Nabídka výběru

3.7 Komponenta náhledu obrázku

Aby byl program přehlednější, vznikla také komponenta s názvem *ViewFinalPicture*. Ta se skládá pouze z panelu komponenty a pictureBoxu pro vykreslení obrázku. Pro vykreslení obrázku jsou nutná data, která odpovídají aktuálnímu návrhu v tabulce. Z důvodu jejich ochrany je zde vytvořena vlastní proměnná *dataView*, do které se data vždy nahrají a rovnou upraví do náhledové podoby pomocí metody *CreateViewData*. Hlavním úkolem náhledu je složení těch sousedních řádků, které jsou pleteny různými vodiči. Ostatní řádky jsou ponechány v originální podobě. Proto

```
private void MakeOneRow(int row,List<char[]> dataFrom)
{
    char[] newRow = new char[dataFrom[0].Length];
    char[] newRow0 = new char[dataFrom[0].Length];

    for (int col = 0 ; col < dataFrom[0].Length; col++)
    {
        if (finishBeforeRow) //pokud se v předchozim pripade nevykreslil radek
            newRow0[col] = dataFrom[row - 1][col];

        if (dataFrom[row][col] != '0')
            newRow[col] = dataFrom[row][col];
        else
            newRow[col] = dataFrom[row+1][col];
    }
    if (finishBeforeRow)
        dataView.Add(newRow0);
    dataView.Add(newRow);
}
```

Zdrojový kód 19: Skládání řádků pro náhled

prochází data po dvou, avšak začátek je posunut o 1 (kontrolují se sudé a liché řádky z uživatelského pohledu). Kvůli tomu je potřeba extra zaznamenávat „vynechání“ předchozího řádku a tedy, pokud k tomu dojde, musí se doplnit. Pro skládání řádků je volána metoda *MakeOneRow* (Zdrojový kód 19), pro normální řádky *MakeOrdinaryRow*.

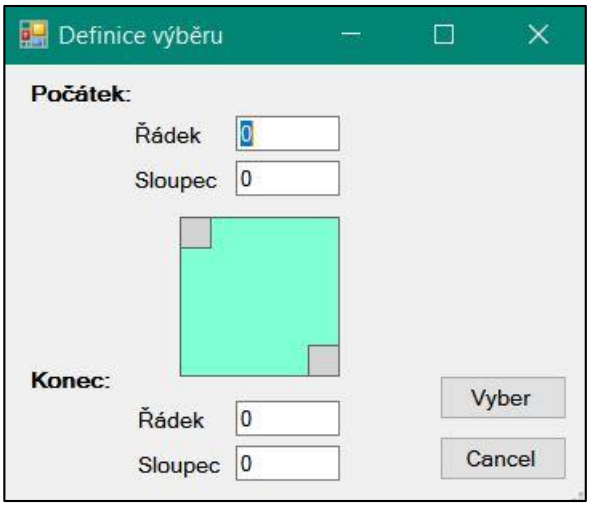
Vykreslování probíhá obdobně jako u hlavní návrhové tabulky, jen je zobrazení otočené o 90° vlevo a výsledný obrázek je navíc zobrazen od levého spodního rohu. Kvůli otočení bylo třeba vykreslování upravit (nepoužívá se zde translace a rotace plátna). Velikost buněk je menší a pevně nastavená na 10 bodů. Ve Zdrojovém kódu 20 je možné vidět úpravu pořadí vykreslování buněk.

```
public void DrawTable(Graphics g)
{
    Brush brush;
    Pen gridLine  = Pens.LightGray;

    //cykly pro sloupce a radky
    for (int row = 0; row < dataView.Count; row++)
    {
        int backCol = 0;
        for (int col = dataView[0].Length-1; col >= 0 ; col--, backCol++)
        {
            TablePattern.TColors colorToDraw =
                                lineStatusToColorName[dataView[row][col]];
            brush = colorTable[colorToDraw];

            //vykresleni ctvercu a mrizky
            g.FillRectangle(brush, row * cellSize, backCol * cellSize,
                            cellSize, cellSize);
            g.DrawRectangle(gridLine, row * cellSize, backCol * cellSize,
                            cellSize, cellSize);
        }
    }
}
```

