Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Vektorový editor v jazyce Java

Plzeň 2013 Jan Kotalík

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 5. května 2013 Jan Kotalík

Abstract

Vector editor in Java language

This bachelor thesis deals with vector graphics in the Java language. The objective of this thesis is to implement a vector editor in Java. A brief description of the usable vector – and bitmap – graphical formats is included. In this thesis, the method of drawing using graphical user interface in Java language are also analyzed. Due to limited abilities of this programming language, this work also deals with possibilities of third–party libraries, designed for working with vector graphical formats in Java. These libraries are then compared for the purpose of implementing a vector editor. The editor's implementation itself addresses the appropriate usage of available tools and third–party libraries. This document includes a detailed description of the entire implementation of the vector editor, including an UML class diagram. In the end, the results of implementation and the chosen usages of third–party libraries are described.

Abstrakt

Vektorový editor v jazyce Java

Tato práce se zabývá vektorovým kreslením v jazyce Java. Cílem práce je pak implementace vektorového editoru v tomto jazyce. Obsahuje také stručný popis použitelných vektorových a rastrových grafických formátů. V této práci jsou také rozebrány způsoby kreslení v grafickém uživatelském rozhraní v jazyce Java. Vzhledem k omezeným možnostem tohoto programovacího jazyka se zabývá i možnostmi knihoven třetích stran, zabývajících se vektorovou grafikou v Javě. Tyto knihovny jsou poté srovnávány pro účel implementace vektorového editoru. Samotná implementace pak řeší vhodné použití těchto knihoven a nástrojů ve vektorovém editoru. V tomto dokumentu je obsažen podrobný popis implementace celého vektorového editoru včetně UML diagramu tříd. V závěru popisuje výsledek implementace editoru a zvoleného způsobu použití knihoven třetích stran.

Obsah

1 Úvod

Tato bakalářská práce se zabývá možnostmi vektorové grafiky v jazyce Java s cílem implementovat jednoduchý vektorový editor, na kterém by bylo možné tyto možnosti demonstrovat. K tomu je potřeba prozkoumat podporu vektorových i rastrových formátů, jejich vhodnost využití a možnosti práce s nimi.

Bude zapotřebí se seznámit se způsobem, jakým Java vykresluje grafiku na obrazovku a zjistit, zda tato grafika může být exportována do rozličných vektorových i rastrových formátů. Pokud nebude existovat možnost v samotném Java SDK, bude zapotřebí najít takovou knihovnu třetí strany, která dokáže tento export zajistit. Dále bude potřeba vyřešit, v jakém formátu se vektorová grafika bude ukládat na disk a zpětně načítat z disku do programu, kde bude moci být modifikována. Tento formát je vhodné zvolit tak, aby byl obecně rozšířen a tudíž existovala možnost editovat jej i v jiných programech. Práci se zvoleným vektorovým formátem pak může zajišťovat libovolná knihovna, která tento formát dokáže zpracovávat a poskytovat korektní výstup na obrazovku.

Vektorový editor by měl být podobný *Microsoft Paint*, avšak na rozdíl od tohoto jednoduchého rastrového editoru by měl být založen na vektorovém kreslení, z čehož vyplývá rozdílný přístup ke grafickým primitivům. Je potřeba, aby vstup i výstup programu byla vektorová grafika libovolného formátu, je však žádoucí, aby bylo daný formát možno otevřít či editovat i v jiném programu. K řešení tohoto problému lze použít jakoukoli knihovnu třetí strany, která se daným tématem zabývá.

Program samotný by měl umět pracovat se základními vektorovými primitivy, je vhodné, aby pro uživatele nebyl příliš složitý a aby jeho použití mělo smysl oproti použití větších a specializovanějších vektorových editorů, zejména z hlediska jednoduché základní funkcionality. Editor by pak měl být schopen exportovat grafiku do různých rastrových i vektorových formátů, k čemuž je rovněž možno použít libovolnou knihovnu pro jazyk Java.

Hotový editor bude pak nutné otestovat. Vzhledem k jeho povaze – grafický editor – je vhodné udělat především uživatelské testování, může tedy být testována jeho intuitivnost a nenáročnost pro uživatele. Také je vhodné vizuálně zkontrolovat grafické výstupy editoru a zdokumentovat případné chyby a nedokonalosti externích knihoven.

2 Možnosti Javy při vykreslování grafiky

V této kapitole se budeme zabývat možnostmi Javy při vykreslování grafiky na obrazovku a její editaci.

2.1 Grafické uživatelské rozhraní v Javě

Pro vytváření grafického uživatelského rozhraní v Javě existuje několik knihoven, z nichž jsou dvě součástí Java SDK (AWT a Swing – viz níže). Existují i další proprietární knihovny třetích stran, např. Java SWT, které ale samozřejmě vyžadují připojení externích knihoven.

2.1.1 Java Abstract Window Toolkit (AWT)

Prostředí AWT vzniklo dle zdroje [1] z důvodu přenositelnosti jazyka Java, neboť se Java jako multiplatformní jazyk nemůže z hlediska programování vázat na služby konkrétních operačních systémů. Pro tento účel bylo vytvořeno jednotné rozhraní AWT, které poté zajišťuje komunikaci s konkrétním operačním systémem. Vyskytuje se ve všech verzích Java SDK a je nejstarším prostředím pro vytváření grafického uživatelského rozhraní v Javě. Java AWT obsahuje mnoho grafických komponent, jako jsou například `Button` (tlačítko), `Label` (popisek), `TextField` (textové pole) nebo `Panel` (plocha o určité velikosti) [1]. Tyto komponenty pak mohou být umisťovány a následně zobrazovány umístěním do kontejneru. V Javě se tyto kontejnery (potomci třídy `java.awt.Container`) používají pro seskupování těchto komponent. Kontejner také může obsahovat jiné kontejnery. Tímto vnořováním pak lze definovat hierarchickou strukturu grafických prvků v okně, které je rovněž kontejnerem a je zajišťováno třídou `java.awt.Frame` [1].

Java AWT má dle [2] tyto vlastnosti:

- bohaté množství komponent uživatelského rozhraní
- robustní model pro odchytávání událostí
- grafické nástroje a nástroje pro práci s obrázky, včetně tříd pro tvary, barvy a písmo
- správce nástrojů, sloužící pro rozvržení komponent (viz níže)
- třídy pro přenos dat, použitelné například pro vyjmutí a vkládání

Ke konkrétnímu rozložení prvků v kontejneru pak slouží správci rozvržení (layout managers). Ty zajišťují správnou velikost jednotlivých kontejnerů a spravují rozvržení a pozice komponent v nich [1]. Java poskytuje několik druhů těchto správců rozvržení,

které mají rozdílné vlastnosti, co se týče flexibility a možností rozvržení prvků. Patří mezi ně například správci `BorderLayout`, `FlowLayout` nebo `AbsoluteLayout`.

Java AWT také nabízí robustní systém zpracování událostí, které nastávají například při stisknutí tlačítka uživatelem. K reakci na takovou událost se pak používají třídy implementující rozhraní `EventListener`. V Javě existuje více rozhraní, které jsou potomky třídy `java.util.EventListener` [1], jsou to třídy obstarávající naslouchání konkrétních typů událostí, například `MouseListener` (chování myši) nebo `ActionListener` (stisknutí tlačítka v rozhraní). Pro reakci na událost pak musí být třída s tímto rozhraním zaregistrována u patřičného prvku. Reakce na konkrétní událost je pak zpracovávána v související metodě třídy s daným rozhraním.

2.1.2 Java Swing

Tato knihovna vznikla jako rozšíření Java AWT za účelem úplného oddělení vzhledu grafického uživatelského rozhraní od operačního systému. Jednotlivé komponenty tedy na rozdíl od Java AWT pro vykreslování komponent nijak nevyužívají služby operačního systému. Grafické uživatelské rozhraní vytvořené pomocí této knihovny vypadá tedy prakticky stejně na všech operačních systémech podporujících Javu. Jako nativní součást Javy je pak tato knihovna obsažena v Java SDK od verze 1.2 [3].

Java Swing používá obdobné komponenty jako Java AWT, rozdíl v nich spočívá kromě jiného názvu také v tom, že operační systém nemá nad těmito komponentami kontrolu a tudíž je možné zcela ovlivňovat jejich vzhled nezávisle na platformě. Třídy komponent v prostředí Java Swing jsou například komponenty `JButton`, `JTextField`, `JPanel` nebo `JFrame`.

Java Swing má následující vlastnosti [2]:

- všechny vlastnosti Java AWT
- vlastní verze všech komponent Java AWT založené pouze na Javě
- velkou kolekci složitějších komponent (například stromové zobrazení adresářové struktury)
- připojitelné *Look and Feel* – zajišťuje konkrétní vzhled komponent
- nezávislost vzhledu na operačním systému

Přístup k obsluze událostí je pak obdobný jako u Java AWT (vychází z něj). Java Swing rovněž používá správce rozvržení z Java AWT. Součásti AWT jsou pak při použití Swingu běžně dostupné, neboť jsou prakticky jeho nezměněnou součástí, a proto je možné tyto dvě knihovny bez problémů kombinovat.

2.1.3 Standard Widget Toolkit (SWT)

Pro vykreslování grafického uživatelského rozhraní v jazyce Java existují také různé knihovny třetích stran. Jednou z nich je i Standard Widget Toolkit, který je alternativou ke knihovnám Java Swing a Java AWT. Na rozdíl od těchto knihoven využívá jiný přístup k operačnímu systému pomocí Java Native Interface [4]. Tato knihovna je napsána v jazyce Java a podporována různými operačními systémy, konkrétní implementace pro daný systém jsou ale různé.

SWT poskytuje podobné vlastnosti jako Java Swing, avšak je zde rozdíl především ve vzhledu a implementaci, z čehož vyplývají i různé vlastnosti těchto knihoven. SWT byl navržen tak, aby byl rychlejší než Java Swing, čehož se snaží dosáhnout například jinou metodou vykreslování prvků. Testy rychlosti ale prokázaly, že toto urychlení je silně závislé na povaze grafického rozhraní aplikace a tudíž nelze jednoznačně rozhodnout, zda je SWT skutečně rychlejší.

Použití SWT je podobné použití Java Swing, jak vyplývá z jeho dokumentace [4]. Jak již ale bylo řečeno, SWT je nástroj třetí strany a jako takový není obsažen v nativních knihovnách Java SDK. Je tedy nutné připojovat externí knihovnu, což s sebou nese jistou nevýhodu, neboť vytvořená aplikace pak potřebuje více místa na disku nebo jiném médiu.

2.2 Možnosti vykreslování grafických primitiv

Dále budeme uvažovat pouze grafické uživatelské rozhraní vytvořené pomocí Java Swing. Při vytváření aplikací s grafickým uživatelským rozhraním je v programovacím jazyce Java často také zapotřebí zobrazit generovaný grafický obsah na obrazovce. Je tedy potřeba nějakým způsobem umožňovat kreslení grafických primitiv, případně celých obrázků (např. fotek, schémat, atd.) do grafického uživatelského rozhraní. Java pro tento účel obsahuje několik tříd, které toto umožňují.

Komponenty grafického uživatelského rozhraní obsahují grafický kontext zastoupený třídami `java.awt.Graphics` a `java.awt.Graphics2D`, které jsou podle zdroje [5] obsaženy ve třídě `java.awt.Component`, což je třída, od které dědí všechny komponenty grafického uživatelského rozhraní. Ke grafickému kontextu každé takové komponenty je pak možné přistupovat pomocí překrytí metody pro vykreslení. S takovýmto přístupem k vykreslování komponenty můžeme zobrazit grafické prvky. Podle zdroje [5] lze takto vykreslit a zpracovávat grafiku, přičemž třída `Graphics2D` umožňuje:

- vykreslení tvarů – tedy všechny geometrické útvary, které je možno vytvořit z křivek a přímých čar

- vykreslení okrajů – u tvarů mohou být zvlášť vykresleny okraje libovolné šířky a libovolnou barvou, také lze specifikovat, jak má čára vypadat, její zakončení a zaoblení rohů
- výplně – každý tvar lze vyplnit barvou, barveným gradientem, texturou a dalšími výplněmi
- transformace – každý objekt v kontextu může být přemístěn, rotován nebo škálován (zvětšován či zmenšován)
- průhlednost – každý objekt může obsahovat alfa kanál
- výřezy – vykreslení objektu může být omezeno na určitý tvar a tím dojde k jeho oříznutí
- vyhlazování hran – pomocí této techniky mohou být hrany v grafice vyhlazeny, čímž vznikne na pohled příjemnější výstup
- vykreslení textu – do kontextu lze vykreslit jakýkoli TrueType text, který je podporován operačním systémem
- vkládání obrázků – v kontextu může být vykreslen rastrový obrázek
- úprava obrázků – na rastrový obrázek mohou být aplikovány různé filtry a efekty
- tisk na výstup – zobrazenou grafiku lze odeslat na výstup, čímž je umožněno její exportování

Vzhledem k těmto skutečnostem je patrné, že za pomoci Javy lze do uživatelského rozhraní vykreslovat grafická primitiva s různými vlastnostmi. Takto vytvořenou grafiku je pak možné exportovat do zobrazitelného formátu.

2.3 Vektorový editor

Zdroj [5] popisuje, jak lze s grafickými prvky v grafickém kontextu zacházet, přičemž způsob uchovávání informace o grafice splňuje definici vektorové grafiky – informace o obsahu jsou popsány různými grafickými primitivy, jako jsou například kružnice, obdélníky, čáry nebo křivky, u kterých jsou stanoveny jejich vlastnosti. Pro vytvoření vektorového editoru je však zapotřebí také možnosti soubor uložit jako vektorový a také jej jako vektorový načíst. Vzhledem k tomu, že je žádoucí, aby bylo výstup programu možno otevřít také v nějakém běžně rozšířeném vektorovém editoru, nelze použít pouze grafický kontext dostupný v jazyce Java jako vstupní a výstupní formát, se kterým by editor měl pracovat. Je však vhodný k interakci s uživatelem a z tohoto pohledu zcela dostačující.

Java sama o sobě však nepodporuje práci s běžnými vektorovými formáty, je tedy potřeba najít formát, který bude možno načíst, převést do grafického kontextu Javy a poté také uložit do souboru. Přitom bude potřeba, aby zvolený formát splňoval několik podmínek, vyplývajících z povahy problému a ze zadání:

- existuje nástroj/rozšiřující knihovna pro jazyk Java, se kterým je zvolený formát možné zobrazit a upravovat alespoň v rozsahu potřebném pro editor
- upravenou grafiku musí být možné exportovat stejným nebo jiným nástrojem/rozšiřující knihovnou v jazyce Java do základních rastrových formátů a musí být možné ji uložit na disk ve stejném formátu, v jakém byla načtena
- ke kresbě by mělo být možné přistupovat a měnit ji přes grafické rozhraní
- formát by měl být co nejrozšířenější, aby byl spustitelný a upravitelný v jiných vektorových editorech

Existuje celá řada běžně rozšířených vektorových formátů, jako jsou například formáty SVG, PS, PDF a další. Základní problém při programování jejich editoru ale spočívá v možnosti jejich úpravy.

3 Vektorové formáty

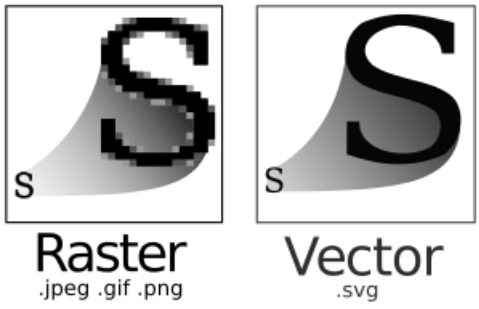
Zdroj [6] v úvodu rozděluje grafické formáty na rastrové a vektorové. Zásadní rozdíl tkví ve způsobu uchovávání informace o obrázku. Zatímco vektorové formáty popisují jednotlivé grafické objekty a přikládají jim různé vlastnosti, rastrové formáty popisují různým způsobem jednotlivé body obrázku. Z toho vyplývá několik rozdílů ve vlastnostech – vektorové formáty mohou být libovolně zvětšovány bez rizika ztráty kvality a vyhlazení hran obrázku. Velké vektorové obrázky jsou také oproti stejně velkým rastrovým obrázkům mnohem méně náročné na paměť. Taktéž umožňují změny jednotlivých objektů grafiky. Oproti tomu ve vektorovém formátu nelze ukládat obrázky jako jsou fotografie nebo malby. K takovému účelu je vhodné použít formáty bitmapové (rastrové – viz kapitola 4).

3.1 Scalable Vector Graphics (.SVG)

Jak se ukázalo při prohledávání zdrojů, existuje značné množství nástrojů umožňujících vytváření některý nebo i více souborů ve vektorovém formátu. Nástrojů schopných tyto formáty rozpoznat, zpracovat, korektně zobrazit a umožnit jejich editaci je omezené množství. Podmínky pro použití by vzhledem ke své struktuře mohl nejlépe splňovat formát *Scalable Vector Graphics* – SVG. SVG je volně dostupný formát souboru, který pomocí značkovacího jazyka XML popisuje vektorovou grafiku [7]. Je podporován skupinou *World Wide Web Consortium* (W3C) , která zajišťuje jeho podporu a normy.

3.1.1 Škálovatelnost a další výhody

Škálovatelnost je jedna z vlastností obsažených dokonce v názvu tohoto formátu a podle zdroje [8] jde o možnost přiblížení obsahu bez ztráty kvality, což je umožněno samotnou povahou vektorového formátu (viz kapitola 3). V samotném souboru jsou uloženy informace o křivkách, textech, tvarech a jiných elementech, které grafika obsahuje a zobrazení samotné pak obstarává až prohlížeč, podobně jako například u známého značkovacího jazyka HTML. Porovnání ztráty kvality obrazu při přiblížení při použití různých typů grafických formátů můžeme vidět na obrázku 3.1.



