VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Nástroj pro tvorbu webových animací

Web Animation Authoring Tool

2015 Dávid Ivan

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student:	**Dávid Ivan**
Studijní program:	B2647 Informační a komunikační technologie
Studijní obor:	2612R025 Informatika a výpočetní technika
Téma:	Nástroj pro tvorbu webových animací Web Animation Authoring Tool

Zásady pro vypracování:

Cílem práce je implementace aplikace pro tvorbu webových animací, založené na standardu HTML5. Součástí práce bude přehled existujících nástrojů pro tvorbu webových animací. V práci budou představeny HTML5 technologie zaměřené na grafiku a animaci (CSS3, canvas, SVG, WebGL, apod.). Vybrané technologie budou následně využity při návrhu a implementaci samotné aplikace. Výstupem aplikace budou animace, jež bude možné spouštět nejen na desktopu, ale i na mobilních zařízeních, jako jsou mobilní telefony, nebo tablety. Nad celou aplikací bude provedeno uživatelské testování.

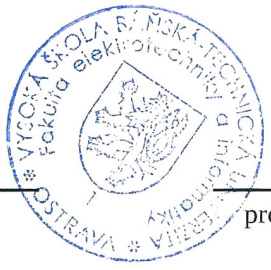
Seznam doporučené odborné literatury:

[1] STOREY, Dudley. Pro CSS3 Animation. S.l.: Apress, 2012. ISBN 978-143-0247-227.
[2] PILGRIM, Mark. HTML5: up and running. 1st ed. Sebastopol, CA: O'Reilly, 2010. ISBN 05-968-0602-7.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Janoušek**

Datum zadání: 01.09.2014
Datum odevzdání: 07.05.2015



doc. Dr. Ing. Eduard Sojka	prof. RNDr. Václav Snášel, CSc.
vedoucí katedry	*děkan fakulty*

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2015 .....

Rád bych poděkoval vedoucímu mé bakalářské práce panu Ing. Janu Janouškovi za odborné vedení a cenné rady, které mi pomohly tuto práci zkompletovat.

Abstrakt

Cílem této práce je vytvoření aplikace pro tvorbu webových animací, založené na standardu HTML5. Při návrhu a implementaci aplikace jsou využity vybrané HTML5 technologie, jako CSS3, SVG nebo canvas. První část této práce popisuje stávající aplikace, řešící tvorbu webových animací. Druhá část představuje samotný HTML5 standard a jeho vybrané technologie, které byly prakticky využity v této práci. Třetí část se věnuje samotnému návrhu a implementaci aplikace. Závěrečná část popisuje výsledky uživatelského testování a výkonnostního testování, provedeného nad vytvářenou animací.

Klíčová slova: HTML5, CSS3, SVG, TypeScript, webové animace, autorský nástroj, klíčové snímky, transformace, přechody, uživatelské testování

Abstract

The main goal of this thesis is to create web animation authoring tool based on HTML5 standard. Selected technologies of HTML5 are used for design and implementation of tool, such as CSS3, SVG or canvas. The first part of this thesis describes existing tools that are used to create web animations. The second part introduces HTML5 standard itself and selected technologies of HTML5 practically used in this thesis. The third part deals with design and implementation of authoring tool. The last part describes results of user experience testing and performance testing of created animation.

Keywords: HTML5, CSS3, SVG, TypeScript, web animation, authoring tool, keyframes, Transforms, Transitions, user experience

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CSS	–	Cascading Style Sheets
CSS3	–	Cascading Style Sheets Level 3
FPS	–	Frames per second
GIF	–	Graphics Interchange Format
HSL	–	hue - saturation - lightness
HSLA	–	hue - saturation - lightness - alpha
HTML	–	HyperText Markup Language
HTML5	–	HyperText Markup Language version 5
ID	–	Identification
JS	–	JavaScript
px	–	pixel
PNG	–	Portable Network Graphics
RGB	–	red - blue - green
RGBA	–	red - blue - green - alpha
SVG	–	Scalable Vector Graphics
UX	–	User Experience
XHTML	–	Extensible HyperText Markup Language
XML	–	Extensible Markup Language
W3C	–	World Wide Web Consortium

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V roce 1994 Håkon Wium Lie publikoval koncept popisující CSS. Kaskádové styly měly oddělit vzhled dokumentu od jeho obsahu a struktury. HTML definovalo strukturu dokumentu - nadpisy, odstavce, obrázky a jiné elementy. CSS řešilo, jak se daný element zobrazoval uživateli. Původní verze CSS a CSS verze 2 byly navrženy pro formátování vzhledu statického obsahu. Pro změnu vlastnosti elementu z jednoho stavu do druhého bylo nutné sáhnout pro jiné technologie, nejčastěji JavaScript nebo Flash.

Směr vývoje udávalo mimo jiné i použití reklam na webu. První reklamní proužky se objevovaly již v 90. letech v podobě statických a později animovaných GIF souborů. Postupně byla požadovaná větší interaktivita, a proto se začaly používat výše zmíněné technologie. Velkou popularitu měl Flash, který je v dnešní době vytlačován CSS verze 3, která podporuje animace. Ve spojení s HTML5 dál poskytuje nové technologie pro vývoj moderních a interaktivních webových aplikací.

Cílem této práce bude popsat dostupné CSS3 a HTML5 technologie, zejména technologie zaměřené na grafiku a animaci, a implementovat webovou aplikaci pro jednoduchou tvorbu animací. První část práce bude věnovaná popisu již dostupných řešení, které mohou být v současné době využity ke tvorbě animací. V této části budou dál popsány využité technologie. Bude popsán samotný HTML5 standard, element `<canvas>`, CSS animace, CSS transformace a SVG formát, který umožňuje vkládat do stránek vektorovou grafiku. Druhá část práce bude popisovat návrh a implementaci aplikace pro tvorbu webových animací. Součástí této části budou i výsledky uživatelského testování a výkonnostního testování.

2 Existující řešení pro tvorbu webových animací

Pro vytváření webových animací s použitím *CSS Keyframes* dnes existuje mnoho nástrojů, které lze rozdělit do dvou kategorií. První kategorií jsou desktopové aplikace, vyžadující instalaci na počítač. Tyto aplikace jsou složitější a nabízející mnoho různých nástrojů. Typickým použitím těchto aplikací je tvorba animovaných a interaktivních bannerů. Jako příklad desktopových aplikací lze uvést **Adobe Flash Professional** nebo **Google Web Designer**. Druhou kategorií jsou online nástroje spustitelné v prohlížeči. Žádný z online nástrojů neposkytuje široké možnosti desktopových aplikací a tyto nástroje jsou, až na pár výjimek, určeny pro tvorbu jednoduché animace jednoho objektu. Mezi výjimky patří úzce zaměřený **BannerFlow** určený pro tvorbu bannerů.

2.1 Adobe Flash Professional

Flash Professional od společnosti Adobe Systems Inc. v aktuální verzi CC 2014.1(Creative Cloud) slouží k vytváření animací a multimediálního obsahu. Původní zaměření této aplikace bylo vytváření aplikací ve Flashi pomocí objektově orientovaného programovacího jazyka *Action Script*, který je součástí Adobe Flash. S příchodem verze CS6 v roce 2012 přichází možnost vytvářet animace a webové aplikace (například i hry) v HTML5 standardu.

Pomocí časové osy, pracovní plochy a dostupných nástrojů lze vytvořit animovaný obsah a výstupem bude právě HTML5 dokument. Program se postará o převedení již vytvořené aplikace do HTML a JavaScriptu. Výsledkem je spustitelný soubor typu html s elementem <canvas>, který obsahuje všechny animované objekty.

2.2 Adobe Edge Animate

Adobe Edge Animate je další z produktů společnosti Adobe, umožňující vytvářet interaktivní animace v HTML, kompatibilní se všemi desktopovými i mobilními prohlížeči. Aplikace patří do skupiny CC (Creative Cloud, v aktuální verzi 2014.1) a je to, stejně jako Adobe Flash Professional, komerční software.