Zdrojový kód 20: Vykreslování náhledu obrázku

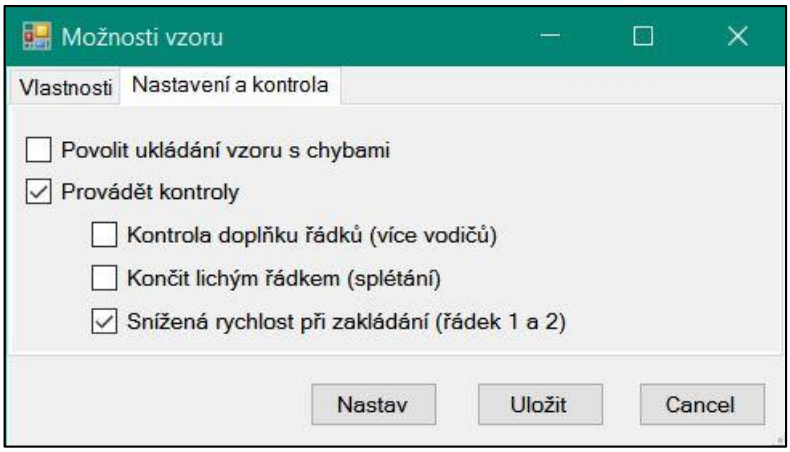
3.8 Pomocné formuláře

Součástí programu jsou 4 vedlejší formuláře určené především pro různá nastavení, či možnosti hodnot. Se všemi je pracováno v hlavním formuláři v módu dialogu, aby bylo možné na výsledek některých ihned reagovat. Všechny formuláře jsou opatřeny jednoduchou kontrolou, která upozorňuje na špatně vyplněné hodnoty v polích (např. nejedná-li se o čísla).



Obr. 11: Formulář pro definici výběru

Jediný *FormSpecifySelect* (Obr. 11) je určený k rozdílné akci, než formuláře ostatní. Pomocí vyplnění hodnot jednotlivých položek pro souřadnice počáteční a koncové buňky je možné přesně definovat výběr oblasti. Po stisknutí tlačítka *Vyber* se nastaví proměnné *startC* a *endC*, ke kterým je možný veřejný přístup. Toto tlačítko také nastaví *DialogResult* na hodnotu *OK* a pak je možné v hlavním formuláři provést příslušné akce s nastavenými buňkami. Po vykreslení výběru se zobrazí i výběrová nabídka a tak je možné pohodlně vybrat potřebnou akci.

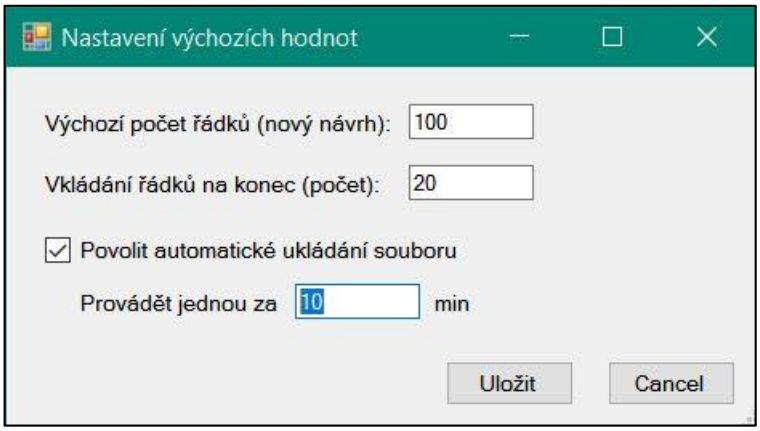


Obr. 12: Formulář pro možnosti vzoru - nabídka pro kontroly

FormPatternSettings (Obr. 12) umožňuje několik nastavení vzoru týkajících se hlavičky souboru a dále kontrol, které se budou provádět při jakékoli kompilaci či exportu souboru. Všechny položky jsou ukládány do souboru nastavení projektu, aby byly volně přístupné ze všech tříd a také aby bylo nastavení zachováno pro daného uživatele i při novém otevření aplikace.