Obrázek 3.1: Srovnání typů grafických formátů

Škálovatelnost je jedna z velkých výhod vektorových formátů, avšak SVG samotné má více výhod [8], díky kterým je používáno například ve webových aplikacích:

- nezávislost na platformě – obsahuje pouze informace o grafických objektech a jejich uspořádání, nikoli o to, jak budou vykresleny jednotlivé body
- přenositelnost
- podporuje jednoduchou tvorbu animací
- jeho zdrojový kód je čitelný i pro programátora
- používá XML
- velikost souboru prakticky nezávislá na rozměrech kresby – neobsahuje informace o jednotlivých bodech obrázku, pouze o jeho velikosti a obsahu

3.1.2 Struktura

Jak již bylo řečeno, SVG je formát založený na XML, což je obecný značkovací jazyk. Začátek souboru by tedy měl podle [8] obsahovat procesní instrukci XML deklaraci `DOCTYPE`. Základní a z pochopitelných důvodů povinný tag každého SVG souboru je párová značka `<svg>`, pomocí jejíchž atributů je potřeba deklarovat velikost kresby. Velikost kresby může být zadána v pixelech i v různých jednotkách délky, například v milimetrech. Také je zde možné definovat celou řadu dalších atributů, které ovlivňují dokument, například deklaraci `namespace`. Značka `<svg>` je párová a proto je třeba ji vždy uzavřít značkou `</svg>`. Mezi těmito dvěma značkami je pak možné umístit značky jednotlivých elementů dokumentu, jako jsou značky `<path>`, `<rect>` nebo značka pro metadata – `<meta>`. V kódu na obrázku 3.2 je vidět příklad prázdného SVG dokumentu.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="5cm" height="4cm" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
</svg>
```

Obrázek 3.2: Příklad prázdného SVG dokumentu

Elementy, které dokument obsahuje vnořené v tagu `<svg>`, by podle normy měly být vykresleny v pořadí, v jakém jsou zapsané v dokumentu.

3.1.3 Základní tvary

Přestože jde každý takový grafický objekt vykreslit z křivek a přímých čar [8], SVG pro přehlednost a z dalších důvodů nabízí několik základních tvarů, jejichž existence usnadňuje orientaci v dokumentu a zachovává možnost tyto tvary editovat uživatelsky přívětivým způsobem, tedy pomocí jejich atributů. Příkladem je objekt *circle*, tedy

kružnice, jejímiž parametry můžeme ovlivňovat její pozici v dokumentu a její poloměr, což by při vykreslení stejně vypadajícího útvaru pomocí křivek nebylo možné. Mezi tyto základní tvary s vlastními tagy a atributy patří například [8]:

- cesty – tag `<path>`, vykresluje křivky po jednotlivých bodech a zakřiveních
- čáry – tag `<line>`, vykreslí přímou nebo lomenou čáru zadanou po bodech
- pravoúhelníky – tag `<rectangle>`, vykreslení obdélníků či čtverců daných počátečním bodem, šířkou a výškou
- kružnice – tag `<circle>`, vykresluje kružnici danou souřadnicemi jejího středu a poloměrem
- elipsa – tag `<ellipse>`, vykreslí elipsu danou souřadnicemi středu a dvěma poloměry
- polygon – tag `<polygon>`, vykresluje polygon daný libovolně velkou množinou bodů
- text – tag `<text>`, vykreslí text zadaný uvnitř tagu podle jeho atributů, které určují například pozici, typ písma, velikost písma apod.

Atributy jednotlivých útvarů, které mají podobný účel, často nemají stejné názvy – například atribut pro určení souřadnice v kresbě použitý v pravoúhelníku je odlišný od atributu pro souřadnici v případě kružnice – a proto je potřeba mít tuto skutečnost na paměti při vytváření algoritmů, které s formátem SVG pracují.

3.1.4 Styly a atributy

Vlastnosti jednotlivých objektů v souboru SVG lze ovlivnit dvěma různými způsoby [8]. Informace o objektech mohou být zapsány, jak již bylo zmíněno výše, pomocí atributů jednotlivých tagů. Existuje však i možnost určit vzhled dokumentu pomocí kaskádových stylů. Tyto styly mohou být vloženy přímo do určitého tagu pomocí atributu *style*, nebo je lze k dokumentu SVG připojit z externího souboru. Případně je možné definovat přímo v SVG dokumentu za pomoci tagů `<defs>` a `<style>`. Může ale dojít k tomu, že je jedna vlastnost objektu definována vícekrát s různými parametry. V takovém případě má pro vykreslení elementu nejvyšší prioritu atribut style, po něm externí styly a až nakonec přímý atribut elementu [8]. To je demonstrováno v příkladu na obrázku 3.3.

Tento SVG dokument obsahuje obdélník, který bude vyplněn modrou barvou. Pokud by tag `<rect>` neobsahoval v atributu *style* informaci o výplni, byl by tento obdélník vyplněn zeleně. Teprve v případě, kdy by se k tomuto elementu nevztahovaly žádné styly, byl by při vykreslování brán v potaz atribut *fill* a obdélník by měl červenou výplň.

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     width="10cm" height="5cm" viewBox="0 0 1000 500">
  <defs>
    <style type="text/css"><![CDATA[
      rect {
        fill: green;
      }
    ]]></style>
  </defs>
  <rect x="200" y="100" width="600" fill="red" style="fill:blue;"
        height="300"/>
</svg>
```

Obrázek 3.3: Příklad SVG dokumentu

3.1.5 Grupy

Podle zdroje [8] existují elementy s tagem <g>, které mohou obsahovat vnořené jiné elementy a tímto způsobem je sdružovat v jeden celek. Tento element se nazývá grupa. S jeho pomocí pak lze přistupovat k elementům v něm vnořeným jako k jedinému celku, pro nějž lze opět specifikovat řadu různých atributů, které se vztahují na všechny vnořené elementy. Do grupy může být vnořen jakýkoli element, tedy i jiná grupa. Každá grupa také může mít svůj vlastní titulek a popisek, definované v párových tazích <title> a <desc>. Díky tomu lze v SVG dokumentu vyznačit a popsat souvislost mezi různými objekty.

3.1.6 Transformace

Na elementy v SVG dokumentu je možné uplatnit různé transformace – translaci (přesun), rotaci, škálování (změna velikosti) nebo zkosení [8]. V základním elementu <svg> je možné podobným způsobem nastavit oblast zobrazení, což je v podstatě totéž co translace celého dokumentu. V případě objektů k vykreslení je možné přiřadit kterémukoli z nich transformaci pomocí atributu *transform*. Do tohoto atributu lze pak vložit jednotlivé transformace s parametry nebo přímo celou transformační matici. Transformační matice je matice 3x3, sdružující všechny transformace. Pomocí ní tedy lze ovlivnit všechny transformace s použitím jediného parametru v atributu. Další výhoda použití transformačních matic spočívá v jejich snadném spojování, neboť na to stačí základní matematické operace s maticemi. Na obrázku 3.4 pak můžeme vidět příklady matic s proměnnými pro základní operace – rotaci, škálování (zvětšení nebo zmenšení) a translaci (přesun).

$$\mathbf{R} = \begin{pmatrix} cos(\alpha) & sin(\alpha) & 0 \\ -sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{S} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{T} = \begin{pmatrix} 1 & 0 & P_x \\ 0 & 1 & P_y \\ 0 & 0 & 1 \end{pmatrix}$$

Obrázek 3.4: Příklady transformačních matic s proměnnými pro jednotlivé operace

3.2 PostScript (.PS)

Zdroj [9] hovoří o PostScriptu jako o jazyku pro popis stránek. Má ale větší možnosti a proto se používá i jako formát k přenosu obrázků. Podle zmíněného zdroje jde o interpretovaný jazyk, založený na práci se zásobníky. Jeho základními grafickými objekty jsou pak cesty, které lze obtahovat a vyplňovat, bitové mapy, rastry, vzorky a písma. Téměř celá jeho specifikace je volně dostupná na Internetu v publikaci PostScript Language Reference Manual (PLRM). PostScript je interpret, který při vykreslování prochází soubor s kódem, tedy soubor s příponou *.ps*, a podle tohoto kódu vykresluje grafiku do výstupní paměti.

3.2.1 Struktura

Základním prvkem PostScriptu je tedy příkaz. Příkazy v PostScriptovém souboru se při interpretaci postupně vykonávají. Velká část příkazů se pak týká přímo vykreslování, příkaz se zpravidla skládá z parametrů a operace, která se má provést, jak je uvedeno v příkladu na obrázku 3.5.

```
1 dict dup /PageSize [40 40] put setpagedevice
0.5 10 moveto
50 30 lineto
showpage
```

Obrázek 3.5: Příklad jazyka PostScript

Interpret nejprve přesune pero na souřadnice [0.5, 10] a poté vykreslí čáru do bodu [50, 30]. Hodnoty souřadnic PostScriptu jsou dle zdroje [10] udávány v bodech, z nichž každý má velikost 1/72 palce. Poslední příkaz `showpage` pak zajistí vykreslení stránky.

3.2.2 Možnosti skriptování

PostScript poskytuje více příkazů, než jen služby pro vykreslování na výstup, lze jím provádět základní matematické operace jako je sčítání, násobení a dělení, vypočítávat tak například potřebné souřadnice a provádět při vykreslování kód, přičemž může využívat proměnných. Jeho interpret obsahuje také zásobník, jenž je možné použít například k ukládání souřadnic a k jejich zpětnému načítání. Je také možné používat definice, například nadefinovat kratší zápis pro některý z příkazů, což je možné vidět v příkladu na obrázku 3.6, kde se pro příkaz `lineto` nadefinuje kratší zápis „line“.

```
/line { lineto } def
```

Obrázek 3.6: Příklad definice v jazyce PostScript

3.2.3 Vlastnosti

PostScript je podle [10] nezávislý na výstupním zařízení pro tisk, což je jedna z jeho hlavních výhod. Zdroj [9] hovoří o tom, že struktura PostScriptu je navržena tak, aby skript při interpretaci nemusel udržovat v paměti obrázek v maximální bitové hloubce(viz kapitola 4), ale pouze v bitové hloubce výstupního zařízení. To přináší velkou úsporu paměti, protože skript nikdy nečte to, co již bylo vykresleno a nevrací se k dříve vygenerovaným datům. To s sebou ale nese dvě omezení [9]:

- chybí podpora průhlednosti – skript nemá informace o aktuálním bodě kresby, tudíž nemůže přepočítat barvu na základě dalších informací
- chybí podpora Floyd–Steinbergova rozptylu (slouží pro vytváření obrazu s menším počtem barev než má obraz původní tak, aby byl obraz vizuálně co nejpodobnější předchozímu) – tento algoritmus vyžaduje znalost okolních pixelů ve vstupní bitové hloubce, kterou PostScript neposkytuje.

3.3 Portable Document Format (.PDF)

Zdroj [11] popisuje typický PDF soubor jako soubor obsahující tisíce objektů, mechanismus mnohonásobné komprese, různé formáty písma a kombinaci vektorové a rastrové grafiky. Také může obsahovat různá metadata, tedy informace o dokumentu, jako je například titulek, jméno autora a klíčová slova. Tato metadata nemají žádný vliv na grafický výstup při otevření souboru, avšak používají se pro popis jeho obsahu a usnadňují vyhledávání.

3.3.1 Syntaxe a struktura

PDF soubor podle zdroje [11] obsahuje nejméně tři rozdílné jazyky, které tvoří základní syntaxi PDF a určují jeho vzhled a chování. Každá z částí dokumentu je tedy popsána za pomoci jiné syntaxe. Jedná se o tyto části PDF:

- obsah dokumentu – zde jsou informace jako stránky, metadata, fonty a zdroje, kterými je dokument popsán
- obsah stránky – popisuje použití textu a grafiky na jedné stránce
- struktura souboru – části souboru, které PDF obsahuje, typicky se PDF skládá z několika níže popsaných částí zdrojem [12], jedná se o:
    - hlavičku – obsahuje verzi dokumentu PDF, například:

```
%PDF-1.7
```

    - tělo dokumentu – obsahuje série objektů obsažených v PDF dokumentu, viz níže

```
1 0 obj
...
endobj
2 0 obj
...
endobj
...
```

- referenční tabulku – specifikuje pozice objektů tímto způsobem:

```
xref
0 6
0000000000 65535 f
0000000010 00000 n
0000000079 00000 n
0000000173 00000 n
0000000301 00000 n
0000000380 00000 n
```

- trailer – obsahuje informace o tom, kde dokument začíná

```
trailer
<<
  /Size 6
  /Root 1 0 R
>>
startxref
492
%%EOF
```

Čtení PDF pak podle zdroje [12] neprobíhá sekvenčně, tedy shora dolů, avšak postupnou analýzou těchto částí souboru. Způsob, jakým to aplikace pro čtení PDF provádí, je podrobně popsán ve specifikaci PDF [13].

3.3.2 Vlastnosti

PDF na rozdíl od například formátu SVG neobsahuje možnost zapsat běžný grafický objekt, jakým je například kružnice, avšak veškeré grafické prvky zapisuje ve formátu křivek. To PDF omezuje, co se týče čitelnosti kódu z pohledu uživatele. Na rozdíl od PostScriptu však PDF od verze 1.4 umožňuje vykreslení průhlednosti a jinou interakci s již nakreslenými objekty. Tento formát je pak pro člověka obecně poměrně nepřehledný a nečitelný, jeho ruční editace je však možná. PDF má velkou podporu ze strany aplikací pro jeho čtení, které dokument vykreslí vždy stejně, což dělá z PDF velmi užitečný formát pro přenášení dokumentů ve finální podobě. Co se však týče načítání pro editaci, PDF není pro takovéto použití navrženo.

4 Rastrové formáty

Podle zdroje [6] jsou všechny rastrové obrázky tvořeny dvourozměrným polem, které obsahuje informace o barvě bodu – pixelu – na dané pozici v poli. Pokud je hustota bodů na jednotku plochy dostatečná (neboli jsou–li body dostatečně malé a blízko sebe), lidské vnímání není schopno rozeznat, že jde o pole různých barvených bodů, ale vnímá celý obrázek.

Jedním z pojmů, používaných v rastrové grafice, je pak bitová hloubka barev. Tento parametr udává, kolik paměti je použito na definici barvy jednoho pixelu. Větší bitová hloubka pak znamená větší spektrum barev – čím více bitů je k dispozici pro uchovávání barvy, tím více různých barev je možné rozlišit a zaznamenat. Přímo úměrně velikosti bitové hloubky se pochopitelně zvyšuje i paměťová náročnost na uchovávání obrázku.

4.1 Windows Bitmap (.BMP)

Tento formát je jeden z nejjednodušších, používá se jako nativní grafický formát v operačních systémech Windows [6]. Tento formát podporuje obrázky s 1, 4, 8, 16, 24 a 32 bity na pixel, tedy bity barevné hloubky. BMP také poskytuje kompresi pro hloubku 4 a 8 bitů na pixel, která spočívá ve sjednocení informace o barvě bodů, které jsou rozmístěny na velké ploše v obrázku a mají stejnou barvu.

Každý soubor uložený ve formátu BMP obsahuje dvě hlavičky s informacemi. Jedná se o hlavičku souboru a hlavičku obrázku. Celý soubor začíná hlavičkou souboru. Tato hlavička obsahuje několik údajů, jako je typ souboru, jeho velikost nebo offset od začátku souboru odkazující tam, kde začínají data o obrázku. Jedná se tedy o datovou strukturu nesoucí informace o souboru a obrázku.

Podle zdroje [14] ukládá BMP data o obrázku ve formátu *Device–Independent Bitmap* (DIB). Z názvu vyplývá, že je tento formát nezávislý na výstupním zařízení. Grafika uložená v tomto formátu má pak maximální možnou kvalitu. BMP totiž zpravidla neupravuje data, neboť často nebývá komprimovaný, popřípadě používá pouze bezztrátovou kompresi. A právě protože formát umožňuje pouze základní a velmi specifickou kompresi, zastoupenou pouze slučováním informací o větších plochách stejné barvy, je jeho hlavní nevýhodou to, že jsou obrázky uložené v BMP velmi velké a paměťově náročné.

4.2 Joint Photographic Experts Group (.JPEG)

Tento formát je podle zdroje [14] hojně užíván především k ukládání fotografií a jiných obrázků, u které je kvůli kvalitě zapotřebí mít velkou barevnou bitovou hloubku. JPEG

používá ztrátovou kompresi, což znamená, že například při konverzi do JPEG z jiného formátu již nebude možné dostat se na předchozí kvalitu zobrazení, neboť JPEG skutečně „zahazuje“ některá data při kompresi. Jako algoritmus pro kompresi používá JPEG diskrétní kosinovou transformaci, která patří do skupiny metod provádějících takzvané transformační kódování nad diskrétním (vzorkovaným) jednorozměrným či vícerozměrným signálem [15]. Podobá se Fourierově transformaci. Formát JPEG však nepodporuje průhlednost ani animace.

Shrnutí vlastností [16]:

- ztrátová komprese pomocí diskrétní kosinové transformace
- 16,7 milionů barev
- nepodporuje průhlednost
- adaptován pro barevné gradienty a fotografie

4.3 Graphics Interchange Format (.GIF)

Zdroj [14] popisuje GIF jako formát používající bezztrátovou kompresi, konkrétně algoritmus LZW. Při této komprimaci se každé barvě přiřadí index a soubor se pak ukládá po řádcích. Z tohoto důvodu je pak ve formátu GIF omezen počet barev, a to jak minimálním počtem – jedním bitem, tedy dvěma barvami – a maximálním počtem – osmi bity, tedy 256–ti barvami.

Díky účinné kompresi je výhodou formátu GIF nízká velikost souboru, která je však vyvážena značně omezeným počtem barev, což činí GIF nevhodným k ukládání například fotografií, kde je barevná hloubka velmi důležitá. Na rozdíl například od formátu JPEG formát GIF umožňuje zápis průhlednosti i tvorbu animací, což z něj v minulosti dělalo oblíbený formát pro webovou grafiku, avšak v dnešní době je již vytlačován modernějším formátem PNG.

Shrnutí vlastností [16]:

- bezztrátová komprese pomocí LZW
- maximálně 256 barev
- podpora průhlednosti

4.4 Portable Network Graphics (.PNG)

Formát PNG byl navržen za účelem nahrazení zastaralého formátu GIF [16]. Podporuje 24–bitovou barevnou hloubku a 8–bitový alfa kanál, což je stupeň průhlednosti (a tedy následné ovlivnění barvy pixelu barvou pod ním). Podporuje i možnost animací, avšak

na rozdíl od GIFu je jejich použití náročnější, takže v této oblasti zůstává GIF stále používaný.