Výstupem aplikace je, na rozdíl od Adobe Flash Professional, plnohodnotný HTML obsah. Je možné vkládat například elementy ** **, **<img>** nebo <svg> a ve stejné podobě se tyto elementy zobrazí i ve výsledné animaci. Oproti tomu Adobe Flash Professional generuje jeden canvas. Prostředí aplikace poskytuje standardní prvky pro tvorbu interaktivního HTML obsahu – panel nástrojů a panel CSS vlastností, pracovní plochu a časovou osu.

Aplikace po vytvoření projektu vygeneruje JavaScript soubor, který ovládá celou animaci a obsluhuje případné události. Skript mění hodnoty CSS transformací daného elementu a tím se vytvoří efekt animace. Velkou výhodou Edge Animate je animování několika CSS vlastností odděleně, nezávisle na jiných. Je tak možné například animovat pohyb objektu v délce dvou sekund a průhlednost z hodnoty 1 na hodnotu 0,5 v délce jedné sekundy. Navíc je nastavení animace přehledně zobrazeno na časové ose, kde každá vlastnost je uvedena na samostatném řádku s přehledem klíčových snímků.

Další výhodou této aplikace je možnost nastavení CSS filtrů pomocí vlastnosti *filter*. Filtr alpha nastaví průhlednost prvku. Mezi další filtry, které lze nastavit patří rozmazání prvku, kontrast, saturace a odstíny šedi.

2.3 Google Web Designer

Google Web Designer je desktopová aplikace od společnosti Google, uvolněna v roce 2013. Slouží k pokročilé tvorbě animací v HTML5. Google tuto aplikaci, běžící na systémech Windows, OS X a Linux, prezentuje jako webovou aplikaci vytvořenou pomocí technologie HTML5 , která umožňuje návrh a vytváření reklam HTML5 a jiného webového obsahu pomocí integrovaného vizuálního a kódového rozhraní [1]. Je tedy vhodný k vytváření bannerů. Ještě nedávno byly interaktivní a animované reklamy vytvářeny pomocí technologie Flash, vyžadující zásuvný modul prohlížeče. Flash reklamy se nezobrazovaly v mobilních zařízeních a tabletech. I dnes se Flash obsah poměrně často na stránkách vyskytuje. Potřeba zobrazovat obsah na různých typech zobrazovacích zařízení vedla k využití nových technologií, zejména *CSS3 Keyframes*. Samozřejmě je možné vytvářet i jiný webový grafický obsah. Aplikace je k dispozici ke stažení zdarma.

Práce s aplikací je jednoduchá a intuitivní. K dispozici je interaktivní i kódové rozhraní. Rozhraní je typické pro aplikace podobného typu a zaměření. Velkou část tvoří pracovní prostor, který má několik režimů. V režimu návrhu se na ploše nacházejí vizualizace objektů ve stejné podobě, v jaké se budou zobrazovat v prohlížeči. V režimu kódu je zobrazen kód popisující tyto objekty, který lze editovat. Na levé straně je panel nástrojů, který umožňuje vkládání obsahu a manipulaci s elementy. Je možné vložit text, HTML značky a grafiku založenou na elementu <canvas>.

Na pravé straně se nachází panel pro editaci vlastností. Obsahuje panely *Barva*, *Vlastnosti*, *Komponenty*, *Události* a *CSS*. *Komponenty* jsou předem vytvořené moduly pro přímé specifické využití (například vložení posuvné galerie nebo mapy). Panel *Vlastnosti* umožňuje změnu CSS vlastností elementu. Přímá editace stylů je možná v panelu *CSS*. Události slouží k zachycení určité akce uživatele a provedení požadované změny. Například uživatel může kliknout na tlačítko a tím se spustí animace.

Dolní část obsahuje časovou osu, která slouží k vytváření jednotlivých klíčových snímků – **keyframes**, které popisují vlastnosti daného elementu v určitém čase.

2.4 Online nástroje

Pro uživatelsky přívětivou definici klíčových snímků existuje nepřeberné množství webových nástrojů dostupných on-line. Výhodou je, že se nemusí nic instalovat, a tak je okamžitě k dispozici nástroj, který zvládne základní animace. Nevýhodou je u některých nástrojů přílišná jednoduchost a tyto nástroje se hodí spíše k základnímu vyzkoušení a demonstraci různých typů animací. Výstupem je HTML a CSS kód s klíčovými snímky.

2.4.1 Stylie

Nástroj [2] slouží k demonstraci animace pozice, rotace a velikosti objektu. Je možné vytvářet a mazat jednotlivé klíčové snímky a měnit pozici objektu v jednotlivých snímcích pohybem po plátně. Pomocí jednoduchého rozhraní lze měnit jak pozici, tak i velikost objektu pomocí funkce scale() a 3D rotaci. Aplikace nemá klasickou časovou osu, disponuje pouze výčtem klíčových snímků. Dobu trvání animace je možné ovládat editováním textového pole. Na plátně lze vidět průběh animace a lze ji pozastavovat a posouvat. Výsledkem je vygenerovaný CSS kód s jednotlivými klíčovými snímky.

2.4.2 CSS Animate

CSS Animate [3] je již pokročilejší nástroj pro vytváření animací. Poskytuje jednoduchou časovou osu s vyznačenými klíčovými snímky, které lze přetahovat a měnit tím dobu trvání animace. Jako u předešlého nástroje, i tady je možné animovat pouze jeden objekt, který nelze měnit a tak je tento nástroj vhodný spíše k demonstraci různých animací a k rychlému vytvoření jednoduché animace. Lze animovat pozici, rotaci, zkosení a průhlednost objektu. Výsledkem je opět vygenerovaný HTML a CSS kód.

2.4.3 BannerFlow

BannerFlow [4] je na rozdíl od předešlých nástrojů propracovanější, komerční a má úzké zaměření - je určen pro tvorbu bannerových reklam. Na plátno lze vkládat libovolný počet objektů, jako jsou obrázky, textová pole a tlačítka. CSS vlastnosti objektů se mění pomocí ovládacího panelu. V dolní části se nachází časová osa, na které jsou vyznačené jednotlivé objekty v podobě pruhů, které označují dobu trvání animace. Předdefinováno je několik animovaných přechodů typických pro pohyblivé bannery – změna velikosti objektů a přilétání objektů z různých stran. Změnou pozice, velikosti, průhlednosti a rotace je možné vytvořit vlastní definice přechodů. Výsledkem je vygenerovaný banner v HTML s klíčovými snímky v CSS stylu.

3 HTML5

HTML5 je aktuální verze značkovacího jazyka HTML vytvořená mezinárodním konsorciem World Wide Web Consortium (W3C). Oproti předchozí verzi HTML 4.01 přináší zásadní změny a poskytuje moderní technologie pro vývoj webových aplikací. Vývoj verze 5 započal kolem roku 2007, kdy byla založena pracovní skupina, mající za úkol vytvořit novou verzi HTML, a to na úkor XHTML. XHTML nepřidaly oproti HTML 4.01 v podstatě žádnou novou funkčnost, nově chystaná verze byla dokonce nekompatibilní s předchozími verzemi jazyka HTML. Proto byl vývoj zaměřen právě na HTML5. Finální specifikaci vydalo konsorcium W3C 28. října 2014 [5].

Velká část technologií standardu HTML5 je v současné době podporována nejznámějšími prohlížeči v aktuální verzi - Internet Explorer, Mozilla Firefox a Google Chrome. Například nahrávání více souborů najednou je podporováno ve všech vyjmenovaných prohlížečích. Dalším příkladem je databáze v prohlížeči **IndexedDB**, která je podporována v prohlížečích Firefox, Chrome a částečně v prohlížeči Internet Explorer [6]. Technologie HTML5 lze tedy bez větších problémů používat již dnes.