Dalším pomocným formulářem je *FormDirectoriesSettings*, který v podstatě není pro uživatele zásadně důležitý, nicméně díky tomuto nastavení je možné rychleji pracovat se soubory. Uložení těchto adresářových cest totiž umožní otevírat ukládací/otevírací dialogy na určené adrese, není potřeba pak při každé manipulaci znovu a znovu hledat správné umístění složky. Tyto položky jsou opět umístěné v souboru projektu.

Posledním je *FormValuesSettings* (Obr. 13), který spíše přidává pár extra funkcí/nastavení do aplikace. První dvě položky jsou pouze pro nastavení výchozích hodnot – počet řádků nové tabulky a počet přidávaných řádků na konec. Třetí položka však umožňuje novou funkci automatického ukládání. Pokud je možnost zaškrtnuta, zpřístupní se políčko pro zadání minut. Lze zadat pouze celé minuty. Po uložení se ihned tato funkce spustí a o proběhlých uloženích je uživatel informován ve spodním informačním panelu aplikace. Před prvním použitím (na daném souboru) je však nutné provést nejprve manuální uložení, díky kterému se nastaví jméno souboru. Pak již ukládání probíhá samo. Pokud jej chce uživatel zrušit, musí to provést pomocí „odškrtnutí“ políčka a opětovného uložení.



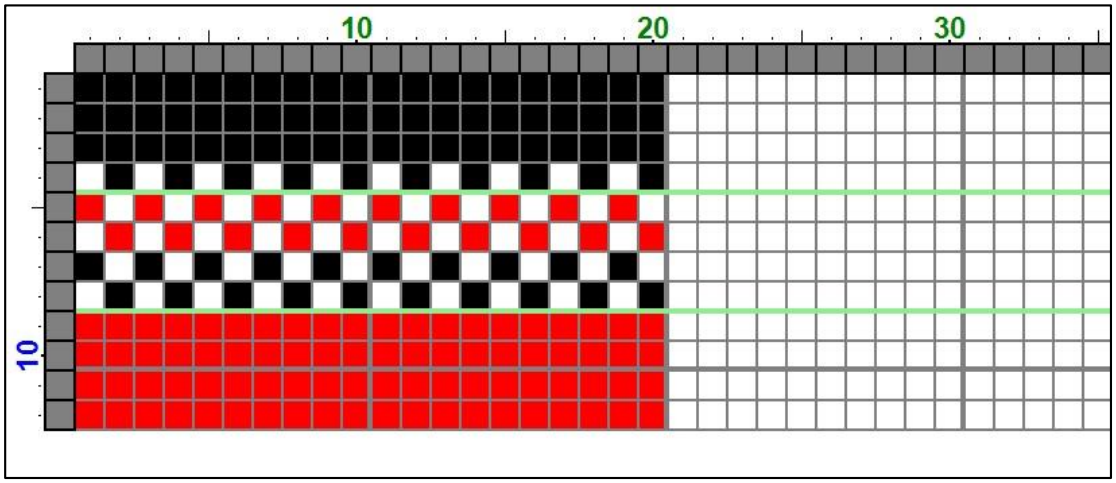
Obr. 13: Formulář pto nastavení výchozích hodnot

3.9 Hlavní formulář

Všechny funkce aplikace jsou soustředěny a spojeny v jednom hlavním formuláři s názvem *Form1*, který také tvoří grafické rozhraní pro uživatele. Vzhled je již popsán v kapitole Struktura vzhledu aplikace, zde jsou uvedeny bližší informace o fungování některých jejích částí. Ve většině tříd je možné se setkat s metodou *Init*, která všude zastává stejnou funkci. Tou je inicializace několika proměnných, vlastností a rozměrů komponent, které je třeba provádět nejen při vzniku třídy, ale například i při otevírání

nových souborů. Vše, co je možné univerzálně nastavovat ve všech situacích, se zde nachází.