PNG používá bezztrátovou kompresi, která obsahuje vylepšení, spočívající v použití předběžných filtrů, které jsou řádek po řádku aplikovány na celý obrázek ještě před samotnou kompresí [16]. Tyto metody filtru pak pro každý řádek zjišťují rozdíl mezi souvisejícími pixely. Na takto upravená data o obrázku je pak možné uplatnit efektivnější kompresi, neboť se předpokládá, že související pixely budou mít podobnou barvu. Samotná komprese pak využívá metodu DEFLATE, což je kombinace slovníkového algoritmu LZ77 a Huffmanova kódování.

Shrnutí vlastností [16]:

- bezztrátová komprese metodou DEFLATE
- 16,7 milionů barev
- plná podpora alfa kanálu a průhlednosti
- W3C standard

5 Externí knihovny

Pro práci s vektorovými formáty v jazyce Java existuje několik volně dostupných knihoven třetích stran. V této kapitole se budeme zabývat těmi, které byly v rámci této práce nalezeny.

5.1 Apache Batik

Apache Batik je sada nástrojů postavených na programovacím jazyce Java pro aplikace nebo applety, které potřebují používat obrázky ve vektorovém formátu SVG k různým účelům, jako je zobrazení, generování nebo manipulace [17]. Knihovna je závislá pouze na Javě a jako taková je multiplatformní a použitelná na různých typech hardwarových zařízení.

5.1.1 Vlastnosti

Smysl projektu Apache Batik spočívá v poskytnutí několika klíčových modulů pro přístup k SVG programátorům. Mezi základní poskytované moduly této knihovny patří:

- SVG Generator – modul s jehož pomocí lze generovat SVG dokument
- SVG DOM – modul umožňující programátorovi přístup k DOM stromu SVG dokumentu a jeho úpravy
- Bridge – modul pro převod SVG dokumentu do vnitřní implementace Batiku
- UI Component – modul umožňující komunikaci knihovny s grafickým uživatelským rozhraním a zajišťující zobrazení SVG dokumentu
- Transcoder – modul pro načítání vstupu a generování různých výstupů, umožňuje konverzi SVG do jiných formátů

Zdroj [17] také popisuje knihovnu Apache Batik jako nástroj schopný generovat SVG obsah, zobrazit SVG obsah nebo také konvertovat SVG obsah do jiných formátů. Dle tvůrců se také vyznačuje implementací velké části SVG normy, definované skupinou *World Wide Web Consortium* (W3C).

5.1.2 Použití

Batik používá celou řadu vlastních specializovaných nástrojů a tak se celá knihovna stává poněkud uzavřeným systémem, přestože programátorovi umožňuje například odchytit události v SVG či mnoho dalších různých způsobů, jak zacházet s obsahem. Například jeho zobrazovací modul se vyznačuje specializovaným grafickým kontextem `SVGGraphics2D`, který je sice standardnímu `java.awt.Graphics2D` velmi podobný, ale přístup a zacházení s ním se od standardního grafického kontextu Javy poměrně podstatně liší. Protože Apache Batik obsahuje mnoho součástí, stává se tak pro

programátora, jenž s ním pracuje poprvé, poněkud složitější záležitostí a způsob použití není vždy zcela zřejmý z dokumentace. Domovská stránka projektu však obsahuje mnoho návodů a prezentací, pomocí nichž je programování s Batikem jednodušší. Knihovna se vyznačuje vytvořením vlastního přístupu k SVG souboru a vytváří nad Javou vlastní implementaci, jejíž použití není vždy zcela jasné. Specifická implementace však zaštiťuje velké množství možností a zjednodušení pro práci s formátem SVG a je v mnoha ohledech zcela dostačující.

5.1.3 Shrnutí

Pro účely editoru je ale třeba, aby bylo obsah možno načíst, editovat a exportovat do různých formátů. Moduly knihovny umožňují načtení souboru z disku i jeho export, avšak editace obsahu probíhá zpravidla na úrovni kódu a zpracování vnějších vstupů od uživatele se Batik příliš nevěnuje, tudíž by při implementaci editoru mohly nastat problémy související s interakcí uživatele s kresbou.

Apache Batik obsahuje velké množství modulů, které by byly dostačující při většině implementace vektorového editoru dle zadání. Tyto moduly mají vlastnosti, umožňující zjednodušení programování editoru. Jedná se však o komplexní a složitou, leč dostatečně dokumentovanou množinu nástrojů. Jejich případná úprava by ale při nutnosti zabezpečení dalších funkcionalit mohla být velice složitá. Batik také obsahuje mnoho nástrojů, které by při implementaci editoru byly zcela nepotřebné. Tato specializovanost a uzavřenost knihovny nemusí být v mnoha případech na závadu, avšak pro vytvoření komplexního programu pracujícího s vektorovou grafikou je zapotřebí, aby nástroj nebránil implementaci jeho stěžejních funkcí. Rozšiřující knihovna Batik také neobsahuje funkce umožňující interakci uživatele s grafickým obsahem, tudíž by při implementaci editoru bylo zapotřebí tento problém vyřešit.

5.2 SVG Salamander

Domovská stránka knihovny SVG Salamander, tedy zdroj [18], popisuje tuto knihovnu jako SVG engine, navržený jako malý a rychlý nástroj umožňující programátorovi pracovat se SVG formátem s minimálním úsilím.

5.2.1 Vlastnosti

Tato knihovna je zaměřená především na integraci SVG do Java her, čímž umožňuje designérům snáze tvořit grafický 2D obsah [18]. Lze ji však použít i na jiný typ aplikací. Sama domovská stránka pak specifikuje některé vlastnosti knihovny takto:

- zjednodušení načítání a vykreslování obrázků pomocí třídy `SVGIcon`

- podstatně menší délka kódu než knihovna Batik, pro chod SVG Salamanderu stačí jediný JAR soubor
- přímý přístup ke stromu SVG dokumentu vykreslenému na obrazovce pomocí příkazů Javy
- snadný přístup k pojmenovaným tvarům v SVG dokumentu
- zjišťování, které elementy se nachází na dané souřadnici, což je využitelné například při implementaci přístupu uživatele k obrázku pomocí myši
- rychlé překreslení obrázků díky možnosti překreslit pouze některé elementy
- snadné vykreslování do libovolného grafického kontextu, neboť na rozdíl od Batiku SVG Salamander nemá vlastní grafický kontext a vykreslování může provést například do kontextu Graphics2D
- rozšířené možnosti importu SVG pomocí automatického importu dokumentů v odkazu, dokonce i dokumentů uložených na serveru

SVG Salamander má dle zdroje [18] vytvořenou většinu implementace potřebné pro animace. Zdrojový kód knihovny je pak na domovské stránce projektu webu Java.net dostupný pod licencemi LGPL a BSD. Dokumentace pak obsahuje kompletní Javadoc a několik tutoriálů pro zacházení s touto externí knihovnou. SVG Salamander však neumožňuje převést otevřené SVG zpět do souboru, pouze je schopný jej vykreslit do grafického kontextu jazyka Java.

5.2.2 Použití

Pro použití knihovny k vykreslení obsahu SVG souboru do grafického kontextu Javy je třeba udělat několik základních kroků. Nejprve se vytvoří třída `SVGUniverse`, což je dle své dokumentace třída, sloužící jako kontejner pro jednotlivá zobrazení SVG a zajišťuje reference mezi nimi. Taktéž umožňuje načtení SVG z externího souboru. Tím vytváří instanci třídy `SVGDiagram`, která slouží jako popis DOM stromu dokumentu SVG v jazyce Java, kde jsou jednotlivé elementy popsány jako objekty a `SVGDiagram` určuje jejich rozložení. Tato třída také poskytuje mnoho možností práce s elementy, mimo jiné například umožňuje jejich výběr. Diagram lze vykreslit do grafického kontextu pomocí metody `render()`. Lze jej také průběžně aktualizovat. Tento diagram je praktickým základem pro práci s formátem SVG v grafickém kontextu. Jeho elementy lze editovat různými způsoby, neboť knihovna zajišťuje přímý přístup k atributům a tagům. Elementy lze libovolně přidávat a odebírat z diagramu. Při špatném použití, například při pokusu přistoupit k neexistujícímu atributu, nastávají výjimky. Jakékoli vykreslení pak probíhá prostřednictvím standardních grafických kontextů jazyka Java, jako jsou například `Graphics` a `Graphics2D`.

5.2.3 Shrnutí

Externí knihovna SVG Salamander pro práci s formátem SVG nabízí základní, leč přehlednou a pružnou funkcionalitu, v níž se často opírá o objekty, které je možné nalézt ve standardních knihovnách Javy. Nenabízí rozsáhlé a specializované množství funkcionalit, které by usnadňovaly práci programátorovi, ale právě kombinace přímého přístupu k vlastnostem SVG dokumentu prostřednictvím obrazu jeho DOM, popsaného objekty v Javě, a vykreslování do standardních grafických kontextů Javy umožňuje vytvořit téměř jakoukoli funkcionalitu programu, k jehož vytvoření je knihovna použita. Implementaci této funkcionality však zjednodušuje jen omezeným množstvím metod. Knihovna je pak schopna korektně vykreslit základní vlastnosti SVG, přičemž obsahuje i možnost efektivně pracovat s jeho objekty. Neobsahuje však řadu komponent, které by kompletně pokryly širší požadavky některých aplikací. Vzhledem k integraci knihovny do standardních knihoven Javy je ale pravděpodobné, že bude SVG Salamander kompatibilní s použitím dalších doplňujících knihoven.

5.3 VectorGraphics2D

Jak již název napovídá, VectorGraphics2D je knihovna sloužící pro převod grafiky z `java.awt.Graphics2D` do obecných vektorových formátů, jmenovitě EPS, SVG a PDF, jak uvádí domovská stránka – zdroj [19]. Také podporuje export do bitmapových obrázků. Je uvedena pod licencí LGPL.

5.3.1 Vlastnosti

Sami tvůrci považují za jednu z jejích předností to, že je malá a zanechává po sobě jen malý otisk – je tedy vhodná především pro malé a rychlé programy, například do mobilních zařízení [19]. Oproti tomu však tato knihovna neimplementuje všechny možnosti `Graphics2D`, například gradienty, vkládání externích písem nebo zpracování metadat.

Na domovské stránce této knihovny jsou uvedeny odkazy na další podobné knihovny, kde je podporováno více možností. Jednou z nich je i součást Apache Batik nebo FreeHEP Vector Graphics.

5.3.2 Použití

Co se týče uživatelské podpory ohledně používání této knihovny, komunita je zde slabší než u ostatních a popisy použití poněkud více strohé. Její užití však spočívá prakticky jen z vytvoření speciálního grafického kontextu, do kterého je třeba vykreslit požadovanou grafiku. Tento grafický kontext má nastavitelné parametry jako jsou například rozměry. V této knihovně existují různé typy grafických kontextů lišících se podle typu výstupního souboru, vyskytují se tu tedy třídy jako `SVGGraphics2D` nebo

`PDFGraphics2D`. Výsledný grafický kontext se na závěr vyexportuje do požadovaného souboru použitím `FileOutputStream` z nativní knihovny Javy `java.io` [19].

5.3.3 VectorGraphics2D – závěr

Tato knihovna nabízí přijatelnou možnost exportování grafického kontextu do různých formátů, které je možno s její pomocí uložit na disk. Její hlavní výhodou oproti ostatním knihovnám, zabývajícím se touto problematikou exportu, je její velikost. To však má za následek menší škálu formátů a omezení funkcionality pouze na části zcela nutné a potřebné k deklarovaným funkcím. Také dokumentace a příklady použití knihovny jsou strohé a špatně dohledatelné. Knihovna se očividně hodí především do malých a rychlých aplikací, které vyžadují možnost exportování jen do několika různých formátů, přičemž je nutné implementovat za pomoci knihovny export každého formátu zvlášť.

5.4 FreeHEP VectorGraphics

5.4.1 Vlastnosti

Další knihovnou pro převod grafického kontextu Javy do výstupních souborů je, rovněž pod licencí LGPL, knihovna FreeHEP VectorGraphics. Na rozdíl od VectorGraphics2D je větší a implementuje větší část možností Graphics2D. Také podporuje export do více formátů, jmenovitě PostScript, PDF, EMF, SVF a SWF. K tomu také dokáže exportovat grafiku do rastrových formátů jako je GIF, PNG, JPG a PPM [20].

Oproti VectorGraphics2D má také dostupnější dokumentaci, demonstrace a příklady, které usnadňují její použití v praxi. Je také pravidelně testována [20].

5.4.2 Použití

Knihovna FreeHEP VectorGraphics oproti VectorGraphics2D nabízí vlastnost, která významně usnadňuje programování exportu. Touto vlastností je jednoduchý dialog pro export, k jehož inicializaci stačí pouze vytvoření pomocí konstruktoru a poté zavolání metody pro zobrazení obsahující informaci o tom, ze které GUI komponenty má být vybrán grafický kontext, která grafická komponenta je rodičem této komponenty, nebo jaký řetězec má obsahovat titulek dialogu. Ze strany programátora se vyžaduje pouze tato činnost, knihovna samotná pak sama nabídne uživateli všechny dostupné formáty pro export spolu s veškerým dostupným nastavením v dialogu. Export samotný je obstarán také poté, co uživatel potvrdí okno dialogu. Knihovna vedle toho však podporuje i ruční způsob exportu, který je velmi podobný způsobu, jež používá knihovna VectorGraphics2D – tedy vykreslování obsahu standardního grafického

kontextu do speciálního grafického kontextu, ze který je pak možné exportovat jako soubor zvoleného typu [20].

5.4.3 Závěr – FreeHEP VectorGraphics

Tato knihovna je svým použitím velmi podobná předchozí knihovně VectorGraphics2D, rozdíl mezi nimi však spočívá především ve velikosti a v dostupných vlastnostech. FreeHEP VectorGraphics je větší, avšak pro programátora mnohem uchopitelnější, a to jak obsáhlejší dokumentací a příklady použití, tak nástroji, které poskytuje. V základních případech užití však rozdíly mezi těmito dvěma knihovnami mizí a zůstává jen porovnání velikosti. Z těchto důvodů lze prohlásit, že knihovna FreeHEP VectorGraphics je vhodná pro aplikace, od nichž se očekávají možnosti exportu do více různých formátů, zatímco pro aplikace vyžadující export jen do jednoho nebo několika formátů je vhodnější využít knihovnu VectorGraphics2D.

6 Analýza vektorového editoru

V této kapitole se budeme zabývat analýzou samotného vektorového editoru, konkrétně možnostmi jeho implementace.

6.1 Specifikace požadavků

Vlastní editor by měl splňovat několik rámcových požadavků daných zadavatelem. V této části budou tyto požadavky uvedeny a rozebrány.

6.1.1 Požadavky na funkčnost

Vytvořený vektorový editor má rámcově být obdobou editoru *Malování.* Stejně jako tento program má zajišťovat základní práci s kresbou, avšak jako vektorový editor. Mělo by tedy být za pomoci tohoto editoru možné vytvářet grafické prvky a tvary, jako jsou čáry, obdélníky nebo kruhy. S těmito prvky by mělo být možné dále pracovat a manipulovat, například měnit jejich barvy nebo je přesouvat. Také je vhodná možnost vkládat do kresby text. Měl by také existovat způsob, jak kresbu vizuálně zvětšit nebo zmenšit, aby bylo možné s ní lépe pracovat. Z požadavků na tento vektorový editor je možné také zmínit požadavek s nižší prioritou, jako je například přítomnost nějakého druhu mřížky, která by umožňovala vyšší přesnost při kreslení.

6.1.2 Požadavky na vstup a výstup

Editor by měl používat takový vektorový formát, který lze uložit na disk a posléze případně otevřít v jiném běžně dostupném vektorovém editoru. Tento formát by měl editor být schopen zpětně načíst a nadále upravovat. Výstup by pak měl být rozšířen o možnost exportu do několika jiných vektorových i rastrových formátů, žádoucí jsou především formáty běžně používané. Co se týče rastrových formátů, měl by být kladen důraz na ty, které jsou vhodné pro export kresby vytvořené pomocí vektorové grafiky. S nižší prioritou je pak požadována možnost nastavit rozlišení DPI u exportovaných rastrových formátů.

6.1.3 Požadavky na uživatelské rozhraní

Uživatelské rozhraní vektorového editoru by mělo být méně obsáhlé a přehlednější než u rozsáhlejších běžně dostupných vektorových editorů. Jednotlivé prvky a způsob jejich používání by pak mělo alespoň částečně připomínat *Malování*, aby pro uživatele tohoto rastrového editoru nebylo příliš obtížné se ve vytvořeném vektorovém editoru zorientovat.

6.2 Případy užití

Vektorový editor by měl sloužit jedinému koncovému uživateli, který by s jeho pomocí mohl vytvářet, upravovat a exportovat do různých formátů jednoduchá schémata a kresby. Z toho lze vyvodit několik případů užití.

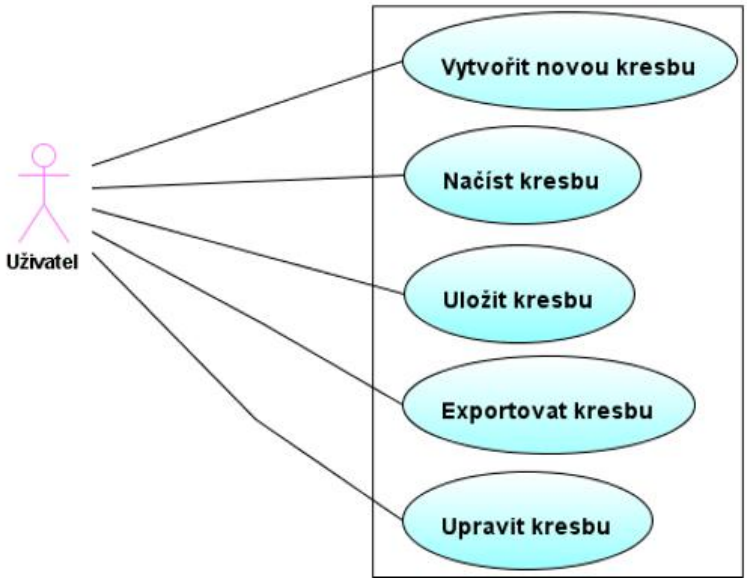
6.2.1 Popis případů užití

V případě tohoto vektorového editoru byly stanoveny tyto případy užití:

- vytvoření nové kresby – uživatel může vytvořit novou kresbu ve vektorovém formátu a určit její rozměry
- uložení vytvořené kresby – uživatel má možnost uložit vytvořenou kresbu na disk
- načtení kresby – uživatel může uloženou kresbu zpětně načíst a editovat její obsah
- exportovat kresbu – uživatel může exportovat kresbu do různých formátů, které editor neumí načíst
- upravovat kresbu – uživatel může upravovat vytvořenou i načtenou kresbu, přidávat do ní různá grafická primitiva a pracovat s nimi

6.2.2 Diagram případů užití

Z uvedených případů užití lze sestavit diagram případů užití, který vidíme na obrázku 6.1.