Pro webovou aplikaci zaměřenou na grafiku a animaci lze využít několik technologií, které nová verze HTML poskytuje. Existují jak technologie přímo pro práci s grafikou, tak i jiné, vhodné pro jakoukoliv moderní webovou aplikaci. Plnohodnotné aplikace využívají pro svůj chod různé technologie, jako například ukládání dat do databáze v prohlížeči, práce s uživatelskými soubory, práce s binárními daty a další jiné technologie. Jednotlivé prohlížeče proto poskytují různá **API**. Je tak možné využít například vkládání multimediálního obsahu nebo kreslení pomocí JavaScriptu do elementu <canvas>. Prohlížeče dále poskytují API pro manipulaci s databázovým úložištěm přímo v prohlížeči. Je možné využít lokální úložiště *Web Storage* k ukládání jednoduchých dat ve formě klíč – hodnota. Ukládat složitější struktury je možné pomocí technologií *Web Database* nebo *IndexedDB*. Pro webové aplikace je v neposlední řadě hodně využívaná *Offline* technologie, zajišťující fungování aplikace i bez internetového připojení. Pro čtení a práci s uživatelskými soubory přímo v JavaScriptu se využívá *File API*. Pro nahrávání jednoduchým přetažením souborů do okna aplikace v prohlížeči slouží technologie *Drag and Drop*.

3.1 Multimediální obsah

V předchozích verzích HTML bylo nutné pro přehrání videa nebo zvuku použít zásuvný modul v prohlížeči. Jako nejznámější plugin pro přehrávání videa lze uvést Adobe Flash. Dnes je však Flash na ústupu a HTML5 poskytuje pro vkládání multimediálního obsahu párové tagy <video> a <audio>.

```
<video width="300" height="200" controls>
    <source src="video.mp4" type="video/mp4">
    <source src="video.ogg" type="video/ogg">
    Vas prohlizec nepodporuje vkladani videa.
</video>
```

Výpis 1: Příklad vložení videa pomocí tagu <video>

Výpis kódu 2 demonstruje vložení videa na stránku o dané velikosti s výchozími ovládacími prvky. Prostý text uvnitř tagu <video> se zobrazí jen tehdy, když prohlížeč nepodporuje tento tag. V ukázce je možné vidět atributy **width** a **height**, které slouží k zadání šířky a výšky videa. Tyto atributy by měly být uvedeny vždy. Atribut **controls** zobrazí ve videu prvky pro ovládání videa. Tag <source> slouží k vložení video souboru. Je možné přidat i několik zdrojů s různými typy formátu videa. Další možností je vložit video pomocí atributu **src**. Tímto způsobem je možné vložit jen jeden formát videa.

Tag <video> má několik dalších atributů. Atribut **autoplay** zajistí spuštění videa ihned po načtení, **loop** zajistí, že video se bude přehrávat ve smyčce a **poster** specifikuje URL obrázku, který se zobrazí namísto videa v době, kde se video načítá.

Všechny známé prohlížeče v nejnovější verzi podporují vkládání videa[1], problémem jsou různé formáty videa. Neexistuje jeden formát, který by fungoval ve všech prohlížečích. Všechny současné prohlížeče (včetně mobilních prohlížečů) lze pokrýt pomocí dvou formátů – *WebM* (kodek *VP8* pro video, *Vorbis* pro audio) a *MP4* (kodek *H.264* pro video, *AAC* pro audio).

Audio soubor je možné vložit použitím tagu <audio>. Kromě atributů width, height a poster lze nastavit stejné atributy jako u videa.

```
<audio controls>
    <source src="audio.ogg" type="audio/ogg">
    <source src="audio.mp3" type="audio/mpeg">
    Vas prohlizec nepodporuje audio.
</audio>
```

Výpis 2: Příklad vložení audio souboru pomocí tagu <audio>

3.2 Canvas

Element <canvas> je párový tag, který vymezuje prostor pro kreslení bitmapové grafiky pomocí JavaScriptu. Velikost plátna je definována šířkou **width** a výškou **height**. Kreslící plátno má podobu obdélníku. Ve výpisu 3 je vytvořeno plátno o velikosti 200x100.

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

Výpis 3: Vytvoření plátna pomocí tagu <canvas>

Canvas má několik metod pro vykreslení čar, obdélníků, kruhů a textu. Do plátna lze vkládat i obrázky. Aby mohlo být možné pomocí JavaScriptu vytvářet grafické objekty, musí se nejdříve získat **context** – kreslící prostředí, ve kterém bude probíhat samotné vykreslování [8]. Ukázka získání contextu je uvedena ve výpisu 4.

3.2.1 Kreslení obdélníků

Před samotným kreslením se nastaví barva výplně pomocí atributu **fillStyle**. Standardní barva před nastavením je černá (#000000). Výplň lze specifikovat barvou, přechodem

[1] Pod pojmem *známé prohlížeče* uvádím tři nejpoužívanější prohlížeče v současné době - Google Chrome, Internet Explorer a Mozilla Firefox [7].

nebo předem definovaným vzorem. Metoda fillRect (x, y, width, height) zajistí vykreslení obdélníku na souřadnicích **x**, **y** o dané šířce **width** a výšce **height**. Plátno je dvojrozměrná mřížka a souřadnice (0, 0) představují horní levý roh plátna.

3.2.2 Kreslení cest

Ke kreslení rovných čar se používají metody moveTo(x, y) a lineTo(x, y). Metoda moveTo(x, y) slouží k přesunutí pomyslného ukazatele na souřadnici definovanou parametry **x** a **y**. Funkce této metody je možné si představit jako zvednutí pera a posunutí na danou pozici. Metoda lineTo(x, y) slouží k nakreslení čáry od současné pozice do požadované pozice. Po zavolání těchto metod se zatím nic nevykreslí. K zobrazení nadefinované cesty slouží metoda stroke(). Před zavoláním této metody je možné nastavit barvu čáry pomocí atributu **strokeStyle**. Šířka čáry se nastavuje atributem **lineWidth**. Je-li požadavkem oddělit různé styly čar, zavolá se metoda beginPath() a vytvoří se nová cesta, které se nastaví nové parametry, jako je barva a šířka. Ukázka práce s cestami je uvedena ve výpisu 4.

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");  //ziskani kontextu
ctx.moveTo(0,0);
ctx.lineTo(200,100);
context.strokeStyle = "#eee";
ctx.stroke();                       // Vykresli nadefinovanou cestu
context.beginPath();                // Nova cesta, s novym stylem
context.moveTo(0, 40);
context.lineTo(240, 40);
context.strokeStyle = "#000";
context.stroke();
```

Výpis 4: Získání kreslícího prostředí - context a ukázka kreslení cesty

Kromě metody stroke() existuje i metoda fill (), která nejdříve uzavře cestu a poté vyplní daný tvar definovanou barvou. Další používaná metoda closePath() zkouší uzavřít tvar vykreslením přímky z aktuálního bodu k bodu představující počátek.

Kreslení přímky není jediný způsob definování cest. Pro složitější tvary slouží metoda bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y), která vykreslí zaoblené čáry pomocí řídících bodů. Parametry **x** a **y** jsou souřadnice konečného bodu, **cp1x** a **cp1y** jsou souřadnice prvního řídícího bodu dvojice parametrů **cp2x**, **cp2y** jsou souřadnice druhého řídícího bodu.

3.2.3 Kreslení kružnic

Metoda arc(x, y, radius, pocatecniUhel, konecnyUhel, protiSmeruHodinovychRucicek) se používá ke kreslení oblouků a kružnic. Parametry **x**, **y** představují souřadnice středu kružnice, parametr **polomer** udává poloměr kružnice, parametry **pocatecniUhel** a **konecnyUhel** udávají počáteční úhel a koncový úhel a nepovinný parametr **protiSmeruHodinovychRucicek** určuje směr vykreslení kružnice.

Technologie canvas umožňuje vykreslit i jiné objekty. Vykreslit text je možné zavoláním metody fillText (text, x, y). Font textu lze nastavit atributem **font**. Obrázek se

na plátno vkládá zavoláním metody drawImage(img, x, y). Jak již bylo zmíněno, objekt lze vyplnit barvou i přechodem. Atribut **fillStyle** přijímá přechod vytvořený metodou createLinearGradient(x0, y0, x1, y1). Přechod je vykreslený podél úsečky z bodu (x0, y0) do bodu (x1, y1). Metoda addColorStop() slouží k definování barev přechodu. První parametr značí milník a druhým parametrem se nastaví barva milníku. Ukázka definice přechodu je uvedena ve výpisu 9.