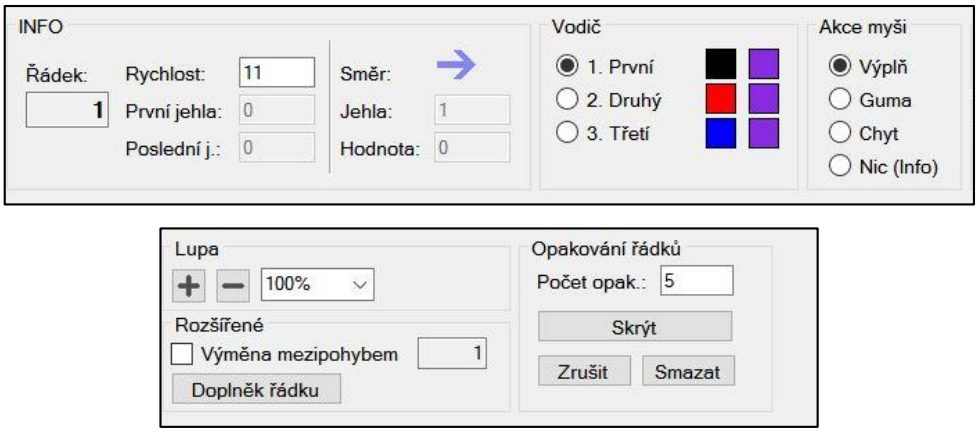
Otevírání a ukládání souborů probíhá na základě intuitivně zobrazených tlačítek. Pro jistotu uživatelů neztratit omylem neuložená data například při zavírání aplikace, byla vytvořena funkce *CheckSaveFile*, která se ve formě dialogu ujistí, že je opravdu možné pokračovat v akci (případně umožní uložení). Při všech událostech, které umožňují uložení, probíhají (dle volby uživatele) kontroly dat vzoru, o kterých je zobrazeno hlášení ve spodním informačním panelu. Export obrázku umožňuje uložit vzor pouze ve formátu JPG, protože není třeba např. průhlednosti a je podporován ve velkém množství editorů. Tato funkce je výhodná především při využití funkce opakování a při jeho skrytém režimu, protože zobrazení výsledného obrázku závisí na volbě tohoto režimu. Podle něj bude totiž vzor exportován a hranice opakování budou samozřejmě zobrazeny (viz Obr. 14). V této formě je výsledek nejlépe použitelný pro přikládání k dokumentacím.



Obr. 14: Výřez z exportovaného obrázku

Pro návrh vzoru v tabulce bylo nutné také připojit několik panelů pro nastavování akcí myši, zobrazování informací a dalších funkcí. Proto jsou panely umístěny ihned pod tabulkou, vidět jsou na Obr. 15 (pro lepší čitelnost jsou rozděleny na 2 obrázky). Jejich vzhled je inspirován z předchozí aplikace, avšak došlo k několika úpravám a sjednocením.

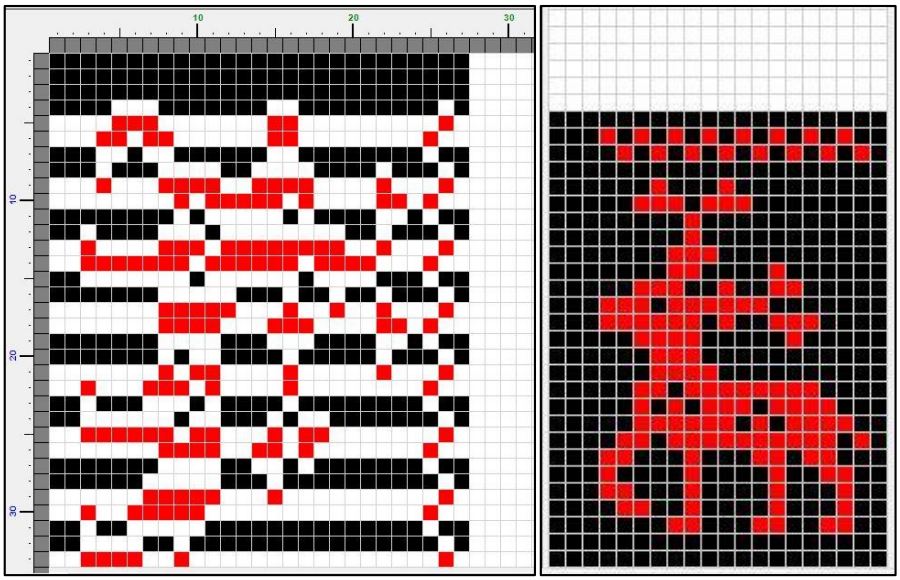
V oblasti *INFO* jsou zobrazeny všechny položky, které doplňují postranní informační tabulku a závisí na zvolené (zvýrazněné) buňce. Jedinou položku *Rychlost* je možné měnit pomocí potvrzení tlačítkem Enter. Oblasti *Vodič* a *Akce myši* spolu úzce souvisejí a týkají se kreslení v návrhové tabulce. v každé je možné mít zvolenou pouze