Obrázek 6.1: UML diagram případů užití programu

6.3 Využití knihoven

Z výše uvedené analýzy knihoven lze určit, ze vzhledem k povaze všech zjištěných knihoven se ve vektorovém editoru nabízí jen jeden formát, se kterým všechny tyto pracují a lze jej tedy využít jako nativní a základní formát editoru – SVG. Co se však týče samotných knihoven, z uvedených se nabízí dvě různé základní možnosti. První z nich je použití knihovny Apache Batik, jejíž hlavní a velkou výhodou je velké množství nástrojů pro práci s grafikou, včetně například přibližování. Její velkou nevýhodou je ale právě tato rozsáhlost, se kterou se špatně pracuje, a také fakt, že Batik pracuje pouze se specializovaným grafickým kontextem, u kterého hrozí možnost, že nebude možné žádným způsobem implementovat některé ze stěžejních funkcí editoru. Rovněž bylo zjištěno, že Batik neposkytuje možnost získat element vyskytující se na daných souřadnicích, což je funkce pro editor zcela nezbytná a vzhledem ke speciálnímu grafickému kontextu Batiku hrozí reálná možnost, že tuto funkci nebude možné doimplementovat bez hlubších zásahů do samotné knihovny.

Jako druhá možnost při volbě knihoven použitých pro vektorový editor se jeví použití knihovny SVG Salamander. Oproti Batiku lze očekávat nutnost implementace funkcionalit editoru, které knihovna neposkytuje, například zmíněné přibližování, avšak vzhledem k tomu, že SVG Salamander pracuje s nativním a výborně dokumentovaným grafickým kontextem Javy, lze předpokládat, že existují možnosti, jak tyto chybějící funkce doimplementovat přímo pro účely editoru. SVG Salamander ale nenabízí žádný způsob, kterým by se dal SVG diagram, který uchovává, exportovat do souboru. Tento problém ale lze vyřešit za pomoci další knihovny, která dokáže tento soubor vyexportovat z nativního grafického kontextu Javy. Pro tento účel se hodí jak knihovna VectorGraphics2D, tak FreeHEP VectorGraphics. Vzhledem k tomu, že je vyžadováno co největší množství možností k exportu grafiky, FreeHEP VectorGraphics se jeví jako vhodnější knihovna k použití. Srovnání těchto dvou možností přineslo rozhodnutí, že pro vektorový editor bude použita kombinace knihoven SVG Salamander a FreeHEP VectorGraphics, neboť implementace editoru za pomoci těchto knihoven bude pravděpodobně časově náročnější, avšak existuje větší pravděpodobnost, že bude splnitelná, což je vzhledem k důležitosti časových termínů při vytváření tohoto projektu rozhodující.

6.4 Architektura

Při návrhu vektorového editoru je potřeba navrhnout takovou architekturu, která by umožňovala především dobré zázemí pro přidávání a upravování jednotlivých nástrojů, se kterými bude možno pracovat na kresbě.

6.4.1 Třívrstvá architektura

Původním záměrem při návrhu vektorového editoru bylo vytvořit tzv. třívrstvou architekturu, tedy architekturu, kde by byl program rozdělen na tři vrstvy – prezentační, aplikační a datovou. Záhy se ale ukázalo, že vzhledem k povaze problému vektorového grafického editoru by bylo úplné použití této architektury značně kontraproduktivní, neboť data (kresba) vyžadují přímé zásahy ze strany uživatele, tedy z prezentační vrstvy. Grafický editor by byl bez prezentační vrstvy zcela nepoužitelný, protože jeho účel spočívá v úpravě grafiky, kterou je třeba zobrazit a interaktivně upravovat. Tyto tři vrstvy tedy často splývají.

6.4.2 Architektura vhodná pro vektorový editor

Při implementaci byl tedy použit poněkud odlišný model, jehož jádrem je grafická komponenta, obsahující grafický kontext, jehož úpravy jsou vyžadovány. Tato komponenta je součástí grafického uživatelského rozhraní, s nímž je úzce propojena. Přístup k obsahu je pak realizován za pomoci správce nástrojů, který umožňuje jednotlivým nástrojům pro úpravu, aby ovlivňovaly obsah grafického kontextu. Části programu, které bylo možno lépe oddělit, jsou pak nezávislejší na jádře a prakticky pouze zprostředkovávají dané operace, což můžeme vidět například u té části programu, která implementuje ukládání a export. Tato architektura, jejíž UML diagram tříd lze vidět v příloze A, tudíž umožňuje postupné přidávání nezávislých nástrojů a možnost výměny knihovny pro export a ukládání.

7 Implementace

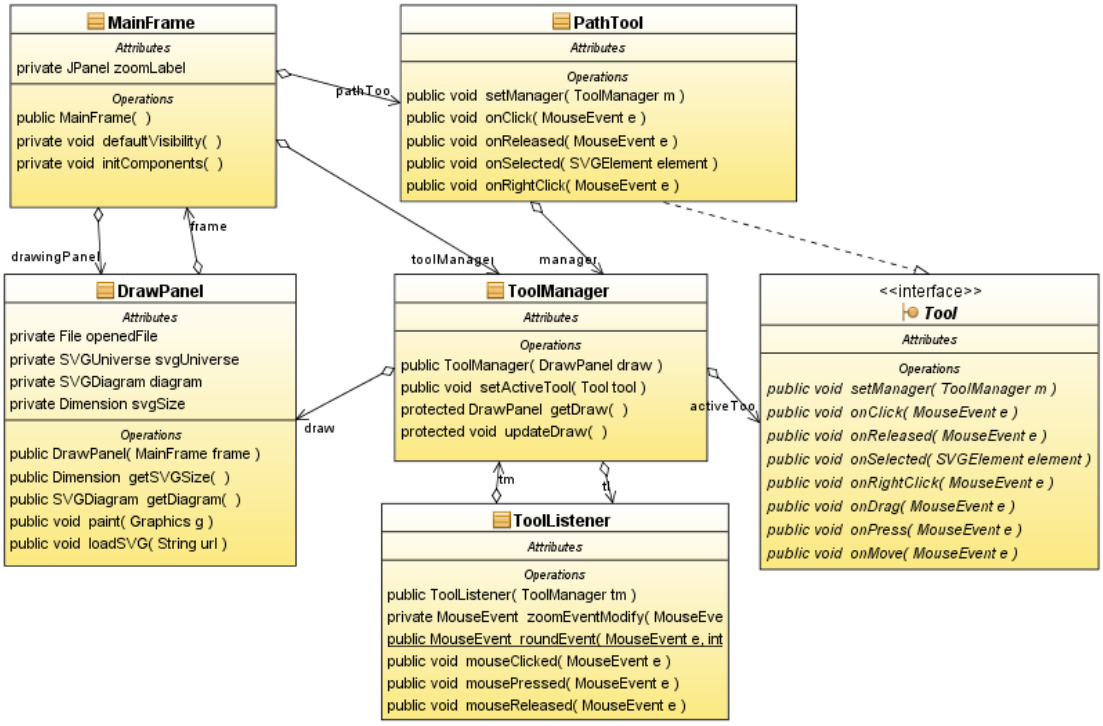
V této kapitole se budeme zabývat implementací samotného vektorového editoru, konkrétně způsoby, jaké byly v této implementaci použity a jaké důvody k tomu vedly.

7.1 Výsledná struktura vektorového editoru

Struktura byla navržena s ohledem na snadnou implementaci jednotlivých nástrojů, které budou v editoru zapotřebí. K těm bude přistupováno za pomoci rozhraní a tak budou využívány jejich funkce. Současně se složitějšími nástroji bude implementováno i více služeb jiných komponent, které by nástroje mohly potřebovat. Struktura programu je pak rozdělena do jednotlivých balíků:

- core – jádro programu, obsahuje hlavní třídu a komponentu kresby
- gui – grafické uživatelské rozhraní, obsahuje komponenty pro jednotlivá okna a dialogy programu
- io – obsahuje třídy využívané při zajišťování vstupů a výstupů programu
- tools – tento balík obsahuje jednotlivé nástroje a třídy zajišťující jejich správu
- misc – pomocné třídy, například GUI komponenty třetích stran pro výběr písma

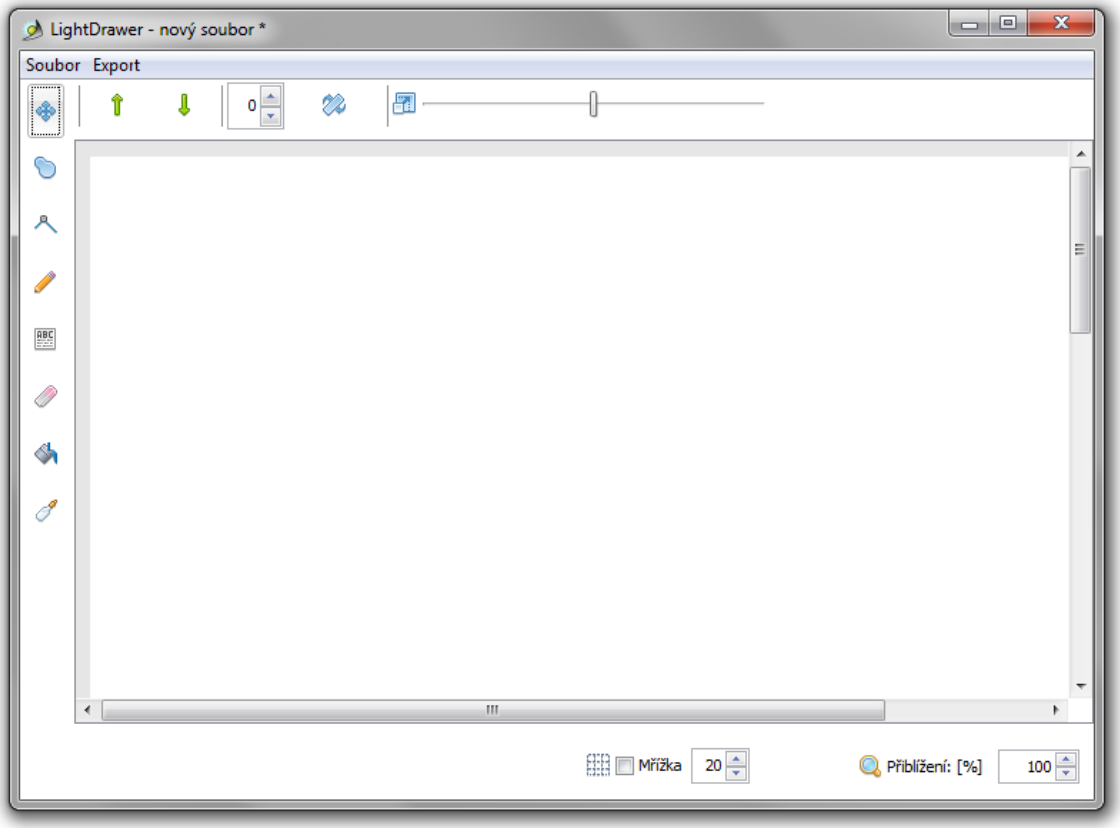
Jádro struktury vektorového editoru můžeme vidět v zjednodušeném diagramu tříd na obrázku 7.1. Kompletní diagram tříd je obsažen v příloze A.



Obrázek 7.1: Zjednodušený UML diagram tříd programu

7.2 Hlavní okno programu

Veškeré grafické uživatelské rozhraní bylo navrhnuto pomocí návrháře vývojového prostředí *Netbeans*. Takto vytvořené hlavní okno programu můžeme vidět na obrázku 7.2. *Netbeans* generuje vlastní kód za pomocí uchovávaného XML dokumentu, který nelze přímo měnit. To způsobilo drobné problémy například s typem komponenty `DrawPanel`, komponentou grafického uživatelského rozhraní, která je třídou dědící od třídy `SVGPanel` – třídou s grafickým kontextem, poskytovanou knihovnou SVG Salamander, vykreslující celý diagram SVG do svého obsahu. Přetypování bylo ale možné řešit v konstruktoru hlavního okna aplikace hned po metodě vygenerované prostředím *Netbeans*. Výhoda použití tohoto vývojového prostředí spočívala především v uživatelské přívětivosti, nastavení jednotlivých komponent grafického rozhraní bylo možno nastavit přímo v editoru a *Netbeans* navíc podporuje rozvržení komponent, které není podmíněno specifikací layoutu rodičovských komponent.



Obrázek 7.2: Hlavní okno programu

Hlavní okno programu `MainFrame` zajišťuje především komunikaci s uživatelem, nastavuje zde různé parametry kreslení a také uchovává instance všech použitých nástrojů a jejich správce. Tyto nástroje uchovává právě hlavní okno z toho důvodu, aby bylo možno mezi nimi přepínat odchytáváním událostí při klikání na tlačítka. Okno si tak zajišťuje například i správnou viditelnost jednotlivých komponent, které nástroje

potřebují, ale také označení používaného nástroje nebo vzhled kurzoru nástroje na použitém `DrawPanelu`. Třída hlavního okna `MainFrame` tedy zajišťuje většinu komunikace s uživatelem (zbytek zajišťují okna dialogů vyvolaných určitou uživatelskou akcí), které přímo nesouvisí s ovlivňováním grafického kontextu komponenty `DrawPanel` – tedy samotnou kresbou.

7.3 Implementace funkcionalit

Implementaci editoru lze díky takto navržené struktuře rozdělit do jednotlivých funkcionalit, které editor poskytuje. Vzhledem k tomu, že se jedná o grafický editor a proto je grafické uživatelské rozhraní jeho nedílnou součástí, jsou tyto funkcionality často součástí více tříd programu, zejména pak často komunikují s grafickým rozhraním a přímo jej ovlivňují. Popis jednotlivých funkcionalit tedy nabízí větší přehlednost. V této části tudíž budou popsány hlavní myšlenky a způsoby implementace těchto funkcionalit.

7.3.1 Komponenta obsahující kresbu

Komponenta s kresbou je stěžejní částí programu. Je potomkem třídy `SVGPanel`, jež je potomkem třídy `JPanel`, a tudíž je s ní možné zacházet jako s komponentou grafického uživatelského rozhraní. Její metoda `paint()` automaticky vykresluje `SVGDiagram` do svého obsahu. Třída použitá v programu je ale rozšířena o několik možností nezbytných pro chod programu. Obsahuje informace o:

- otevřeném souboru
- jeho aktuálním stavu
- jeho velikosti
- přiblížení (implementace přibližování obsahu je podrobně popsána níže)
- implicitní šířka čtverce mřížky a je–li mřížka zapnutá
- konstanty jako vynechaný prostor okolo komponenty nebo rychlost rolování

Kromě vykreslování diagramu otevřeného SVG dokumentu umožňuje vykreslování tvarů, určených pro statické vyznačení, které mohou nástroje potřebovat, jako je například vyznačení objektů, se kterými pracuje nástroj pro transformaci. Rovněž vykresluje jediný tvar určený jako pomocný tvar pro některé nástroje, příkladem může být označování více objektů najednou, které vyžaduje vykreslení obdélníka. Tato komponenta se stará i o vykreslování mřížky pro přesnější práci, avšak její funkcionalita je implementována jinde.

Komponenta také přímo pracuje se základními třídami knihovny SVG Salamander, uchovává referenci na instanci třídy `SVGUniverse` a samozřejmě instanci třídy

SVGDiagram. Rovněž zajišťuje načtení svého obsahu se souboru na dané URL adrese. Také umožňuje vytvořit zcela novou kresbu o zadaných rozměrech v milimetrech. Rozměry v těchto jednotkách však zpracovává SVG Salamander z vytvořeného řetězce v souboru a proto není možné s nimi dále pracovat. Velikost kresby je dána velikostí komponenty s ní. Tato komponenta také obsahuje několik metod pro aktualizaci kresby a překreslování související grafiky, které jsou využívány především jednotlivými nástroji při změnách v kresbě.

7.3.2 Správce nástrojů

Další součástí důležitou pro chod programu je třída ToolManager. Ta zajišťuje správu nástrojů a jejich správnou funkci. Také se stará o vyhodnocování vstupů od uživatele, zejména pak těch vstupů, co se týkají samotné komponenty DrawPanel. Správce nástrojů kresbě registruje naslouchače potřebné k vyhodnocování vstupů od uživatele.

Správce nástrojů také obsahuje celou řadu metod, které nástroje mohou vyžadovat, jako je například možnost vybrat jeden element z kresby na určitých souřadnicích – pokud je na těchto souřadnicích elementů více, je vrácen ten, který překrývá všechny ostatní (tedy ten, který je z těchto objektů vykreslen poslední). Správce nástrojů také umožňuje aplikovat na objekty různé transformace prostřednictvím třídy AffineTransform. Dále poskytuje nástrojům různé obecné informace. Pokud však nástroj potřebuje informace rozsáhlejší, ToolManager může předat i celou kresbu. Implementuje tedy především základní metody pro použití v nástrojích, avšak není navržen pro obsáhnutí všech jejich potřeb.

7.3.3 Nástrojový naslouchač

Implementace nástrojového naslouchače ToolListener implementuje hned několik rozhraní, například rozhraní MouseListener, MouseWheelListener, MouseMotionListener nebo KeyListener. Všechny vstupy dostupné těmto rozhraním a potřebné pro chod programu jsou tedy snoubeny v jediné třídě, která je implementuje. Zatímco události týkající se stisku kláves pouze mění pravdivostní hodnotu v ToolManageru, protože tyto klávesy jsou používány výhradně při akcích jako je kreslení pravoúhlých lomených čar a tudíž by měly být dostupné v každém okamžiku spuštění jiné události, události týkající se myši jsou složitější. Vzhledem k tomu, že jednotlivé nástroje pracují právě s událostmi pocházejícími z myši, jsou navrženy tak, že musí implementovat interface Tool. Pomocí metod v tomto interface může ToolListener přeposlat MouseEvent právě aktivnímu nástroji, který se pak podle toho zachová dle svého účelu a implementace.