```
var gradient = context.createLinearGradient(0, 0, 0, 225);
gradient.addColorStop(0, "black");
gradient.addColorStop(1, "white");
context. fillStyle  = gradient;
context. fillRect (0, 0, 300, 225);
```

Výpis 5: Vyplnění objektu vytvořeného v canvasu přechodem

4 CSS3

Třetí verze **CSS (Cascading Style Sheets)** patří spolu s HTML5 k jednoznačným trendům současnosti. CSS3 je často mylně zahrnováno do rodiny HTML5 i proto, že je s ním úzce spjatý. Dnes se kombinace HTML5 a CSS3 používá v každé moderní webové aplikaci. Vývoj třetí verze započal v roce 2005 a dodnes se vyvíjí. Specifikace CSS3 se dělí do modulů, ke kterým se přistupuje jednotlivě a jeden modul může být dokončen a implementován jako samostatný prvek. I když finální specifikace CSS3 ještě nebyla vydána, je v dnešní době podpora prohlížečů na velmi dobré úrovni. Z velké části má na to vliv právě již zmiňované rozdělení vývoje do modulů. Výrobci prohlížečů přidávají podporu různých modulů postupně a tyto moduly nemusí být ani finální. Specifikace modulů se může změnit, a tak výrobci prohlížečů používají **vendor prefixes**. Jedná se předponu u CSS vlastnosti, která říká, že prohlížeč implementuje zkušební, prototypovou verzi. Jestliže se specifikace změní, výrobci prohlížečů ji implementují již do finální verze CSS vlastnosti bez prefixu. Je možné se setkat s prefixy *-webkit*, *-moz*, *-ms*, *-o*.

4.1 Vlastnosti zaměřené na grafiku a animaci

Z hlediska animací a grafiky na webu má CSS3 několik velmi užitečných modulů, které přinášejí například kulaté rohy, přechody nebo stíny u textů a elementů. K zadání barvy se složkou průhlednosti je možné využít barevný model *RGBA*. CSS3 umožňuje vytváření animací bez využití JavaScriptu a i v mnoha dalších oblastech může verze 3 úplně nahradit JavaScript. Dobrým příkladem je nahrazení jQuery metody animate() modulem *Transitions*, který umožní animovat přechod mezi stavy objektu.

4.1.1 Zaoblené rohy

CSS3 vlastnost **border-radius** specifikuje zaoblené hrany elementu. Je tak možné docílit vykreslení kulatých a elipsovitých hran objektů. Hodnota představuje poloměr zakulacení a je možné ji zadávat v pixelech *px* nebo v procentech %. V příkladu 6 je na prvním řádku nastaveno zakulacení s poloměrem 10px pro každý roh a na druhém řádku je nastaveno zaoblení pro každý roh zvlášť. První hodnota udává levý horní roh a postupuje se po směru hodinových ručiček.

```
border-radius: 10px;
border-radius: 15% 15% 0 0;
```

Výpis 6: Příklad použití border-radius

Vlastnost border-radius je zkratka k vlastnostem **border-top-left-radius**, **border-top-right-radius**, **border-bottom-right-radius** a **border-botom-left-radius**, které specifikují zaoblení pro každý roh samostatně.

Další možný tvar zápisu je border-radius: 5px 10px 5px 10px / 10px 5px 10px 5px, který zajistí zakulacení ve tvaru elipsy, kde první 4 hodnoty definují horizontální poloměr elipsy všech 4 rohů a hodnoty za lomítkem definují vertikální poloměr elipsy všech 4 rohů.

V současné době podporují vlastnost border-radius všechny známé prohlížeče v aktuální verzi a to bez prefixů.

4.1.2 Stíny

Pro nastavení stínu elementu slouží vlastnost **box-shadow**. Stín u textu se vytvoří pomocí vlastnosti **text-shadow**. Vlastnost box-shadow může mít několik variant a jeho syntaxe je složitější: box−shadow: (inset)h_posun v_posun (rozostreni)(roztazeni) barva, (dalsi_stin)

Nejjednodušší stín se vytvoří uvedením povinných parametrů - *horizontální* a *vertikální posun*, které definují posun stínu od objektu, a *barva*, která definuje barvu stínu. Tady se hodí definovat barvu ve tvaru RGBA s nastavením průhlednosti tak, aby stín vypadal realisticky. Ukázka stínu je uvedena ve výpisu 7.

```
box−shadow: 5px 5px rgba(0,0,0,0.3);
```

Výpis 7: Příklad použití box-shadow

Mezi nepovinné vlastnosti patří *rozostření*, které udává, jak bude stín rozostřen a v jakém místě bude přecházet stín do ztracena. Uvedením vlastnosti roztažení se nastavuje *roztažení* stínu do stran. Uvedením klíčového slova *inset* bude stín uvnitř objektu. Stíny lze řetězit tak, že za čárkou následuje další definice stínu a lze tak například nastavit jak vnější, tak i vnitřní stín.

4.1.3 Barevné modely

CSS3 přináší nové barevné modely **RGBA**, **HSL** a **HSLA**. Hlavním důvodem zadávání barvy ve formátu RGBA je definice průhlednosti pomocí alpha kanálu. Odpadá tím řešení průhledných objektů pomocí různých PNG obrázků.

Při zmínce o průhlednosti je důležité uvést vlastnost *opacity*, která rovněž nastavuje průhlednost. Rozdíl oproti RGBA modelu je v rozsahu průhlednosti. Vlastnost opacity zprůhlední celý objekt, RGBA zprůhlední pouze pozadí objektu [9].

Další barevný model je HSL. Tato funkce má tři hodnoty: barva, sytost a světlost. První číslo je nastavení základní barvy z intervalu <0, 360>. Druhá hodnota se udává v procentech a určuje sytost, kde 100% je maximální barevnost. Třetí hodnota je světlost a udává se také v procentech, kde 0% je tmavá barva. HSLA se od HSL liší přidáním průhlednost, podobně jako to je u RGBA. Příklad použití těchto barevných modelů ilustruje výpis 8.

```
background−color = rgba(204, 51, 51, 0.5);      // cervena barva pozadi s pruhlednosti 0,5
background−color: hsla(0, 75%, 80%, 0.5);       // stejne  nastaveni pomoci HSLA
```

Výpis 8: Příklad použití RGBA a HSLA

RGBA i HSLA podporují prohlížeče Firefox, Chrome a Internet Explorer od verze 9.

4.1.4 Přechody

Pro definici vícebarevného pozadí s přechodem slouží v CSS3 vlastnost **linear-gradient** pro rovnoměrný barevný přechod z jedné strany objektu na druhou a **radial-gradient** pro kruhový přechod. Lze použít 2 nebo více barev, mezi kterými bude přechod.

Směr přechodu definují klíčová slova (*to bottom right*, *to right*, *top to bottom*) nebo úhel, kde 0°vede zdola nahoru, 90°zleva doprava a takto to pokračuje po směru hodinových ručiček. Spolu s barvou je možné uvést místo, ve kterém bude daná barva začínat. Příklad lineárního přechodu je ve výpisu 9.