Obr. 15: Panely pro možnosti návrhu vzoru v tabulce

jednu položku. První oblast nastavuje vodič, se kterým je aktuálně pracováno. Jejich barvy (v celé tabulce) je možné měnit pomocí kliknutí na barevnou oblast za názvem. Akce myši vystihují svůj název, položky *Výplň* a *Chyt* jsou propojené s nastavením vodiče, zbylé dvě položky jsou použitelné pro jakoukoliv oblast v tabulce (neomezeně na vyplněném vodiči). Dalším panelem je *Lupa*, která manipuluje se zobrazenou velikostí buněk tabulky. Neovlivňuje zobrazení v exportovaném obrázku ani v náhledu obrázku. *Rozšířená* nastavení jsou dvě, první nastavuje pro daný řádek se zvýrazněnou buňkou (číslo za volbou) výměnu vodiče mezipohybem. Druhá umožňuje vytvoření následujícího automatického doplňku zvoleného řádku a vyplní ho právě zvoleným vodičem. Obě funkce jsou nejužitečnější při vzoru s více vodiči. Na konci je oblast, která ovládá možnosti *Opakování řádků*. První položku pro počet je nutné použít před jeho nastavením, další tlačítka se používají po jeho vytvoření.

V záložce *Náhled obrázku* jsou umístěny dva důležité objekty – tlačítko zobrazující náhled a plátno pro jeho zobrazení. Náhled je vytvářen pro aktuální stav dat v tabulce a to tedy včetně skrytého opakování. Jak již bylo uvedeno, tento náhled nevykonává žádné kontroly, k tomu slouží jiné oblasti. Slouží tedy především pro lepší vizuální představu o navrhovaném vzoru, na následujícím Obr. 16 je vidět jednoduchá ukázka malé části vzoru v návrhové tabulce a vedle něj v náhledu. Další záložka *Náhled kompilace* je téměř totožná z verzí v předchozí aplikaci. v horní části se zobrazují hlavičkové informace vzoru a některé je možné nastavit a uložit. Tlačítkem *Zkompiluj* se zobrazí náhled textového souboru.



Obr. 16: Obrázek soba v návrhové tabulce (vlevo) a v náhledu (vpravo)

Ve spodní části formuláře se nachází panel pro zobrazování událostí a výsledků kontrol. Zprávy jsou vypisovány pomocí jednoduché metody *ShowInfoMessage*, kterou můžete vidět ve Zdrojovém kódu 21. Na začátek každé události přidá informaci o aktuálním čase. Poslední komponenta zcela vespodu aplikace slouží k zobrazení

```
public void ShowInfoMessage(string message)
{
    string time = "[" + DateTime.Now.ToLongTimeString() + "] ";
    txbMainMessageInfo.AppendText(time +message +System.Environment.NewLine);
}
```

Zdrojový kód 21: Výpis zpráv do informačního panelu

doplňujících informací, jako je název aktuálního souboru, dále souřadnice buňky, nad kterou se právě nachází myš a nakonec přesné souřadnice počáteční a koncové buňky existujícího výběru oblasti. Protože výpočet souřadnic buněk probíhá na jiné úrovni programu, bylo nutné vytvořit třídu *GlobalMouseHandler*, získanou ze zdroje [4], která bude umožňovat zachycení pohybu myši nad jakoukoliv komponentou formuláře. Zároveň však bylo nutné identifikovat myš pouze nad návrhovou tabulkou, díky čemuž bylo možné zobrazovat informace o pozici. To je vyřešeno v metodě *mouseMoving_TheMouseMoved* (Zdrojový kód 22).