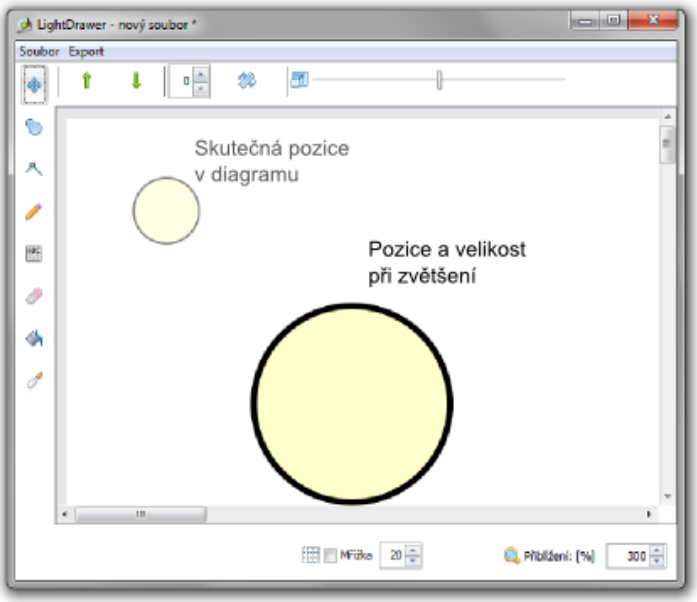
Existují však situace, kdy je nutné tuto událost poupravit. V současné implementaci editoru jsou to situace, kdy:

- je aplikováno zvětšení/zmenšení kresby – v takovém případě je potřeba přepočítat souřadnice tak, aby odpovídaly situaci bez zvětšení, SVG diagram jako takový se totiž při změně velikosti nemění a měnit se nemůže, škáluje se pouze grafický kontext, v němž je vykreslen
- je použita mřížka – souřadnice je třeba zaokrouhlit tak, aby se nacházely v levém horním rohu čtverce, v němž událost nastala, tato funkce je implementována staticky, aby byla použitelná i pro jednotlivé nástroje

`ToolListener` také upravuje některé události tak, aby vyhovovaly potřebám interface `Tool`, například při kliknutí rozlišuje mezi kliknutím levým a pravým tlačítkem myši.

7.3.4 Přibližování

Jak již bylo zmíněno v kapitole o externích knihovnách, použití knihovny SVG Salamander namísto knihovny Batik s sebou nese jisté nevýhody. Jednou z nich je nepřítomnost nástroje, který by umožňoval zvětšování či zmenšování obrazu. Pro vektorový editor je tedy potřeba tuto funkci doimplementovat. Vzhledem k tomu, že stránka dokumentu je reprezentována komponentou, co je potomkem třídy `JPanel`, je možné měnit její rozměry. Jejich změnu si pak lze snadno dopočítat z nastavené hodnoty přiblížení, které je v nynější verzi editoru dáno procentuální hodnotou, podobně jako ve vektorovém editoru *Inkscape*.



Obrázek 7.3: Problém domnělé a skutečné pozice elementu při trojnásobném přiblížení

Zvětšení rozměrů `JPanelu` však nezmění vzhled jeho grafického kontextu, proto je nutné překrytím metody `paint()` nastavit, aby proběhlo škálování grafického kontextu panelu podle nastaveného přiblížení. Změna velikosti ale přináší i další komplikaci, zatímco je zobrazení větší, samotný diagram SVG zůstává stejný. Tento problém je naznačen na obrázku 7.3. To je řešeno přepočítáváním souřadnic při kliknutí na hodnoty, kde by se kurzor v kresbě nacházel, kdyby nebylo přiblížení nebo oddálení vůbec aplikováno. To zajišťuje obecně správnou funkci editoru za jakéhokoli přiblížení.

7.3.5 Mřížka

Mřížka je velmi praktickou součástí editoru, neboť umožňuje přesné kreslení. Pokud je zapnuta, je vykreslena jednoduchým algoritmem do grafického kontextu `DrawPanelu` a správci nástrojů je předána informace o tom, že je mřížka zapnuta. Ten ji pak posléze může předat kterémukoli nástroji, který si ji vyžádá společně s parametrem udávajícím šířku čtverce mřížky (tj. vzdálenosti mezi jejími rovnoběžnými čarami). Nulová hodnota pak znamená, že mřížka není použita. Vzhledem k tomu, že u všech nástrojů není žádoucí, aby se zaokrouhlovaly souřadnice kurzoru při jejich použití (například u nástroje guma), je nutné, aby si nástroje, u kterých použití mřížky žádoucí je, samy vyžádaly informaci o jejím použití a případně využily možnosti zaokrouhlení, kterou jim poskytuje `ToolListener`. Díky tomu je možné, aby byla mřížka respektována nejen pouze vybranými nástroji, ale případně i při určité rozdílné činnosti těchto nástrojů.

7.3.6 Nástrojový interface

Jak již bylo řečeno výše, všechny použité nástroje musí implementovat rozhraní `Tool`, obsahující především metody pro obsluhu jednotlivých událostí přicházejících z listeneru `ToolListener`, tedy především události týkající se činnosti myši. Mimo ně však může každý nástroj obdržet referenci na instanci svého správce, kterou si může a nemusí uložit. Tím případně získá přístup k funkcím a informacím, které správce poskytuje. Podle návrhu také obsahuje metodu, která umožňuje nástroji aplikovat své funkce na právě označené objekty, tato metoda však nebyla v současné implementaci použita.

7.3.7 Nástroj transformace

Tento nástroj, jak název napovídá, slouží pro aplikaci transformací na element nebo i více elementů najednou. Pomocí nástroje lze jednotlivé objekty označit. K tomu je využita služba poskytovaná správcem nástrojů. Při stisknutí levého tlačítka myši a tažení myší je do komponenty `DrawPanel` odesílán obdélník, který se následně vykresluje. Při události puštění tlačítka je zjištěno, zda bylo puštěno kvůli klepnutí na jediný objekt nebo při ukončování označení více objektů. Ve druhém případě jsou

pak procházeny všechny objekty v dokumentu a je zjišťováno, zda se nacházejí v nakresleném obdélníku. Pokud ano, jsou zařazeny do spojového seznamu označených objektů. Pro všechny označené objekty je pak vygenerován tvar pro jeho označení, získaný za pomoci metody `getBoundingBox()` knihovny SVG Salamander. Všechny tyto tvary jsou pak odeslány k vykreslení opět komponentě `DrawPanel`.

Označené objekty, popřípadě jediný objekt, je možné přesouvat. Toho je docíleno tím, že je na ně aplikována transformace translace. Ta je na tyto objekty aplikována pomocí správce nástrojů prostřednictvím třídy `AffineTransform`. Způsob transformace spočívá ve vytvoření nové, prázdné transformace, na kterou je posléze aplikována translace. Tato transformace je pak správcem nástrojů spojena s původní transformací objektu. Tento způsob se ukázal být mnohem vhodnější než změna atributů jednotlivých objektů, neboť atributy pro pozici elementu mívají jiné názvy i chování pro jednotlivé typy elementů.

U označeného objektu nebo objektů je také možné ovlivnit, který objekt překrývá jiný objekt, neboli který je „nad“ a který „pod“ jiným. V SVG dokumentu a tedy i diagramu je toto ovlivněno tím, v jakém pořadí se elementy (v dokumentu jejich tagy, v diagramu jejich pořadí ve stromu) nacházejí. Jelikož knihovna SVG Salamander umožňuje rodičovskému prvku získat a prohodit libovolné dva své potomky, a protože každý element, který je možno vykreslit, má nějakého rodiče, je implementace poměrně jednoduchá. Pro každý označený element je nalezen rodič a je zjištěna jeho pozice mezi potomky svého rodiče. Poté může být element prohozen buď s předcházejícím nebo následujícím potomkem. Bylo však nutno ošetřit případy, kdy je element na začátku nebo na konci v seznamu potomků.

Nástroj umožňuje označené objekty také škálovat nebo je rotovat. Způsob provedení je obdobný jako u translací, avšak zde je třeba si uvědomit, že záleží na pořadí transformací. Také je zapotřebí dopočítat translaci tak, aby se objekty chovaly způsobem, který uživatel očekává, například aby při škálování neměnily svou pozici. Pro rotaci lze pak nastavit střed rotace přímo za pomoci knihovny SVG Salamander. Tato knihovna také usnadňuje získání tohoto středu poskytováním obdélníka, který obsahuje element a přitom má co nejmenší velikost.

7.3.8 Nástroj kreslení tvarů

Při implementaci nástroje kreslícího tvary bylo důležité zajistit vyhodnocování vstupních souřadnic od myši tak, aby se program choval podle očekávání uživatele. Bylo zapotřebí zjistit, který z bodů je počáteční a který koncový, u tvarů jako jsou kružnice a elipsy bylo pak potřeba analyzovat rozměry vznikající porovnáním těchto dvou bodů. Podle toho bylo také nutné vykreslit pomocný tvar, podle kterého by se uživatel mohl orientovat. Vzhledem k odlišnosti specifikace některých tvarů – například

kružnic – v SVG dokumentu a v grafickém kontextu, bylo zapotřebí přepočítat souřadnice tak, aby odpovídaly objektu, který bude skutečně nakreslen do diagramu. Implementace se tedy liší u každého tvaru.

Po stanovení tvaru, jeho pozice a rozměru, bylo nutné zohlednit všechna nastavení z grafického uživatelského rozhraní, která se týkají barvy výplně, barvy okraje nebo jeho tloušťky. Také je třeba zohlednit nastavení toho, zda se má výplň nebo okraj vykreslit. K tomu slouží pomocná třída podle návrhového vzoru Slave – `Painter`. Ta vyhodnotí všechna tato nastavení a podle nich nastaví parametry kresleného objektu v diagramu, čímž zajistí jeho správné vykreslení.

7.3.9 Nástroj kreslení lomené čáry

Tento nástroj neslouží pouze ke kreslení čar, ale obsahuje stejná nastavení pro vykreslení výplně, šířku okraje (čáry) a způsob vykreslení. Implementace je obdobná jako u kreslení tvarů, s tím rozdílem, že obsluha vstupů od uživatele probíhá poněkud odlišně, aby byl program intuitivnější. Místo informacích o tvaru je zde pak uchovávána informace o čáře, kde je nutno zajistit správné vygenerování korektního řetězce pro parametr cesty v dokumentu SVG. Algoritmus pro generování tohoto řetězce je vyobrazen na obrázku 7.4. Navíc tento nástroj při kreslení čáry testuje stisknutí klávesy *Shift*. Pokud je klávesa stisknuta, zkoumá aktuální souřadnice kurzoru a porovnává je s posledním bodem čáry. Ta souřadnice, jejíž velikost rozdílu s toutéž souřadnicí posledního bodu je menší, je zanedbána a nahrazena souřadnicí posledního bodu, což má za následek kreslení pouze svislých nebo vodorovných čar. Nástroj lomené čáry také obsahuje speciální nastavení, vyplývající z vlastností SVG – dokončování. Jediný rozdíl je v řetězci pro parametr cesty, který je ukončen jiným znakem, v implementaci tedy postačila podmínka.

```
if (start == null) {
        return;
}
String dattr = "M" + start.getX() + " " + start.getY();
for (Point point : points) {
        dattr = dattr + " L" + point.getX() + " " + point.getY();
}
if (manager.haveToEndPath()) {
        dattr = dattr + " Z";
}
```

Obrázek 7.4: Kód sloužící k vygenerování řetězce pro uchovávání cesty dle norem SVG

7.3.10 Nástroj tužka

Tento nástroj je implementačně prakticky totožný s nástrojem pro kresbu lomených čar, pouze jiným způsobem vyhodnocuje vstupy od uživatele. Nevyhodnocuje stisknuté klávesy a reaguje pouze na stisknutí tlačítka, jeho uvolnění a na tah myši. Další body

pak přidává při tahu myši během stisknutého tlačítka, místo aby jako v případě nástroje pro lomenou čáru přidával body při kliknutí myší.

7.3.11 Nástroj vkládání textu

Při kliknutí tímto nástrojem na zvolené místo v komponentě `DrawPanel` tento nástroj vytvoří a zobrazí okno `TextInsertFrame`, kterému v konstruktoru předá dotyčný `MouseEvent` a související `DrawPanel`. V tomto okně je použita externí komponenta `JFontChooser`, která vytváří formulář s dostupnými systémovými písmy, jejich nastavením a polem pro text. Také podporuje potřebné lokalizace. Tato komponenta byla pro účely editoru drobně poupravena, zejména kvůli popisu jednotlivých nastavení a rozšíření informací, které je z komponenty možno získat. Díky tomu je v metodě pro obsluhu události, nastávající při potvrzení dialogu, získat všechna potřebná nastavení písma. Ta jsou pak v atributech předána nově vytvořenému objektu `Text` z knihovny SVG Salamander a poté je tento objekt vložen do SVG diagramu. V současné implementaci vektorového editoru však není možné tento text zpětně editovat.

7.3.12 Nástroj guma

Vzhledem k rozdílu v povaze vektorových a rastrových formátů nemůže být tento nástroj implementován podobně jako například v *Malování*, slouží však k mazání celých objektů z diagramu. Po zachycení události kliknutí nalezne za pomoci správce nástrojů `ToolManager` element na souřadnicích kliknutí a odebere tento element z diagramu. Pokud žádný element nenalezne, neprovede nic. Po odebrání elementu zavolá překreslení komponenty `DrawPanel`, čímž aktualizuje zobrazení diagramu `SVGDiagram`.

7.3.13 Nástroj plechovka

I tento nástroj pracuje ve vektorovém editoru jinak než v jeho rastrových protějšcích. Je takto nazván aby uživatel mohl alespoň částečně předpokládat jeho funkci. Slouží k upravování již nastavených parametrů jednotlivých elementů podle aktuálního nastavení v grafickém uživatelském rozhraní, tedy výplně, ale i barvy okraje a jeho šířky. Na rozdíl od jiných nástrojů však nemůže využít pomocnou třídu `Painter`, neboť tato třída předpokládá aplikaci na nový element bez atributů. Plechovka je však aplikována na element již nakreslený, popřípadě i načtený ze SVG dokumentu vytvořeném v jiném programu. Z toho důvodu musí ověřovat existenci parametrů ve stylech pro daný element. Teprve poté může tato nastavení přepsat nebo vytvořit nová, protože knihovna SVG Salamander ošetřuje tento stav pouze formou výjimek. Nová nastavení pak plechovka ukládá výhradně do stylů elementu, aby měly v případě zbývajících atributů přednost při vykreslování. Atributy se stejným typem nastavení ponechává z tohoto důvodu bez povšimnutí.

7.3.14 Nástroj kapátko

Kapátko slouží jako nástroj pro nastavení barev podle zvoleného elementu. Pro stisknutí tlačítka myši a následnému odchycení souřadnic nalezne za pomoci správce nástrojů `ToolManager` element, jehož se kliknutí týká, a zkoumá jeho vlastnosti. Nejdříve se vždy zajímá o styly elementu kvůli jejich upřednostnění při vykreslování, pokud žádné pro danou vlastnost nenajde, hledá je v atributech. Nalezené barvy pak přemění do formátu barev Javy, třídy `Color`, a tuto barvu pak nastaví příslušným tlačítkům pro barvy v grafickém uživatelském rozhraní, jejich parametr barvy slouží v editoru jako proměnná pro informace o barvách.

7.3.15 Vytváření souborů

O vytváření nových kreseb se stará přímo komponenta `DrawPanel`. Vytváří řetězec reprezentující prázdný SVG dokument, v němž jsou parametry *width* a *height* nastaveny dle daných rozměrů. O vytvoření vnitřního diagramu se pak postará sama knihovna SVG Salamander prostřednictvím třídy `SVGUniverse`. Po načtení souboru lze pak získat jeho diagram, který si komponenta uloží a dále s ním pracuje při vykreslování a jeho úpravách. Pokud byl předchozí dokument měněn, je před samotným vytvářením nového souboru uživatel upozorněn dialogem `SaveRecomendForm`.

7.3.16 Načítání souborů

O načtení souboru do diagramu a tedy i grafického kontextu se rovněž stará komponenta `DrawPanel` a to obdobným způsobem jako při vytváření nového. Před jejím použitím je rovněž zkontrolována změna předchozí neuložené kresby a uživatel je popřípadě upozorněn. Poté je využito nativního dialogu Javy `JFileChooser`, která uživateli umožňuje procházet soubory na disku a programu poté předává URL vybraného souboru. Tato URL je pak předána komponentě `DrawPanel` a použita pro načtení souboru.

7.3.17 Ukládání

Pro ukládání a export je určena třída `Exporter`, která využívá externí knihovnu FreeHEP VectorGraphics. Při běžném ukládání je nejprve ověřeno, zda již byla vybrána URL adresa pro uložení a při kladném výsledku tuto adresu použije. Pokud vybrána nebyla, použije tu, kterou uživatel zadá v otevřeném dialogu `JFileChooser`. Uživatel má také možnost tento dialog otevřít bez ohledu na to, zda byla adresa již jednou zadána. Po předání URL adresy třídě `Exporter` je vytvořen `FileOutputStream`, kde je tato adresa parametrem. Poté jsou přidány implicitní parametry pro třídu `SVGGraphics2D`, která je vytvořena záhy. Je také velice důležité, aby byl u této třídy nastaven parametr stanovující, že nebude závislá na zařízení. Tím je

možné vyhnout se problémům s lokalizací a českou diakritikou například ve slovním vyjádření data vytvoření, se kterou tvůrci knihovny patrně nepočítali. To je také jeden z důvodů, proč se veškeré texty exportují jako křivky a proto není možné je zpětně editovat. To však s sebou nese nezávislost na přítomnosti použitého typu písma v operačním systému. Poté je knihovna FreeHEP VectorGraphics využita způsobem popsaným v teoretické části – do grafického kontextu SVGGraphics2D je vykreslen `SVGDiagram`. Po ukončení exportu je vhodné zavřít `FileOutputStream`.

7.3.18 Export

Takzvaný „Rychlý export“ použitý v editoru a sloužící uživateli jako dostupnější volba exportu do rozšířených vektorových formátů, rovněž zajišťuje třída `Exporter` způsobem technicky téměř stejným jako v případě ukládání, rozdíl spočívá pouze v použití specializovaných kontextů pro vykreslení do daného formátu. Pro všechny exporty i ukládání je pak nastaven `LightFileFilter`, což je potomek třídy `FileFilter` pro účely editoru, který umožňuje filtrování druhů souborů, v tomto případě podle jejich názvu, konkrétně přípony.