```
background: linear-gradient(45deg, lightgreen, darkgreen 33%);
```

Výpis 9: Příklad linear-gradient

U kruhového přechodu radial-gradient se definuje tvar a velikost přechodu. Výchozí hodnotou je *circle* a mimo to je možné nastavit i *ellipse*. Za tvarem následuje velikost udávající poloměr. U kružnice je to jedna hodnota, u elipsy jsou to hodnoty dvě. Definice barev a umístění přechodů je stejné jako u lineárního přechodu.

4.2 Transformace

Transformace (modul **Transforms**) umožňují dané elementy různě otáčet, zvětšovat i zmenšovat, posouvat a různě deformovat. Transformace tvaru objektu je možné provést jak ve 2D tak i ve 3D prostoru. Prvek, který má být transformován, bude nejprve standardně umístěn na stránku a poté se provede transformace. Pro 2D existují 4 funkce: posun (*translate*), otočení (*rotate*), změna velikosti (*scale*) a zkosení (*skew*).

4.2.1 Translate

Metoda **translate** zajistí posunutí elementu o tolik, kolik je uvedeno v parametrech metody. Porovná-li se tato metoda s použitím `position: absolute` a nastavení umístění elementu pomocí atributů *left* a *top*, může se zdát, že výsledek bude stejný. Výsledný efekt bude stejný, ale transformace translate je přívětivější v případě animace umístění elementu protože x a y souřadnice udávají, o kolik bude element posunut od původního umístění. U absolutního pozicování se udávají absolutní hodnoty. Je tak možné animovat posun elementu, který může být na stránce umístěn kdekoli. Změní-li se pozice elementu, na animaci to nebude mít vliv. Nastavením `transform: translate(100px, 50px)` se posune element o 100 pixelů po ose x a o 50 pixelů po ose y.

Tato funkce má variantu pro transformaci jen po jedné ose – `translateX(x)` a `translateY(y)`.

4.2.2 Rotate

Metoda **rotate** zajistí otočení elementu o danou hodnotu. Zadaná hodnota může být v stupních (*deg*), radiánech (*rad*), gradiánech (*grad*) a kompletních otočeních (*turn*), kde 1 turn = jedno otočení = 360deg. Hodnota může být kladná i záporná, záporným znaménkem se element otočí proti směru hodinových ručiček. Uvedením `transform: rotate(30deg)` se element otočí o 30 stupňů.

4.2.3 Scale

Metoda **scale** zvětší nebo zmenší element o zadaný násobek oproti původní velikosti. Uvede-li se hodnota od 0,0 do 1,0, element se zmenší (0,5 znamená zmenšení o 50%). Zadáním transformace se nezmění pouze šířka a výška elementu, ale i vše, co je uvnitř elementu.

Kromě zápisu scale(n) je možný i zápis scale(x, y), který zvětší nebo zmenší element na ose x a y různě nebo zápis scaleX(x) a scaleY(y), který transformuje element podél jedné osy. Uvedením záporného násobku se element transformuje a navíc se zrcadlově otočí podél zadané osy [10].

4.2.4 Skew

Metoda **skew** definuje zkosení elementu. Hodnotu lze zadat ve stupních. Zadáním jedné hodnoty se element zkosí podél osy x. Zadáním dvou hodnot se element zkosí podél osy x a osy y. Podobně jako u všech metod pro transformaci, i tady existuje varianta v pro definici jen pro jednu osu - skewX(x) a skewY(y).

Transformace skewX(21deg) znamená, že levá a pravá hrana elementu bude vychýlená o 21 stupňů oproti vertikální ose. Podobně u skewY(21deg) bude horní a dolní strana elementu vychýlená o 21 stupňů podél horizontální osy.

4.2.5 Matrix

Různé transformace je možné kombinovat uvedením metody následovanou mezerou a další metodou transformace. Napsáním více vlastností *transform* pod sebou nebude mít očekávaný výsledek. Prohlížeč element transformuje podle posledního zápisu *transform* ve stylopisu. Proto se musí uvést metody do jedné vlastnosti *transform*, jako například ve výpisu 10. Dalším způsobem, jak uvést kombinace transformací, je metoda matrix() a matrix3d().

```
transform: scale(1.5) skew(−15deg);
```

Výpis 10: Kombinace transformací

Prohlížeč tyto kombinace zpracuje a výsledná transformace je uložena v podobě matice 3x3. U 2D transformací má metoda matrix() 6 hodnot, poslední 3 hodnoty matice jsou vždy (0, 0, 1) a neuvádějí se. Použije-li se jQuery pro výpis CSS vlastnosti *transform*, zobrazí se právě tato matice. Má-li element nějakou 3D transformaci, zobrazí se matice v podobě matrix3d().

Výpočet transformační matice je složitější a přesahuje rámec této práce. Existují ale nástroje, které vypočítají hodnoty matice podle vložené transformace.

4.2.6 Transform-origin

Vlastnost **transform-origin** specifikuje souřadnice bodu, ze kterého transformace vychází. Výchozí hodnota je střed elementu (50%, 50%). Tento bod udává střed transformo-

vaného elementu. Jednoduchou demonstrací je transformace rotate(), která otáčí element a bod určený vlastností transform-origin je střed, kolem kterého se element otáčí.

Hodnoty lze uvádět v procentech, pixelech nebo pomocí klíčových slov *top*, *left*, *right*, *bottom* a *center*.

4.3 Animace

Mezi hlavní přínosy CSS3 se řadí animace a přechody [11, 12]. Moduly **Animations** a **Transitions** umožňují plynulou změnu stavu objektu bez použití Flash technologie nebo JavaScriptu. Animace v CSS3 mají oproti těmto technologiím výhodu kratšího kódu, který se projeví na rychlosti načítání a rychlosti zobrazení v prohlížeči. I z pohledu výkonnosti je na tom CSS3 lépe, než animace pomocí jQuery, zejména při animaci velkého množství objektů. CSS3 animace může využívat hardwarovou akceleraci, zatímco jQuery musí přepočítat všechny styly objektů a aplikovat je na každý animovaný objekt zvlášť [13]. Hlavním důvodem, proč CSS3 animace vítězí v rychlosti, je způsob výpočtu hodnot. Zatímco jádro prohlížeče je napsané v kompilovaném programovacím jazyce C++, a tudíž vykonání výpočtu je rychlé, JavaScript je z řad interpretovaných jazyků, které budou vždy pomalejší [14].

První možností, jak animaci vytvořit, je pomocí přechodů (Transitions), které umožní vytvářet jednoduchý plynulý přechod z jednoho stavu do druhého. Inicializace animace může být způsobena například událostí kliku myší.

Modul Animations vytvářejí oproti transformacím složitější, plnohodnotné animace. Nad tvorbou animace má kodér daleko větší kontrolu. Pomocí klíčových snímků (**Keyframes**) je možné definovat stavy elementu, mezi kterými prohlížeč vypočítá přechody.

4.3.1 CSS3 Keyframes

Keyframes určuje podobu a průběh samotné animace. Jednotlivé klíčové snímky určují, jak bude element v danou chvíli vypadat. Jednotlivé klíčové snímky se uvádějí do těla zavináčové funkce @keyframes, která je základem animace. Funkce musí mít své jméno, pomocí kterého se následně nadefinovaná animace zavolá. Nejjednodušší animace se začátkem a koncem má 2 klíčové snímky, které mohou být označeny klíčovými slovy *from* a *to*. Daleko použitelnější je uvádění průběhu animace v procentech. Lze tak vytvořit mnoho stavů které vytvoří různě složité animace. Klíčová slova *from* a *to* lze tedy přepsat na ekvivalent 0% a 100%. Počáteční snímek 0% nemusí být uveden. V takovém případě si animace vezme počáteční stav ze základní definice elementu.

Pro spuštění animace stačí nadefinovat vlastnost **animation-name** a **animation-duration** v CSS stylopisu elementu. Vlastnost animation-name obsahuje jméno animace, které je uvedeno v definici @keyframes. Tímto způsobem je volána samotná animace, která se provede a bude trvat časovou dobu specifikovanou v sekundách nebo milisekundách pomocí vlastnosti animation-duration. Ukázka definice klíčových snímků a spuštění animace je ve výpisu 11. Element se třídou *.object1* bude měnit barvu z modré k žluté a následně k červené po dobu dvou sekund.

Vlastnosti animation-name a animation-duration jsou povinné minimum pro spuštění animace. Existují i další nepovinné vlastnosti, které různě upravují nastavení animace.