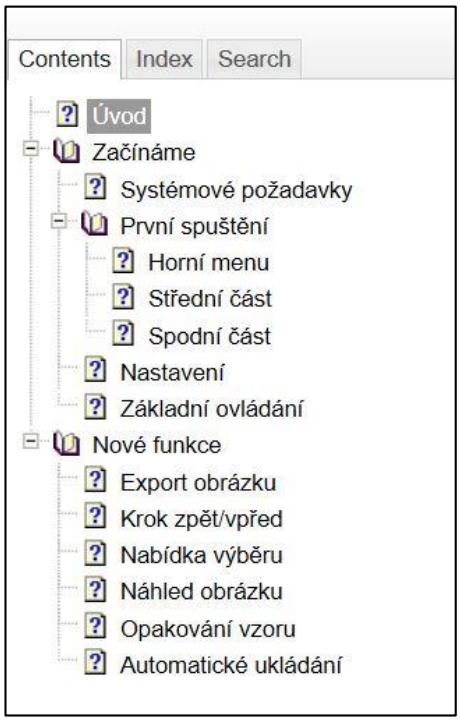
```
DataPattern.TCellCoords mouseCoords;
void mouseMoving_TheMouseMoved()
{
    Point curPosTable = pictureBoxPattern1.GetPictureBox().PointToClient(
                        System.Windows.Forms.Cursor.Position);
    Point curPosPanel = pictureBoxPattern1.GetPictureBoxPanel().PointToClient(
                        System.Windows.Forms.Cursor.Position);

    if (pictureBoxPattern1.GetPictureBoxPanel().Size.Height < curPosPanel.Y
     || pictureBoxPattern1.GetPictureBoxPanel().Size.Width < curPosPanel.X
     || 0 > curPosPanel.Y || 0 > curPosPanel.X) //pokud jsi mimo tabulku
    {
        tsCoordinates.Text = "[0,0]";
    }
    else
    {
        mouseCoords = pictureBoxPattern1.table.GetCellIndex(
                      curPosTable.X, curPosTable.Y);
        tsCoordinates.Text = String.Format("[{0},{1}]",
                             mouseCoords.rowNum, mouseCoords.colNum);
    }
...
}
```

Zdrojový kód 22: Část metody pro výpis souřadnic myši

4 Uživatelská nápověda

Jeden z důležitých požadavků na aplikaci byla také uživatelská přívětivost a možnost otevření nápovědy. Z těchto důvodů je v celé aplikaci umístěn velký počet nápovědních okének ve formě *ToolTipu,* která jsou umístěna nejen na většině tlačítek, ale i na některých polích určených k vyplňování, či pouze ke čtení (postranní informační tabulka).