Možnost exportu do všech formátů dostupných knihovně FreeHEP VectorGraphics je pak realizována použitím komplexního dialogu pro export, který tato knihovna nabízí. Před jeho vytvořením je přiblížení nastaveno na hodnotu 1.0, tedy na skutečnou velikost, neboť zmíněnému dialogu se v parametrech předává komponenta, v tomto případě `DrawPanel`, jejíž grafický kontext je přesně exportován včetně rozměrů. Dialog pak nabídne uživateli dostupná nastavení a volby. Zároveň také znemožní uživateli přístup k editoru, dokud je dialog aktivní. Teprve až po jeho zrušení či potvrzení (a následnému vytvoření souboru na disku v požadovaném formátu) je možné opět pracovat s editorem.

7.3.19 Nastavení DPI

Tato funkcionalita neimplementuje DPI v pravém smyslu, prakticky pouze upravuje přiblížení (tedy velikost) kresby v závislosti na rozměrech a rozlišení zvolených uživatelem. Tato možnost byla přidána z důvodu absence možnosti volby tohoto parametru při exportu do rastrových formátů. Vytvořené okno nejprve zruší přiblížení, aby bylo možné snáze pracovat s jeho skutečnými rozměry. Pracuje také s poměrem jeho stran, aby uživatel nemohl nastavením narušit vzhled kresby. Po potvrzení uživatelem vypočítá z hodnot požadované velikosti obrázku při daném DPI požadované rozměry, z kterých spočte potřebnou hodnotu zvětšení. Tu pak aplikuje na kresbu za pomoci nástroje pro zvětšování. Poté ihned vyvolá dialog pro export poskytovaný knihovnou FreeHEP VectorGraphics, kterým je možné toto zobrazení vyexportovat.

8 Testování

V této kapitole se budeme zabývat testováním funkcí již hotového vektorového editoru, pojmenovaného *LightDrawer*. Budou zde testovány jednotlivé funkcionality a klíčové situace, kterým by uživatel mohl být vystaven. Vzhledem ke grafické povaze programu bude testování prováděno především uživatelsky a správnost vstupů a výstupů bude kontrolována vizuálně.

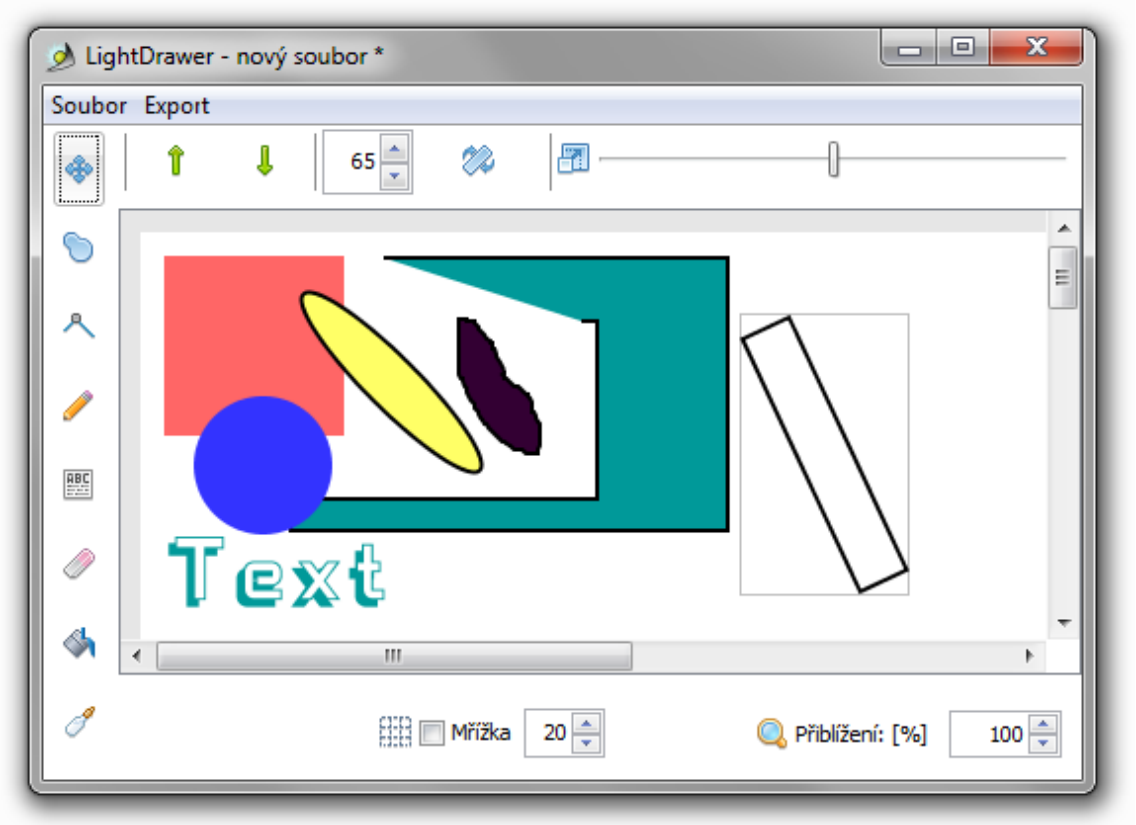
8.1 Možnosti kreslení

V této části byla testována funkčnost z hlediska možnosti kreslení, testovalo se tedy, co je možné nakreslit nebo upravit. Také se zde budeme zabývat ukládáním a načítáním obsahu a následnou prací s ním.

8.1.1 Kreslení pouze v editoru

V editoru lze pracovat s několika druhy grafických primitiv. Patří mezi ně čtverec, obdélník, kruh, křivka (čára) a text. Tyto prvky jsou rovněž základními prvky většiny vektorových formátů a především formátu SVG, který editor považuje za svůj základní formát, neboť pouze z něj dokáže načítat obsah. Kromě kreslení primitiv vektorový editor *LightDrawer* dokáže těmto primitivům měnit parametry pro barvy výplně a okraje. Také dokáže každému objektu včetně textu měnit šířku tohoto okraje. Editor také může prohazovat pořadí jednotlivých prvků, tedy posouvat libovolný prvek „nad“ nebo „pod“ jiný, avšak toto prohazování probíhá v rámci všech prvků v dokumentu a z toho důvodu bývá poněkud těžkopádné.

Editor také zvládá aplikaci několika základních transformací grafických primitiv, přičemž je možné označit více objektů najednou a tak aplikovat stejnou transformaci na více objektů. Dokáže objekty rotovat o libovolný počet stupňů, avšak pouze podle jejich vlastního středu. Objekty je také možné přesouvat. Také je umožněno škálování objektů, které však nenabízí žádnou orientaci co se týče přesnosti (objekty například není možné zvětšit přesně na dvojnásobek velikosti). Tímto škálováním je možné zvětšit nebo zmenšit objekt na takřka libovolnou velikost. Patrně se zde ale projevují nějaké nepřesnosti při přepočítání translace, neboť škálovanému objektu se mírně mění pozice. Transformované objekty pak občas vykazují známky podivného chování například při přesunu nebo označení, kdy jsou jejich označení – a tedy i prostor, na který lze kliknout, aby byl element označen – zbytečně velké. Toto chování ale zpravidla bývá pouze obtěžující, práci s grafikou přímo neznemožňuje. V editoru také nelze označit element, který je zcela překrytý jiným, aniž by bylo nutné překrývající objekt odsunout. Ukázku základní práce s editorem můžeme vidět na obrázku 8.1.



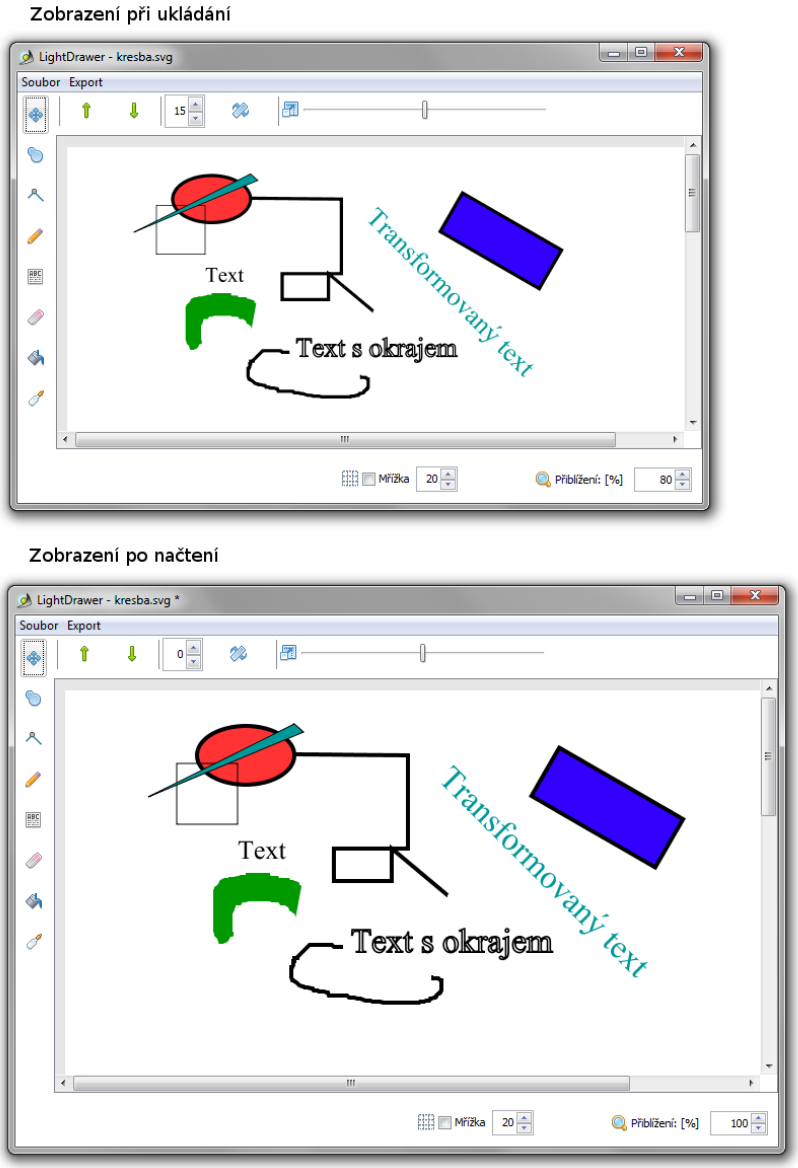
Obrázek 8.1: Příklad práce s editorem LightDrawer

Co se týče ostatních funkcí editoru, které usnadňují zacházení s ním, jako je například mazání vektorových primitiv, získávání jejich barvy či popřípadě změna této barvy, fungují očekávatelným způsobem na všechny druhy objektů, které lze v editoru vytvořit.

Jak můžeme vidět na obrázku 8.2, kde je znázorněno zobrazení po uložení a poté po zpětném načtení, editor při ukládání nijak nemění grafický obsah a dokáže jej bez chyb a rozdílů v zobrazení znovu načíst a pracovat s ním. V příkladu jsou použita různá přiblížení obsahu, aby bylo demonstrováno, že oddálení při ukládání nemělo vliv na dokument (a tudíž jsou zobrazení kresby v příkladu různě velká). Vidíme tedy, že v zobrazení rozdíl nenastává. Co se však týče práce s takto načteným souborem, došlo k několika nepříznivým jevům:

- z okrajů jednotlivých elementů se staly samostatné elementy
- kontejnery jednotlivých objektů se začaly objevovat na špatných místech a u některých elementů i o špatných rozměrech
- chování aplikace transformací se změnilo

Tyto chyby jsou pravděpodobně způsobeny vnitřní implementací použitých knihoven, neboť nalezení kontejnerů objektů a načítání do diagramu zajišťuje knihovna SVG Salamander, zatímco ukládání je prováděno přímo z grafického kontextu knihovou FreeHEP Vector Graphics. Tato knihovna neobsahuje způsob, jakým uložit informace, jako jsou například transformační matice obsažené v diagramu. Knihovna SVG Salamander pak při načítání tyto informace postrádá, čímž vznikají výše zmíněné chyby. Obsah je však zobrazitelný a upravitelný ve vektorovém editoru *Inkscape* i webovém prohlížeči *Google Chrome*. V obou těchto programech se pak zobrazil stejně jako v *LightDraweru* a bylo s nimi možné pracovat. Dochází zde pouze k chybám způsobeným výstupní knihovnou.

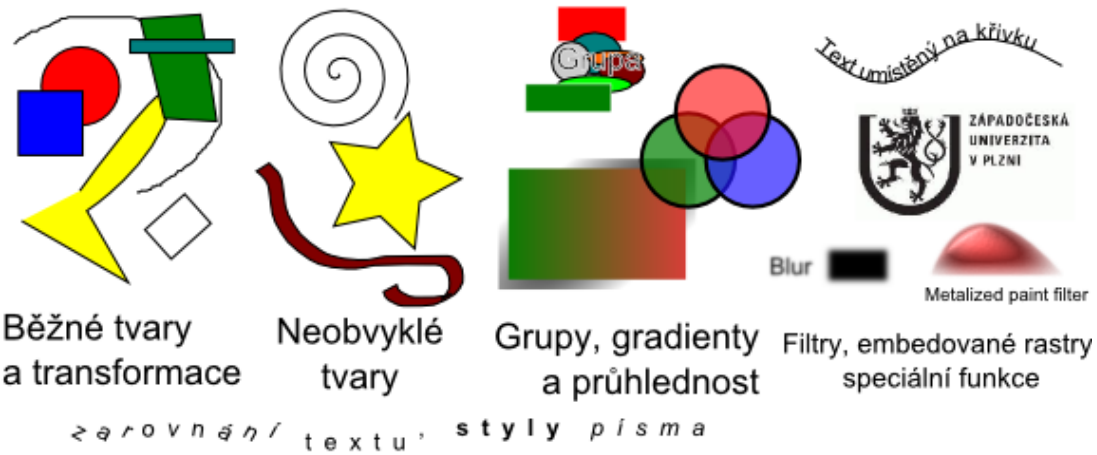


Obrázek 8.2: Srovnání vzhledu kresby před uložením a po zpětném načtení

8.1.2 Použití jiných editorů

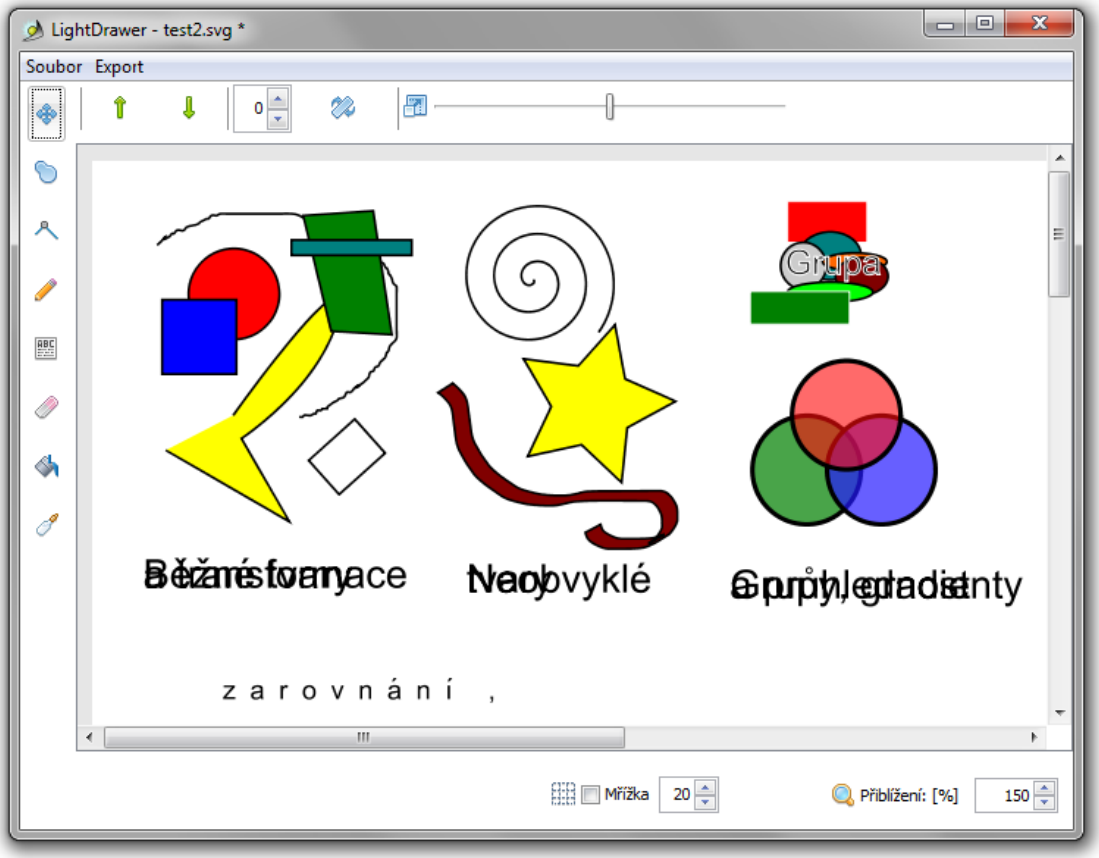
V této sekci se budeme zabývat tím, jak se obsah chová, je–li načten z jiného editoru, popřípadě jak se obsah z LightDraweru chová v jiných editorech. K tomuto účelu byl použit rozšířený vektorový editor *Inkscape*, především z důvodu jeho dostupnosti a práci s formátem SVG. SVG umožňuje použití velkého množství vlastností elementů, které *LightDrawer* nepodporuje. Jedná se například o grupy, speciální tvary, barevné gradienty nebo průhlednost barev. Možnosti knihovny pro načítání SVG Salamander jsou však také obsáhlejší než možnosti editoru, stejně tak možnosti knihovny pro export a ukládání FreeHEP VectorGraphics. Editor *LightDrawer* byl testován tak, že byl vytvořen testovací SVG dokument, který obsahoval jak prvky, které je možné vytvořit v *LightDraweru*, tak i některé z prvků, se kterými jeho rozsah implementace nepočítal. Ten byl pak v *LightDraweru* otevřen, byla otestována možnost změny grafických primitiv a opět uložen a zobrazen v *Inkscape*. Také bylo vyzkoušeno, zda má na funkčnost vliv velikost otevíraného souboru. Velikost souboru ovlivňuje rychlost načítání i práce s kresbou, ale na funkcionality nemá vliv.