```
@keyframes jmeno-animace {
    0%   {background: blue}
    50% {background: yellow}
    100% {background: red}
}

.object1 {
    animation-name: jmeno_animace;
    animation-duration: 2s;
}
```

Výpis 11: Ukázka použití @keyframes

animation-delay umožňuje odložit spuštění animace. Je to tedy prodleva mezi akcí, která animaci spouští a animací samotnou. Udává se v sekundách nebo milisekundách. Přednastavena je nulová hodnota. Uvedením záporné hodnoty se animace spustí hned, ale přeskočí se doba, rovnající se absolutní hodnoty daného záporného čísla.

animation-direction nastavuje směr animace. Uplatní se například v případě, kdy se animace opakuje. Výchozí hodnota *normal* určuje, že opakovaná animace skočí na začátek (0%) a poté se opakuje. Opakování lze navázat na sebe, stačí uvést hodnotu *alternate*, která zajistí, že každé sudé opakování má průběh pozpátku (od 100% do 0%).

Další možnosti poskytují klíčová slova *reverse* – animace poběží od konce (100% - 0%) a *alternate-reverse* – animace na sebe navazuje a začíná se od konce (první iterace 100% - 0%, druhá iterace 0% - 100%).

animation-iteration-count nastavuje počet opakování. Hodnota se udává celým číslem nebo klíčovým slovem *infinite*, které zajistí, že počet iterací animace bude nekonečný.

animation-play-state ovládá průběh animace. Do průběhu animace lze určitým omezeným způsobem uživatelsky zasahovat. Vlastnost animation-play-state animaci pozastaví klíčovým slovem *paused* a znovu spustí klíčovým slovem *running*. Opětovné spuštění naváže v okamžiku předchozího zastavení. Vlastnost se dá použít přímo ve stylu elementu nebo k ní lze přistupovat pomocí JavaScriptu. Lze tak do určité míry pozastavovat animaci pomocí JS událostí. Ukázka je uvedena ve výpisu 12.

```
<script>
    // Manipulace s animaci pomoci JavaScriptu
    var obj = document.getElementById("object");
    obj.style.animationPlayState="paused"
</script>
```

Výpis 12: Ukázka manipulace s animací pomocí JavaScriptu

animation-fill-mode určuje, v jakém stavu bude animovaný element po skončení animace. Výchozí hodnotou je *normal*. V tomto stavu se provede animace a po skončení se vše vrátí do původního stavu. Před a po průběhu animace se na objekt neaplikují žádné CSS vlastnosti z klíčových snímků. Toto chování je někdy nežádoucí a je možné ho změnit pomocí následujících klíčových slov:

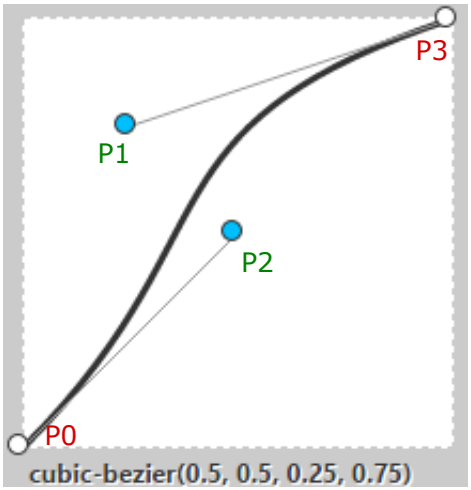
- *backwards* - Pro počáteční zobrazení před samotnou animací se použijí vlastnosti z počátečního klíčového snímku. Tyto hodnoty přepíšou původní hodnoty vlastností elementu.
- *forwards* - Po skončení animace zůstane objekt ve stejném stavu, jaký popisuje poslední klíčový snímek a původní vlastnosti elementu se neaplikují.
- *both* - Aplikuje se nastavení *backwards* i *forwards*.

animation-timing-function udává, jaká funkce se má použít pro výpočet průběhu animace. Nastavením této vlastnosti lze například zpomalit animaci na začátku a ke konci ji zrychlovat nebo nastavit *lineární průběh* a animace bude mít po celou dobu stejnou rychlost. Vlastnost může být nadefinována pomocí klíčových slov, nebo funkcí.

- *ease* - Výchozí hodnota. Animace se zrychluje a konci se zpomalí. Důvodem použití je přirozenější pohyb než má lineární průběh.
- *linear* - Lineární průběh. Animace má stejnou rychlost po celou dobu.
- *ease-in* - Pomalý start.
- *ease-out* - Pomalý konec.
- *ease-in-out* - Kombinace vlastností *ease-in* a *ease-out*.

Tato klíčová slova nejsou ničím jiným, než zkratkou předem nadefinovaných funkcí. Například nastavení *ease-in* je ekvivalentní funkci cubic−bezier(0.42, 0, 1, 1).

cubic-bezier je funkce, která definuje průběh animace. Funguje na základně kubické Bézierovy křivky. Tímto lze detailně nastavovat rychlost animace v jednotlivých fázích. Funkce má podobu cubic−bezier(x1, y1, x2, y2), kde x a y jsou souřadnice tečných bodů. Souřadnice **P0** jsou vždy [0, 0] a souřadnice **P3** jsou [1, 1]. Vzniklé tečny popisují Bézierovu křivku, která určuje hodnoty animované vlastnosti v čase. Vytvořený nástroj implementuje nastavení průběhu animace pomocí jednoduchého táhnutí bodů *P1* a *P2*, které jsou zobrazené na obrázku 1.



Obrázek 1: Nastavení funkce cubic-bezier v aplikaci

4.3.2 Zkrácení definice animace

Všechny vlastnosti pro animaci je možné zkrátit do jedné vlastnosti **animation**. Syntaxe vlastnosti animation je uvedena ve výpisu 13.

```
animation: name duration timing-function delay iteration-count direction  fill  -mode play-state;
```

Výpis 13: Syntaxe vlastnosti animation

Ukázka zkrácení definice animace z několika řádků do jediného řádku je uvedena ve výpisu 14. Lze uvést pouze některé vlastnosti s tím, že název animace a doba animace jsou povinné. Příkaz animation: mymove 5s infinite stačí k tomu, aby se spustila animace definovaná pomocí klíčových snímků pojmenovaných jako mymove, trvající 5 sekund s nekonečným počtem iterací.

```
.object1 {
    animation-name: jmeno_animace;
    animation-duration: 2s;
    animation-delay: 1s;
    animation-iteration-count: 3;
    animation-timing-function: ease-in;
    animation-direction: alternate;
}

// predesly zapis animace lze zapsat na jeden radek
.object1 {
    animation: jmeno_animace 2s ease-in 1s 3 alternate;
}
```

Výpis 14: Zkrácení animace do vlastnosti animation

4.4 Přechody

Při zachycení nějaké události nad elementem lze měnit skokově CSS vlastnosti daného elementu. Typickým příkladem je událost :hover, která aplikuje požadované vlastnosti při najetí myší. Přechody definované modulem **Transitions** řeší plynulost této změny.

Modul Transitions slouží k jednoduché animaci vlastností HTML elementu z jednoho stavu do druhého. Animace se aktivuje kdykoli, kdy se element dostane do jiného stavu. Příkladem je najetí myší na element nebo kliknutí na element. Pro spuštění animace stačí specifikovat transformaci přímo do CSS definice elementu a následně pomocí pseudo tříd :hover nebo :focus změnit hodnotu vlastnosti, která má být animovaná. Kód uvedený ve výpisu 15 zajistí, že po najetí myší na element se třídou *.object1*, se plynule změní šířka elementu (vlastnost *width*) ze 100px na 300px. Doba této animace bude 2 sekundy. Vlastnost **transition** nastavuje přechod a bez ní by byla změna velikosti skoková. První parametr určuje název vlastnosti, která má být animovaná, druhý parametr určuje dobu trvání přechodu, třetí parametr představuje časovou funkci a poslední, čtvrtý parametr, nastavuje zpoždění. Tyto vlastnosti lze zapsat i samostatně pomocí následujících vlastností. Ukázka rozepsaných vlastností je ve výpisu 16.