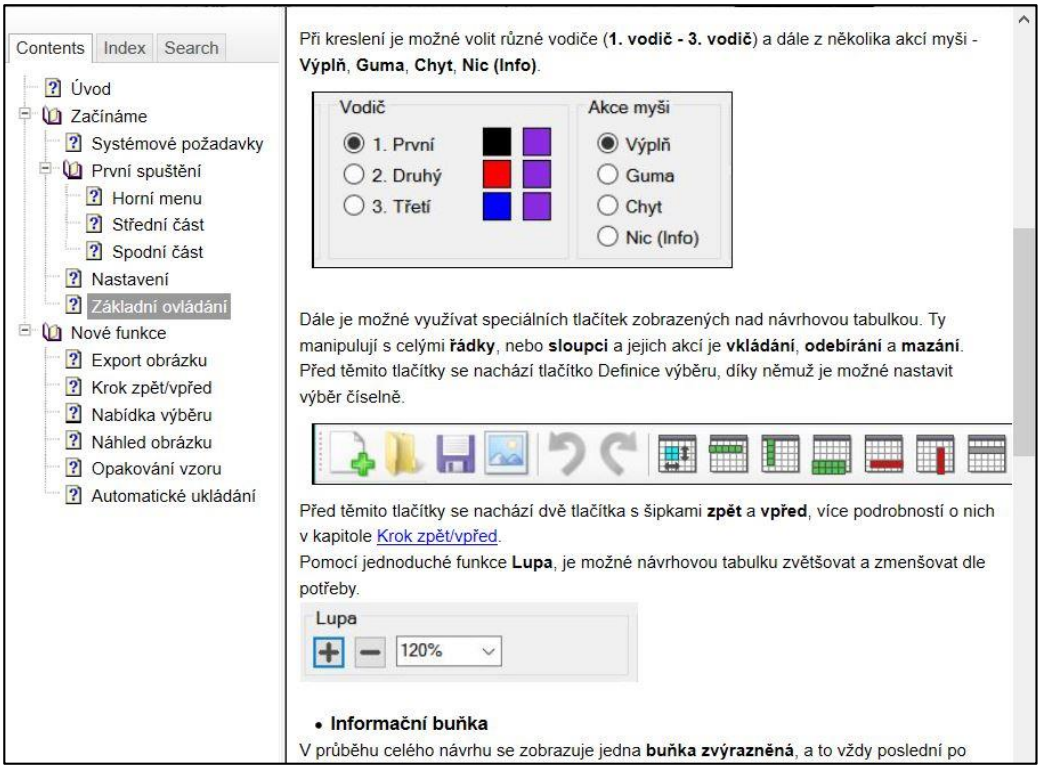
Další kompletnější forma nápovědy je k dispozici v položce horního menu, která nápovědu otevře. Tato nápověda byla vytvořena v programu HelpNDoc 4 [3], který je volně použitelný pro nekomerční účely. Jako výsledný formát nápovědy byl zvolen HTML soubor, ve kterém ve výsledku nebyly problémy s českou diakritikou, oproti klasickému formátu CHM, užívaného běžně pro Microsoft nápovědu. Vytváření nápovědy je v tomto programu velice jednoduché a intuitivní. Prostředí připomíná velice zjednodušený textový editor s možností vkládání odkazů do textu, organizací stromové struktury témat a vytváření klíčových slov pro téma.



Obr. 17: Stromová struktura výsledné nápovědy

Tato nápověda má jednoduchou stromovou strukturu uvozenou třemi výchozími položkami – Úvod, Začínáme, Nové funkce. Celá struktura je vidět na Obr. 17 z běžící nápovědy. Pod záložkou *Index* se nachází seznam všech klíčových slov, seřazených podle

abecedy, celkem jich je 56. v záložce *Search* je možné vyhledávat dle jakéhokoliv slova vyskytujícího se v textu. Jednotlivá témata nápovědy jsou doplněna o veliké množství obrázků, znázorňujících jednotlivé funkce programu, jejich umístění a příklady jejich použití včetně vysvětlení některých pravidel. Protože některé funkce jsou velice podobné jako v předchozí verzi aplikace, nejsou zde do detailu všechny popsány. Na Obr. 18 je vidět malý náhled otevřeného tématu v této nápovědě.



Obr. 18: Ukázka otevřené nápovědy

5 Závěr

Tato diplomová práce řeší reálný požadavek firmy v reakci na technický vývoj operačních systémů. Výsledná aplikace ELCAP Designer 2.0 bude používána pro návrh vzoru díla pro pletací stroj Elektrocap, umožňující hotový návrh zkompilovat do textového souboru, který lze přenést do paměti řídící jednotky stroje a ten podle instrukcí navržený produkt uplete. Během vývoje byl kladen důraz na uživatelskou přívětivost grafického prostředí, díky čemuž bude zaškolování do používání programu rychlejší.