Původní testovací SVG dokument, který můžeme vidět na obrázku 8.3, obsahující možnosti SVG, které knihovna SVG Salamander nepodporuje, nemohl být načten. Program sám však ve většině případů dále fungoval. Testovací soubor byl tedy zjednodušen odebráním pravé části obsahující text umístěný na křivku, filtry a rastrový obrázek. Posléze také bylo zapotřebí odebrat z dokumentu i barevný gradient, který také způsoboval, že program nebyl schopen vykreslit tento dokument. Důvodem je patrně to, že knihovna SVG Salamander barevné gradienty nepodporuje. Souhrnně se tedy dá říci, že program vlivem vlastností této knihovny nedokáže otevřít soubor obsahující ty vlastnosti SVG, které knihovna SVG Salamander neimplementuje.



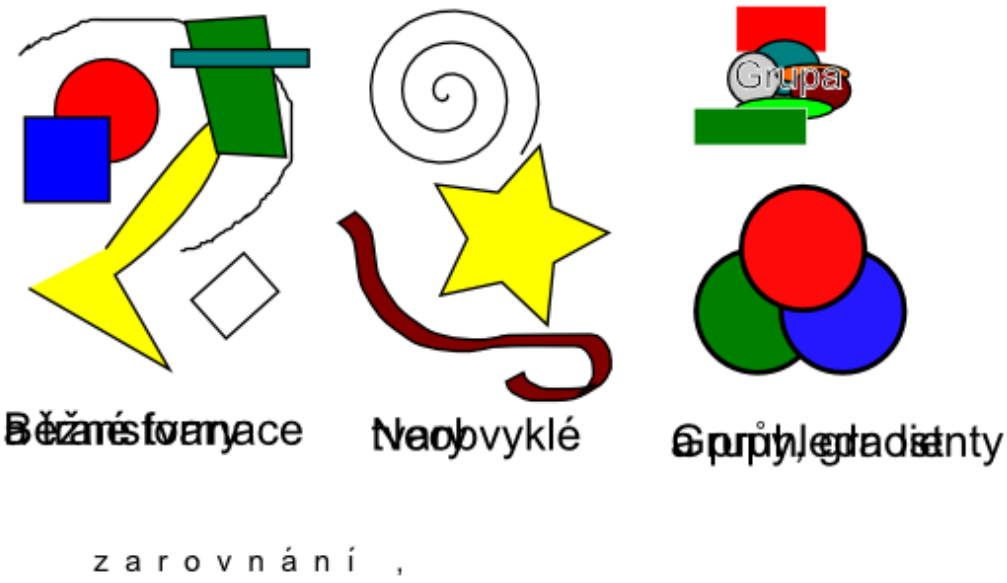
Obrázek 8.3: Původní SVG dokument určený k testování (otevřeno v Inkscape)

Jak můžeme vidět na obrázku 8.4, který vyobrazuje zjednodušený testovací soubor otevřený v testovaném editoru *LightDrawer*, program otevřel dokument vytvořený v jiném editoru a zobrazil správně všechny tvary, grupu i průhlednost. Vidíme zde také to, že program nepodporuje zarovnání textu a většinu kombinovaných stylů písma, avšak například větší odsazení písmen zpracovat dokáže. Práce s takto načtenými objekty se příliš neodlišovala od práce s objekty načtenými ze souboru, který byl vytvořen samotným testovaným editorem. Výjimkou byla práce s grupou, kde při změnách vrstev (přesunu objektů „nad“ nebo „pod“) docházelo k tomu, že mohla být měněna pozice objektů v grupě vůči sobě navzájem i pozice ostatních objektů vůči celé grupě, avšak nebylo možné zařadit některý objekt mezi objekty v grupě.



Obrázek 8.4: Testovací dokument SVG otevřený v LightDraweru

Na obrázku 8.5 můžeme vidět, jak byl dokument změněn po uložení testovaným programem *LightDrawer*. Na obrázku je jeho zobrazení vektorovým editorem *Inkscape*. Můžeme si povšimnout, že výstup byl změněn pouze o část nepodporovanou knihovnou zpracovávající výstup, a to o průhlednost, kde však byla pouze zanedbána hodnota alfa kanálu. V *Inkscape* bylo pak možné soubor editovat běžným způsobem.



Obrázek 8.5: Zobrazení po uložení pomocí testovaného editoru

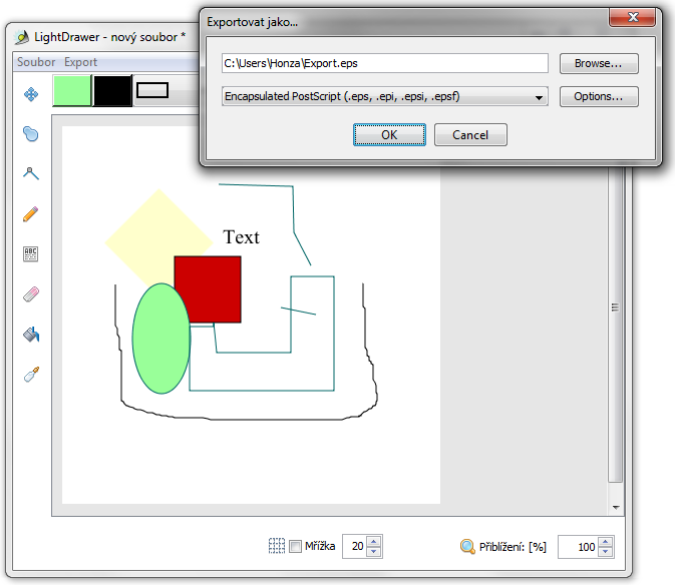
8.2 Export

V této části se budeme zabývat kvalitou exportu do vektorových a rastrových formátů nabízených testovaným editorem. Vzhledem k tomu, že tyto formáty nejsou určeny k tomu, aby byly v editoru dále upravitelné, budeme se zajímat pouze o vizuální kontrolu kvality a srovnání výstupního souboru s grafikou vyobrazenou v editoru. Vzhledem k množství dostupných formátů zde budou uvedeny jen ty nejpoužívanější.

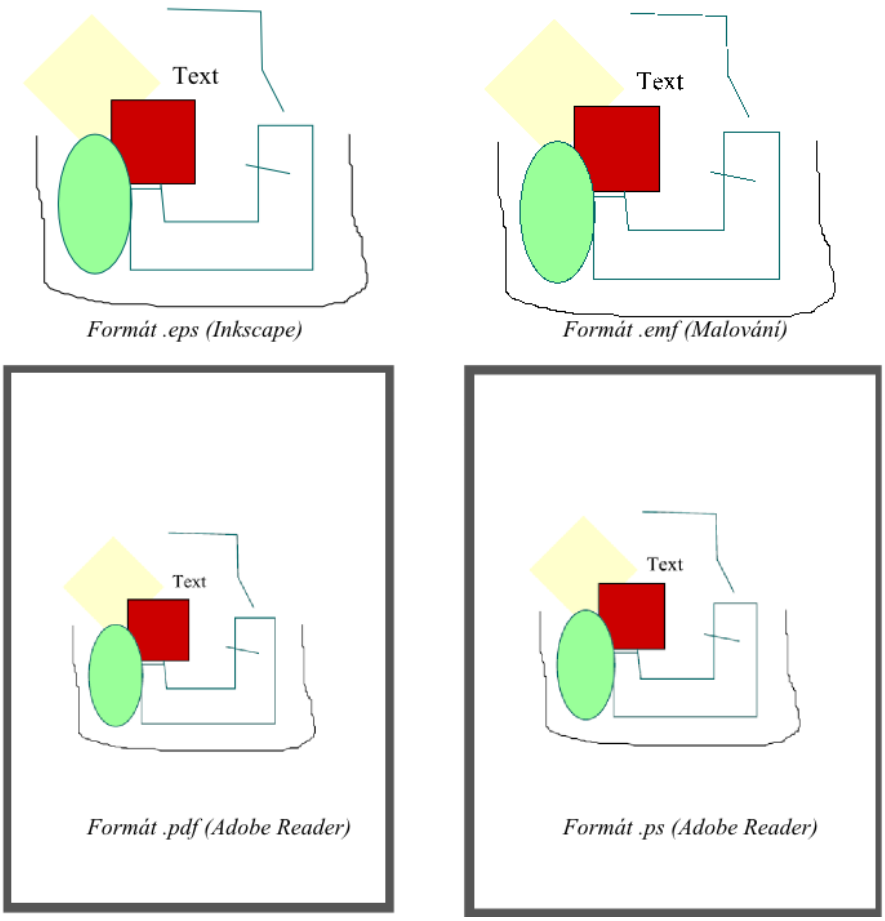
Na obrázku 8.6 vidíme kresbu připravenou k exportu spolu s exportním dialogem. Jedná se o obrázek s nestandardními rozměry, jeho šířka i výška jsou 10cm. Obsahuje tvary, které je možné editorem nakreslit. Tato kresba byla postupně exportována do různých formátů. Srovnání výstupů exportů do vektorových formátů pak vidíme na obrázku 8.7, kde jsou také uvedeny programy, ve kterých byl výstup následně otevřen.

8.2.1 Vektorové formáty

Jak si můžeme povšimnout, export do formátu *.eps* proběhl zcela správně. V editoru *Inkscape* bylo dokonce možné jej dále editovat. Stejně tak bylo možné upravovat kresbu exportovanou do formátu *.emf*, otevřenou v *Malování*. Zde se však projevilo nedokonalé vyhlazování hran, pravděpodobně kvůli samotnému *Malování*, které vyhlazování nenastavuje. Formáty *.pdf a .ps*, otevřené v programu *Adobe Reader*, byly zobrazeny sice korektně, ale při exportu očividně nerespektovaly nastavené rozměry a tak byl výstup umístěn na střed stránky formátu A4. V obrázku 8.7 byl u těchto formátů ponechán okraj stránky, aby bylo možné vidět nedokonalost exportu co se týče rozměrů stránky.



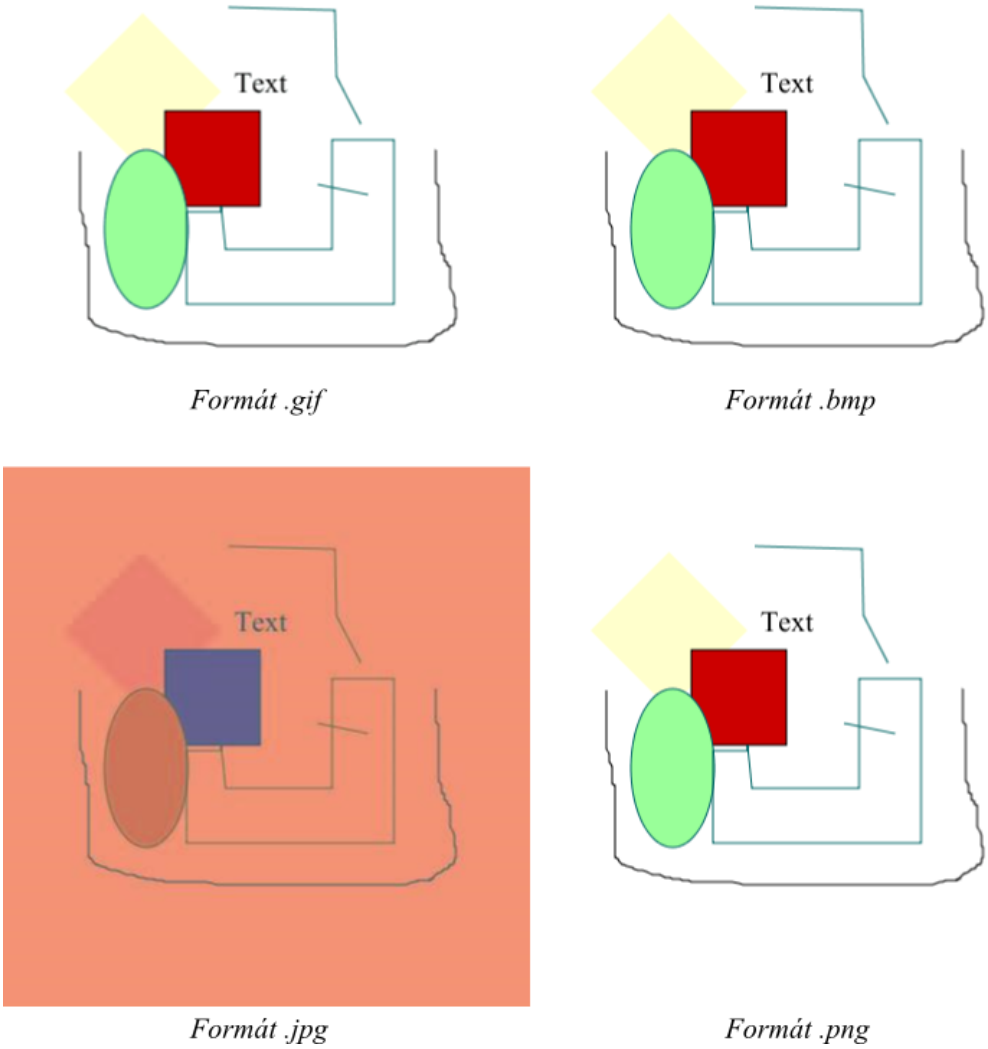
Obrázek 8.6: Kresba připravená k exportu



Obrázek 8.7: Srovnání exportu do vektorových formátů

8.2.2 Rastrové formáty

Obdobný přístup byl pak použit při testování exportu do rastrových formátů. Zde byla daná velikost obrázku bez výjimky zachována. To můžeme vidět vyobrazené na obrázku 8.8. U všech formátů s výjimkou *.jpg* pak byla zobrazena exportovaná kresba s drobnými odchylkami v závislosti na vlastnostech formátu. Při exportu však byl na formát *.jpg* uplatněn jakýsi barevný filtr v odstínu červené, jehož původ není zcela jasný, avšak patrně se bude jednat o nějaký druh chyby při kompresi do tohoto formátu. U zbylých formátů je zobrazení prakticky identické, neboť při tomto rozsahu kresby a možností editoru se vlastnosti těchto rastrových formátů příliš neprojeví. Rozdíl je však vidět ve velikosti jednotlivých souborů s obrázky, zatímco má kresba ve formátu *.gif* pouhých 8kB, kresba uložená v *.png* má velikost 12kB a bitmapový obrázek ve formátu *.bmp* je se svými 420kB mnohonásobně větší.



Obrázek 8.8: Srovnání exportu do rastrových formátů

9 Závěr

Při práci na tomto vektorovém editoru byly prozkoumány možnosti jeho realizace a tím i osvětlena práce s několika různými knihovnami pro práci s vektorovými formáty v jazyce Java. Knihovny vhodné k použití byly popsány a na základě vyhodnocení pak využity při implementaci vektorového editoru.

Zvolený způsob použití knihoven, tedy kombinace knihovny SVG Salamander a FreeHEP VectorGraphics se ukázal být realizovatelný a také realizován byl. Práce tak zcela splňuje zadání. Implementaci však provázely problémy, které byly předpokládány v teoretické části. Editor tak obsahuje několik nedokonalostí, z nichž mnohé vyplývají právě z nedokonalosti použitých knihoven. Tyto problémy souvisí především s načítáním, ukládáním a exportem.

Editor byl průběžně i celkově testován. Popis testů a jejich výsledky jsou rovněž součástí práce. Editor svým chováním a možnostmi zcela dostačuje běžnému uživateli pro kreslení základních vektorových primitiv. Jeho několik nedokonalostí vyvažuje jeho jednoduchost, při které je vytváření jednoduchých vektorových kreseb méně uživatelsky náročné než v případě komplexnějších a složitějších editorů.

Zpracovaný program vzhledem částečně připomíná svůj rastrový protějšek *Malování,* který byl dán jako hrubá předloha, a to jak chováním, tak způsobem použití. Vzhledem k jeho vektorové povaze se však od tohoto programu v mnohém odlišuje. Specifikované požadavky však vytvořený program splňuje, včetně požadavků s nižší prioritou implementace.

Reference

1. Semecký, Jiří. Naučte se Javu - grafické uživatelské rozhraní. *Interval.cz.* [Online] [Citace: 10. Prosinec 2012.] http://interval.cz/clanky/naucte-se-javu-graficke-uzivatelske-rozhrani-1/.

2. Fletcher, Josh. AWT vs Swing. *Embarcadero developer network.* [Online] [Citace: 11. prosinec 2012.] http://edn.embarcadero.com/article/26970.

3. Swing (Java). *Wikipedia The Free Encyclopedia.* [Online] [Citace: 11. Prosinec 2012.] http://en.wikipedia.org/wiki/Swing_(Java)#History.

4. SWT: The Standard Widget Toolkit. *Eclipse.org.* [Online] [Citace: 20. Duben 2013.] http://www.eclipse.org/swt/.

5. Knudsen, Jonathan. *Java 2D Graphics.* Sebastopol : O'Reilly Media, Inc., 1999.

6. Miano, John. *Compressed image file formats.* místo neznámé : Addison Wesley Longman, Inc., 1999.

7. (W3C), World Wide Web Consortium. W3C SVG Working group. [Online] [Citace: Říjen 11, 2012.] http://www.w3.org/Graphics/SVG/.

8. Eisenberg, J. David. *SVG Essentials.* Sebastopol : O'Reilly Media, Inc., 2002.

9. Brabec, Stanislav. Grafika v UNIXu - PostScript. *Root.cz.* [Online] [Citace: 14. Listopad 2012.] http://www.root.cz/clanky/grafika-v-unixu-ix-postscript/.

10. *Adobe PostScript Tutorial and Cookbook.* místo neznámé : Adobe Systems Incorporated.

11. Whitington, John. *PDF Explained.* místo neznámé : O'Reilly Media, Inc.

12. Introduction to PDF. *http://www.gnupdf.org/Introduction_to_PDF*. [Online] [Citace: 23. Březen 2013.] http://www.gnupdf.org/Introduction_to_PDF.

13. PDF Reference and Adobe Extensions to the PDF Specification | Adobe Developer Connection. [Online] Adobe. [Citace: 20. Duben 2013.] http://www.adobe.com/devnet/pdf/pdf_reference.html.