```
.object1 {
    width: 100px;
    -webkit-transition: width 2s;
     transition : width 2s;
}

.object1:hover {
    width: 300px;
}
```

Výpis 15: Ukázka použití přechodu na vlastnosti width

transition-property určuje, na jaké CSS vlastnosti se přechod vztahuje. Jestliže má element nastaveny vlastnosti *width* a *background-color*, které se v pseudo třídě :hover mění, a transition-property má nastaveno **width**, animuje se pouze tato vlastnost - šířka. Barva pozadí se změní skokově, protože není zahrnuta v přechodu. Více vlastností lze oddělit čárkou. Pro uvedení všech možných vlastností se používá klíčové slovo *all*.

transition-duration specifikuje, podobně jako u vlastnosti animation-duration, jak dlouho má přechod trvat. Hodnota může být zadávána v sekundách nebo milisekundách.

transition-timing-function je totožná s vlastností animation-timing-function, která je popsána v kapitole "CSS3 Keyframes"na straně 17. Tato vlastnost specifikuje rychlost průběhu animace.

transition-delay je opět totožná s vlastností animation-delay a specifikuje zpoždění přechodu. Má-li se čekat 2 sekundy na start přechodu, který se spustí najetím myší na

element, kurzor musí být nad tímto elementem minimálně 2 sekundy, jinak se přechod nespustí. Změní-li se stav elementu v průběhu vykonávání přechodu, přechod se přeruší a element se začne animovat k původnímu stavu. Je-li nastavena prodleva pomocí vlastnosti transition-delay, změna průběhu se projeví až po stanovené době po změně stavu.

```
.object1 {
    transition -property: width, background, height;
    transition -duration: 2s;
    transition -timing-function: linear;
    transition -delay: 2s;
}
```

Výpis 16: Ukázka definice přechodu

U vlastnosti *transform* je možné přechody řetězit. Tímto se může dosáhnout spuštění více animací na elementu v různou časovou dobu. Animace mohou mít různé trvání. Stačí jednotlivé definice oddělit čárkou. Příkladem uvedeným ve výpisu 17 se druhý přechod spustí se sekundovým zpožděním po dokončení prvního.

```
transition : transform .2s, background .2s 3s;
```

Výpis 17: řetězení přechodů

Vlastnost *transition* je podporována v prohlížečích Google Chrome, Mozilla Firefox a Internet Explorer od verze 10.

5 SVG

Scalable Vector Graphics (SVG) je značkovací jazyk vycházející z XML standardu, který popisuje dvojrozměrnou vektorovou grafiku. Jedná se o plnohodnotný vektorový formát pro tvorbu základních geometrických tvarů a křivek. Historie tohoto otevřeného formátu sahá až do roku 2001, širší zastoupení a oblíbenost získal tento formát teprve nedávno. Důvodem byla nedostatečná podpora v prohlížečích Internet Explorer, kde se formát SVG objevil až ve verzi 9 [15].

Obsahem XML kódu je matematický popis křivek, jejich barev a barevných výplní, který umožňuje zobrazení ve vysoké kvalitě v libovolné velikosti. Proto je tento grafický formát využíván hojně tam, kde je potřebná jednoduchá webová grafika nezávislá na rozlišení, jako například vkládání ikon a logotypů na stránky. Mezi hlavní důvody využití a výhody SVG formátu patří:

- Nástup displejů s vysokým rozlišením (jako například *Retina* v Apple produktech) přinesl problémy s bitmapovou grafikou. Tam, kde obrázek vypadal na normálních rozlišeních dobře, ve vysokém rozlišení byl tento obrázek při stejné velikosti rozmazaný. Řešením je použití vektorové grafiky, která je nezávislá na rozlišení.
- Vektorovou grafiku lze libovolně zmenšovat i zvětšovat bez ztráty kvality. Proto se SVG používá v responzivním designu.
- Datová náročnost je menší než u bitmapové grafiky. Pro jednoduchý trojúhelníkový tvar je potřebné popsat 3 body, což je oproti definici pixelů značná úspora.
- SVG se snadno strojově generuje a je dobře čitelný i pro člověka. S tím souvisí i snadná přenositelnost mezi různými programy a nezávislost na platformě.

Mezi nevýhody patří nutnost vytvářet alternativní řešení (tzv. **fallbacky**). Jestliže prohlížeč bez podpory SVG stáhne SVG obrázek a zjistí, že s ním neumí pracovat, je nutné mu poskytnout alternativu ve formě bitmapového obrázku. Další nevýhodou jsou výkonnostně náročnější animace.

5.1 Hlavička SVG dokumentu

SVG je XML formátu, a proto by hlavička dokumentu měla obsahovat XML deklaraci. Na druhém řádku se nachází deklarace typu dokumentu, v tomto případě typu svg. Následuje samotný tag <svg>, který má definovanou velikost a jmenný prostor. Uvnitř tagu jsou vykreslené grafické objekty.

```
<?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="500" height="400" version="1.1" xmlns="http://www.w3.org/2000/svg">
    <!-- definice objektu -->
</svg>
```

Výpis 18: Hlavička SVG dokumentu

Samotné vkládání do HTML dokumentu se řeší několika způsoby. Prvním způsobem je vložení pomocí <**object**> elementu, který vloží SVG dokument pomocí atributu *data*. Výhodou je jednoduché použití fallbacku v případě, že prohlížeč nepodporuje SVG. V takovém případě se zobrazí jako alternativa všechno, co je uvnitř elementu <object>, kam se umístí například odkaz na bitmapový obrázek.

Další možností je přímé vložení elementu <svg> do stránky. Všechny současné verze prohlížečů podporují přímé vložení.

S SVG grafikou lze pracovat podobně jako s jiným obrázkem a lze ho tak vložit odkazem na externí soubor elementem <**img**>. Podobně lze odkazovat na SVG soubor i v CSS stylopisech.

5.2 Základní tvary

Základním způsobem pro tvorbu vektorové grafiky je definování cest. Cesta se může skládat z úseček, kruhových a eliptických oblouků a křivek. Tyto segmenty jsou popsány pomocí příkazů v atributu *d*, který je součástí elementu <path>. Definice cesty může vypadat následovně: <path d="M150␣0␣L75␣200␣L225␣200␣Z"/>.

Z důvodu co nejmenší datové náročnosti se jednotlivé segmenty udávají jejich jednopísmennými zkratkami následované argumenty, které popisují samotnou cestu. Uvede-li se malé písmeno, uvedené souřadnice jsou relativní vzhledem k předchozímu umístění grafického kurzoru. Velké písmeno udává absolutní pohyb.

- **lineto** (L) - úsečka na souřadnice *x, y* definovné v argumentu
- **moveto** (M) - pohyb na souřadnice bez kreslení
- **horizontal** (H) - horizontální úsečka
- **vertical** (V) - vertikální úsečka
- **Z** - způsobí uzavření cesty úsečkou, která vede z aktuálního bodu do bodu představující začátek.

Pomocí cest lze graficky zobrazit základní i složitější tvary. Popisovat elementární tvary pomocí cest by bylo náročnější, a tak jsou pro základní tvary definovány speciální elementy, uvedeny níže.

úsečka K nakreslení úsečky se používá element <line>, který má atributy *x1*, *y1*, určující souřadnice začátku a *x2*, *y2*, které určují souřadnice konce úsečky. Velikost a barvu úsečky je možné definovat v atributu *style*, jak je ukázáno ve výpisu 19.