Navzdory tomu, že se jedná o „aktualizaci“ současně používané verze aplikace, nebyl z původní aplikace přenesen žádný kód a její vývoj probíhal odděleně. K tomu přispívá i nově zvolené vývojové prostředí a jazyk programu, jímž je Visual studio 2013 jazyk C#, oproti původnímu Delphi 7. Nové prostředí bylo zvoleno také díky předem určeného cílového operačního systému Microsoft Windows 7 a vyšší.

Během postupu práce bylo řešeno několik problémů, včetně prvotního, kterým bylo zvolení vhodné komponenty pro návrhovou tabulku. Po průzkumu volně dostupných komponent, které většinou nevyhovovaly rychlostí a vhodnými funkcemi, bylo přistoupeno k vývoji vlastnímu. Díky tomu bylo možné vytvořit hlavní tabulku na míru a implementovat pouze potřebné funkce. Další vybrané problémy a jejich řešení jsou uvedeny v textu práce. Autorka zaznamenala během programování veliký vývoj v oblasti zjednodušování funkcí. Po závěrečné kontrole kódu došlo v mnoha funkcích k několika úpravám, které pouze zrealizovaly použití již existujících funkcí a metod.

Největší přínos této aplikace je shledán v možnosti běhu na novějších operačních systémech, dále také v několika nových funkcích pro návrh vzoru. Především export obrázku by měl být velice nápomocný, a to z důvodu odstranění současných závěrečných úkonů dokumentace. Tím do teď bylo ruční vybarvování vytištěného rastru na papíře podle nakresleného vzoru v počítači. Další pozitivní novinkou je možnost nastavování opakování vybraných řádků vzoru a jejich následná manipulace – skrývání, zobrazování, ale i rušení a mazání. Díky tomu je možné z několika set řádků vytvořit pouze ty základní a pracovat s nimi jako s hotovými vzory.

Použitá literatura

[1] BAYER, Jürgen. *C# 2005: velká kniha řešení*. Brno: Computer Press, 2007. Programování. ISBN 978-80-251-1620-3.

[2] DRAYTON, Peter, Ted NEWARD a Ben ALBAHARI. *C# v kostce: pohotová referenční příručka*. Praha: Grada, 2003. ISBN 80-247-0443-9.

[3] *HelpNDoc* [online]. 2016 [cit. 2016-12-28]. Dostupné z: http://www.helpndoc.com/

[4] How do I capture the mouse move event. In: *Stack Overflow* [online]. c2016 [cit. 2016-12-28]. Dostupné z: http://stackoverflow.com/questions/2063974/how-do-i-capture-the-mouse-move-event

[5] KOPAL, Jaroslav. 2003. *Pletařské, proplétací a splétací stroje: 1. část*.

[6] Limit the size of a generic collection? In: *Stack Overflow* [online]. c2016 [cit. 2016-12-28]. Dostupné z: http://stackoverflow.com/questions/14101310/limit-the-size-of-a-generic-collection/20490486#20490486

[7] Memento .NET Design Pattern in C# and VB. In: *DoFactory* [online]. c2016 [cit. 2016-12-28]. Dostupné z: http://www.dofactory.com/net/memento-design-pattern

[8] MICROSOFT. 2015. *Developer network: Visual C#* [online]. [cit. 2016-12-28]. Dostupné z: https://msdn.microsoft.com/en-us/library/kx37x362.aspx

[9] PETZOLD, Charles. *Programování ve Windows: legendární publikace o programování: Win32 API*. Vyd. 1. Praha: Computer Press, 1999, xxiii, 1216 s. Profi. ISBN 80-7226-206-8.

[10] Textilní terminologie zbožíznalství: Pleteniny. *Škola textilu* [online]. [cit. 2016-12-28]. Dostupné z: http://www.skolatextilu.cz/elearning/316/textilni-terminologie-zboziznalstvi/pleteniny/

[11] VONDRÁČKOVÁ, Jana. *Rozbor řešení aplikace pro návrh vzoru díla pro pletací stroj*. Liberec, 2015. Semestrální projekt. Technická univerzita v Liberci. Vedoucí práce Ing. Martin Diblík, Ph.D.