14. Rastrové formáty. *Centrum NLP.* [Online] [Citace: 28. Březen 2013.] http://nlp.fi.muni.cz/cs/Rastrove_formaty.

15. Tišnovský, Pavel. Programujeme JPEG: diskrétní kosinová transformace (DCT). *Root.cz.* [Online] http://www.root.cz/clanky/programujeme-jpeg-diskretni-kosinova-transformace-dct/.

16. Raster Graphics Format. [Online] [Citace: 11. Duben 2013.] http://www.e-cartouche.ch/content_reg/cartouche/formats/en/html/gformats_learningObject2.html.

17. Apache(tm) Batik SVG Toolkit - a Java-based toolkit for applications or applets that want to use images in the Scalable Vector Graphics (SVG). *Apache(TM) Batik SVG Toolkit.* [Online] [Citace: 23. Září 2012.] http://xmlgraphics.apache.org/batik/.

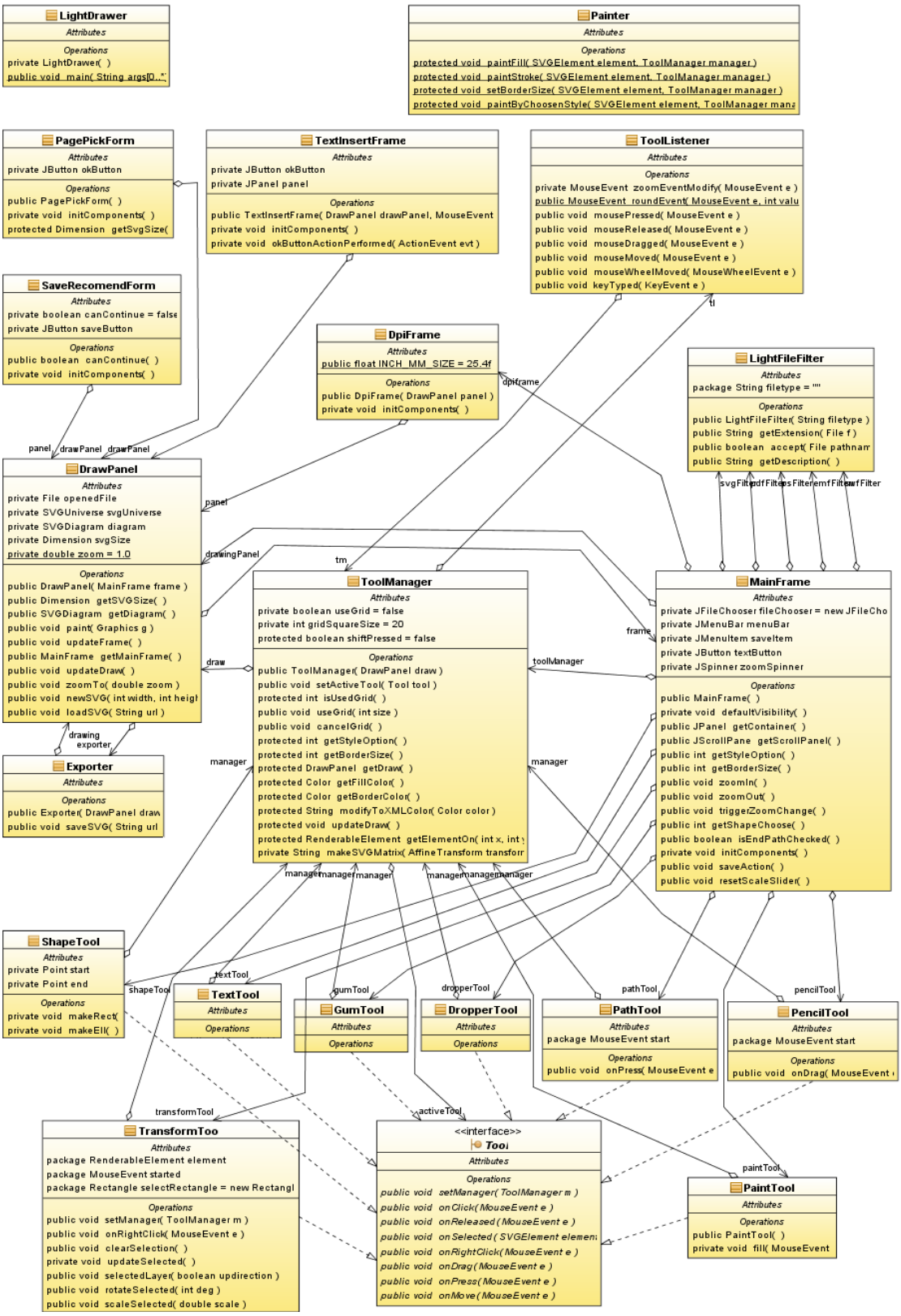
18. SVG Salamander - SVG Parser and Player. *Java.net.* [Online] [Citace: 20. Srpen 2012.] https://svgsalamander.java.net/.

19. VectorGraphics2D. *Vector Graphics 2D.* [Online] [Citace: 10. Listopad 2012.] http://trac.erichseifert.de/vectorgraphics2d/.

20. VectorGraphics. *Vector Graphics 2D Vector an Image Library.* [Online] [Citace: 10. Listopad 2012.] http://java.freehep.org/freehep1.x/vectorgraphics/index.html.

Přílohy

Příloha A – UML diagram tříd



Příloha B – Uživatelská příručka

Tento dokument slouží jako uživatelská příručka k vektorovému editoru *LightDrawer*. Jsou zde popsány jeho jednotlivé funkce a způsoby, jakými je používat.

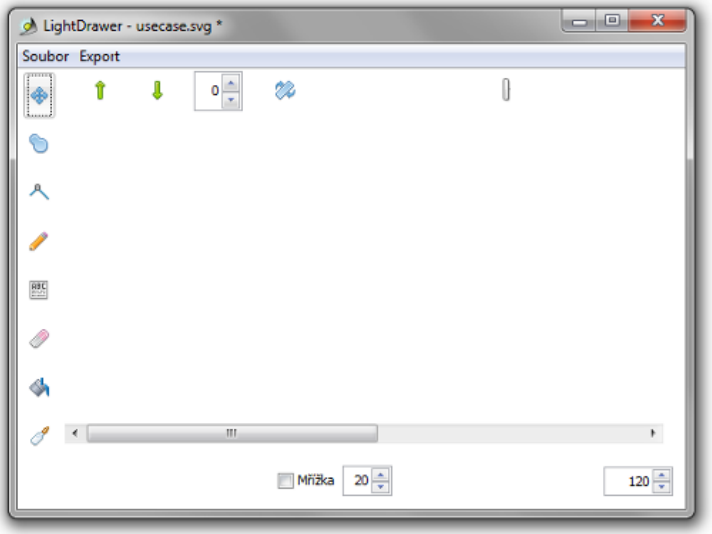
Spuštění

Pro spuštění programu je potřeba mít v operačním systému nainstalovánu Javu SDK 1.6 nebo vyšší. Způsob spuštění samotného programu se pak odvíjí od způsobu spouštění souborů typu *.jar* v daném operačním systému, v systému *Windows* jde typicky o pouhé otevření dvojitým poklepání.

Práce se soubory kreseb

Po spuštění programu se na plátno automaticky načte kresba s velikostí stránky formátu A4. Pokud chcete vytvořit novou kresbu s jinými rozměry, klepněte na menu Soubor → Nový. Objeví se dialog, do kterého zadejte velikost kresby v milimetrech, popřípadě máte na výběr několik různých přednastavení pro běžné formáty stran. Po potvrzení dialogu je stávající plátno nahrazeno novým s odpovídajícími rozměry.

Pro načtení existujícího souboru z disku klepněte na volbu Soubor → Otevřít. Otevře se dialog s adresářovou strukturou souborů na vašem disku. Vyberte požadovaný soubor k otevření ve formátu SVG. Po potvrzení se soubor načte na plátno a je možné jej editovat. Pokud se soubor nemůže načíst (plátno se nezobrazí apod.), znamená to, že soubor vytvořený v jiném programu je příliš složitý pro *LightDrawer*. V takové situaci je vhodné pro editaci takových souborů zvolit jiný editor. Příklad takové situace můžeme vidět na obrázku níže.



Ukládání probíhá následujícím způsobem – napoprvé je potřeba určit, kam se má soubor ukládat, k tomu vede volba Soubor → Uložit i Soubor → Uložit jako… . Pokud je cesta k uložení již zvolena, volba Uložit jej znovu uloží na tuto zadanou cestu. Zda je soubor uložený poznáme podle toho, že zmizí hvězdička indikující změnu dokumentu v hlavičce okna editoru. Při zvolení Uložit jako… lze vždy zvolit, kam bude soubor ukládán. Editor se také při možnosti ztráty neuložených změn dotáže, za má daná data uložit.

Export kreseb

Editor LightDrawer nabízí několik možností, jak exportovat kresbu do jiného formátu než je *.svg*. K tomu slouží záložka v menu Export. První z možností je takzvaný Rychlý export. Je to možnost sloužící pro export do vektorových formátů, aniž by je uživatel musel vyhledávat v dialogu při běžném exportu. Také exportuje soubory pokaždé v jejich skutečné velikosti a nezajímá se o aktuální nastavení. Export do zvoleného formátu probíhá stejně jako v případě ukládání dokumentu.

Další možností je nastavení DPI. Při jeho zvolení se objeví dialog, kde lze nastavit požadovanou velikost obrázku a jeho rozlišení, pokud budeme exportovat do rastrových formátů. Tento dialog zachovává poměr stran originální kresby, stačí tedy určit pouze jeho šířku či výšku v milimetrech. Po nastavení těchto parametrů a potvrzení dialogu se patřičně upraví velikost kresby a objeví se dialog pro export, ve kterém je třeba nastavit typ souboru pro export, zvolit umístění souboru na disku a případně určit další parametry pro export do daného formátu. Po potvrzení se soubor v zadaném formátu uloží na dané místo na disk. Tento dialog je pak možné vyvolat znovu pomocí volby Export→Export. Tato volba již nezobrazuje dialog pro nastavení DPI, ale kresbu změní podle předchozího nastavení DPI. Pokud dialog pro DPI ještě nebyl spuštěn, použije se skutečná velikost kresby.

Zacházení s editorem

Při práci s editorem lze plátno přibližovat či oddalovat, a to pomocí kolečka myši, popřípadě nastavením pole Přiblížení v pravém dolním rohu editoru. Tím se kresba procentuálně zvětšuje či zmenšuje, ale při ukládání toto nastavení nebude mít žádný vliv. Hodnoty přiblížení jsou udávány v procentech, pokud jsou tedy vyšší než 100, znamená to přiblíženou kresbu, pokud nižší, znamená to, že kresba je oddálená.

Během práce je také možné využívat mřížku, sloužící pro přesnou práci s kresbou, kdy se nástroje pro práci s objekty přichytávají k jejím průsečíkům, a to vždy k levému hornímu rohu čtverce, ve kterém se nachází kurzor myši. Její nastavení se nachází

v dolní části. Je zde zaškrtávací políčko pro její zapnutí nebo vypnutí mřížky. Také lze parametrem nastavit velikost čtverců této mřížky a tím upravit rozsah přichytávání. Mřížka bude zobrazena pod kresbou, ale při exportu se neprojeví.

Editace kresby

K editaci kresby se používají nástroje ve sloupci v levé části editoru. Při klepnutí na některý z nich se tento nástroj označí jako aktivní a případně se v horní části editoru objeví jeho doplňující nastavení. Interakce nástroje s kresbou probíhá za pomoci použití myši na kresbu. To je různé pro každý nástroj. Funkce jednotlivých nástrojů je popsána níže.

Nástroj transformace

Ikona: 

Tento nástroj slouží k přesouvání, rotaci, změně vrstvení nebo změně velikosti objektů. Umožňuje označování objektu pomocí kliknutí na něj, popřípadě i zvolení více objektů pomocí kliknutí do prázdného prostoru a tažení myši. Také je možné označit další objekt přidržením klávesy *Shift (levý)* a kliknutím na daný objekt. Kliknutím na objekt nebo skupinu objektů a následným tažením myši můžeme objekty přesouvat po plátnu.

Nastavení tohoto nástroje pak neovlivňují chování nástroje jako takového, ale mění přímo elementy kresby označené tímto nástrojem.

Dostupná nastavení:

- Vrstvení (šipky) – Pomocí nastavení v horní části pak můžeme zvolené objekty posouvat „nad“ nebo „pod“ jiné pomocí tlačítek se šipkami. Toto nastavení se však vztahuje ke všem objektům v kresbě a proto je někdy potřeba kliknout na šipku vícekrát, aby se efekt vizuálně projevil.
- Rotace – zvolené objekty můžeme rotovat podle jejich středu. Toho dosáhneme tak, že označíme objekty, které chceme otáčet, zvolíme v tomto nastavení úhel otočení ve směru hodinových ručiček a klepneme na tlačítko pro rotaci. Úhel lze zadat zcela libovolně pomocí zadání konkrétního čísla do pole.
- Škálování (změna velikosti) – změna velikosti označených objektů se provádí táhlem pro škálování. Při tažení táhlem doprava se objekt (objekty) zvětšuje, při tažení doleva zmenšuje. Po změně velikosti se táhlo vrátí doprostřed, takže je možné výsledný objekt zvětšit/zmenšit znovu.

Nástroj kreslení tvarů

Ikona: 

Tento nástroj slouží pro kreslení základních geometrických tvarů – čtverců, obdélníků, kruhů a elips. Používá se tak, že se klikne do plátna a následně se provádí tažení myší. Začne se vykreslovat pomocný tvar, podle kterého poznáme, jak bude vypadat výsledný obrazec. Chování kreslení závisí na nastavení nástroje. V nastavení se také volí tvar, který bude kreslen a podle toho se chová i kreslení – počáteční kliknutí určuje buď levý horní roh čtverce nebo obdélníka, nebo střed kružnice či elipsy.

Dostupná nastavení:

- Barva výplně – první tlačítko s barvou má takovou barvu, jaká bude použita při výplni objektu (je–li výplň vyžadována dalším nastavením). Pro její změnu lze kliknout na toto tlačítko a ve zobrazeném dialogu vybrat a potvrdit barvu jinou.
- Barva okraje – druhé tlačítko se chová stejně jako tlačítko pro barvu výplně, avšak určuje barvu okraje.
- Nastavení výplně a okraje – následující nastavení určuje, zda bude při kreslení vykreslen okraj, výplň nebo obojí. To je znázorněno obrázky pro tuto volbu.
- Šířka čáry – pole s číslem určuje šířku okraje v pixelech, pokud má být okraj vykreslen.
- Typ objektu – poslední volba nastavuje typ objektu, který má být nakreslen, tedy čtverec, obdélník, kruh či elipsu. Barevné rozlišení je zde pouze pro orientaci.

Nástroj kreslení lomené čáry

Ikona: 

Tímto nástrojem lze kreslit lomenou čáru, která však nemusí mít okraj a může mít výplň. Obsahuje podobná nastavení jako kreslení tvarů. Ovládá se kliknutím do plátna levým tlačítkem. Tím se vytvoří první bod čáry. Pro vytvoření dalšího bodu klikneme do plátna znovu a tyto body jsou propojeny. Pokud chceme čáru ukončit, klikneme pravým tlačítkem myši, čímž se vytvoří poslední bod a čára se vykreslí. Pokud při kreslení podržíme klávesu *Shift (levý)*, čáry se budou kreslit pouze vertikálně nebo horizontálně.

Dostupná nastavení:

- Spojení – zaškrtávací tlačítko pro spojení konce čáry s jejím začátkem. Je velmi praktické například při kreslení trojúhelníků. Při zaškrtnutí se po dokreslení čáry propojí oba konce.

- Další nastavení – jsou identická s nastaveními pro kreslení tvarů, viz výše. Pokud chceme kreslit pouze čáru, je třeba zvolit takové nastavení kreslení, aby se vykresloval pouze okraj.

Nástroj tužka

Ikona: 

Tímto nástrojem lze kreslit kliknutím a tažením myši, přičemž se po dobu kliknutí kreslí pomocná čára. Po uvolnění tlačítka se vykreslí čára o daných nastaveních. I tento nástroj může mít stejně jako lomená čára výplň, tudíž je pro kreslení pouze čáry nutné to zohlednit v nastavení.

Dostupná nastavení:

- Nastavení jsou identická s nastaveními lomené čáry, viz výše.

Nástroj text

Ikona: 

Tímto nástrojem lze vkládat do kresby text. Tento text posléze není možné měnit, po zobrazení se chová jako běžný objekt jako je například lomená čára. Ovládá se kliknutím do kresby, kde má text začínat. V dialogu, který se následně objeví, je možné nastavit parametry textu jako je velikost, typ písma nebo použití kurzívy. Do pole pod nastavením je pak třeba vložit text, který má být vykreslen. Po potvrzení dialogu se tento text objeví na daném místě. s tímto textem se poté dá zacházet stejně jako s jinými objekty.

Nástroj guma

Ikona: 

Tento nástroj se chová jinak než v běžných rastrových editorech. Neslouží pro „gumování“ v obrázku, ale ke smazání celých objektů. Po zvolení tohoto nástroje stačí kliknout na objekt, který má být z kresby vymazán.

Nástroj plechovka

Ikona: 

Nástroj plechovka neslouží navzdory svému názvu jako nástroj pro výplň ohraničených ploch, ale pro změnu jednotlivých objektů, kterým lze měnit především barvy výplně a ohraničení (proto plechovka). Pro její použití stačí zvolit tento nástroj a kliknout

na objekt, který má být změněn. Po kliknutí se objektu nastaví ty parametry, které jsou dány v nastavení nástroje, tedy barva výplně a okraje, způsob vykreslení a šířka okraje.

Dostupná nastavení:

- Nastavení jsou prakticky identická s nastaveními pro kreslení objektů, viz výše. Pochopitelně však neobsahují nastavení typu objektu.

Nástroj kapátko

Ikona: 

Tento nástroj slouží pro zpětné zjištění nastavení barev výplně a okraje. Pro použití stačí kliknout na objekt, jehož barvy chceme zjistit. Pokud objekt výplň nemá, nastaví se výplň bílá, pokud nemá barvu čáry, nastaví se na její místo černá. Tyto barvy jsou pak použity v ostatních nástrojích, které je využívají. Převzaté barvy lze případně upravit kliknutím na tlačítko, které je přenáší.