```
<line x1="178" y1="102" x2="236" y2="37" style="stroke:rgb(255,20,20);stroke-width:1;stroke-
    opacity:0.8;" />
```

Výpis 19: Ukázka použití elementu <line> a stylování

obdélník Element <rect> určuje pozici obdélníku pomocí atributů *x* a *y*. Velikost obdélníku je určena atributem *width* pro šířku a *height* pro výšku. Existují i atributy *rx* a *ry* určující zaoblení hran obdélníku.

```
<rect x="0" y="0" width="419" height="430" style="fill:rgb(30,30,30);" />
```

Výpis 20: Ukázka použití elementu <rect>

kružnice Pro vykreslení kružnice slouží element <circle>, který má atributy *cx* a *cy* určující souřadnice středu kružnice. Atribut *r* definuje poloměr kružnice.

```
<circle cx="314" cy="106" r="5" style=" fill :rgb(200,200,200);" />
```

Výpis 21: Ukázka použití elementu <circle>

elipsa Podobně jako u kružnice, určují atributy *cx* a *cy* střed elipsy. Atribut *cx* definuje horizontální poloměr a *cy* definuje vertikální poloměr elipsy.

```
<ellipse cx="200" cy="80" rx="100" ry="50" style=" fill :yellow;stroke:red;stroke-width:2" />
```

Výpis 22: Ukázka použití elementu <ellipse>

polygon Element <polygon> slouží k vytvoření mnohoúhelníku. Atribut *points* definuje souřadnice každého rohu mnohoúhelníku, kde jednotlivé souřadnice jsou oddělené mezerou. Tyto body jsou spojeny úsečkou a tvoří uzavřený tvar (poslední bod se spojí s prvním bodem).

```
<polygon points="200,10 250,190 160,210" style="fill:lime;stroke:purple;stroke-width:1" />
```

Výpis 23: Ukázka použití elementu <polygon>

polyline Element <polyline> vykreslí grafický útvar složený z úseček. Na rozdíl od mnohoúhelníku se vytvořená cesta neuzavře.

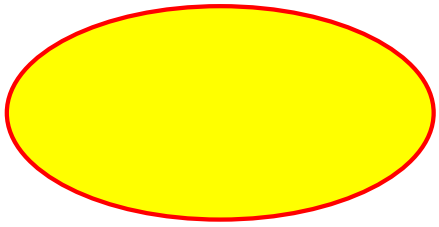
```
<polyline points="20,20 40,25 60,40 80,120 120,140 200,180" style="fill:none;stroke:black;
    stroke-width:3" />
```

Výpis 24: Ukázka použití elementu <polyline>

5.3 Aplikace stylů na SVG objekty

Vzhled SVG objektu lze upravovat pomocí CSS stylů. Pro lepší pochopení a názornou ukázku je možné využít výpisu 22, který vykreslí obrázek 2.

Způsob použití a syntaxe CSS je shodná s použitím v HTML stránkách. Jediným rozdílem je, že u SVG jsou používány vlastnosti typické pro tento formát. Jedná se zejména o tyto vlastnosti:



Obrázek 2: Objekt vykreslený kódem z výpisu 22

- **fill** určuje barvu výplně. V ukázce je nastavena žlutá barva zápisem `fill : yellow`.
- **stroke** definuje barvu okraje. V ukázce je barva výplně červená pomocí zápisu `stroke: red`. Definice barev je možné zapisovat pomocí klíčových slov, nebo pomocí šestnáctkové soustavy.
- **stroke-width** určuje šířku okraje, v ukázce je šířka nastavena na 2px.

V ukázce je použitá tzv. **inline deklarace stylů** – definice vlastností je v atributu *style*. Kromě inline deklarace je možné vlastnosti uvést i v interním seznamu stylů 25. Je možné využít třídy podobně jako v u HTML elementů. Styly by měli být obaleny do sekce *CDATA*, kterou parser XML ignoruje.

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
    <style type="text/css" >
        <![CDATA[
        circle .myCircle {
            stroke: #006600;
             fill :   #00cc00;
        }
        ]]>
    </style>

    <circle  class="myCircle" cx="40" cy="40" r="24"/>
</svg>
```

Výpis 25: Ukázka interního seznamu stylů v SVG

5.3.1 Seskupování objektů

Objekty,které mají stejné vlastnosti lze seskupit párovým elementem <g>, který slouží k vytváření skupin SVG objektů. V ukázce 26 jsou seskupeny kružnice, které mají stejné vlastnosti uvedené v elementu <g>.

```
<svg width="100%" height="100%" viewBox="0 0 95 50" xmlns="http://www.w3.org/2000/svg">
<g stroke="green"  fill ="white"  stroke−width="5">
<circle  cx="25" cy="25" r="15" />
<circle  cx="40" cy="25" r="15" />
<circle  cx="55" cy="25" r="15" />
<circle  cx="70" cy="25" r="15" />
</g>
</svg>
```

Výpis 26: Seskupení objektů v SVG [16]

5.4 Transformace SVG

V kapitole "Transformace"na straně 15 je ukázáno, jak lze HTML elementy různě transformovat pomocí CSS vlastnosti *transform*. Použije-li se element **<img>** k zobrazení SVG dokumentu pomocí odkazu, lze takto vytvořený obrázek rovněž transformovat. Například zápisem **<img src**="logo.svg"**width**="100"**height**="100"**style**="transform: rotate(30deg)"> se celý SVG dokument otočí o 30°.

Podobným způsobem je možné transformovat i samotné objekty přímo v SVG dokumentu. Jeden objekt je otočený, zatímco druhý může být zvětšený. Stačí uvést vlastnost *transform* v atributu style, jak je to ukázáno ve výpisu 27.

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
   <rect  x="50" y="50" height="110" width="110" style="stroke:#ff0000; fill: #ccccff"
      transform="translate(30, 0) rotate(45, 50, 50)">
   </rect>
</svg>
```

Výpis 27: Ukázka transformace SVG objektu

Metody pro transformaci jsou již známé z CSS transformací. Jediným rozdílem je způsob aplikace metody rotate(), která CSS objekt otáčí ve výchozím stavu kolem středu elementu. SVG objekt je otáčen kolem bodu [0, 0]. Tento výchozí stav lze změnit uvedením druhého a třetí parametru metody rotate().

Transformace, stejně jako i samotné zobrazení SVG objektů, je výkonnostně náročnější. Známou a v praxi používanou metodou je rozdělit SVG do menších částí a tyto části následně obalit elementem ** ** a na tento kontejner následně aplikovat CSS transformaci. Tímto způsobem se například u objektu <line> zrychlila frekvence zobrazení z 12 FPS až na 60 FPS [17].

5.5 Animace SVG

SVG nabízí vlastní možnosti animace mnoha vlastností, například umístění geometrických tvarů, jejich barvu, velikost nebo průhlednost. Princip je totožný s animací HTML elementů pomocí CSS3, definují se klíčové snímky, které představují mezní stavy geometrických tvarů a průběh animace se dopočítává automaticky. Přechod mezi jednotlivými klíčovými snímky může být skokový, lineární nebo definovaný Bézierovými křivkami.

Animace se realizují pomocí pěti různých elementů, které jsou vnořené v elementu představující grafický objekt v SVG.

<animate> má několik atributů, pomocí kterých se určuje, jaká vlastnost se má animovat, jak dlouho má trvat animace a v jakém rozsahu hodnot. Dál je možné nastavit počet opakování animace a co se stane, když animace proběhne. Ukázku animovaného objektu popisuje kód ve výpisu 28.

```
<circle  cx="30" cy="30" r="25" style="stroke: none; fill : #0000ff;">
   <animate attributeName="cx"
            from="30"  to="470"
            begin="0s" dur="5s"
            fill  ="remove"
            repeatCount="indefinite"
   />
</circle>
```

Výpis 28: Ukázka animace SVG pomocí <animate>

Element <animate> obsahuje několik atributů, kterými se definuje samotná animace:

- **attributeName** určuje, jaká vlastnost se má animovat.
- **from** definuje počátečního stavu hodnoty animované vlastnosti.
- **to** definuje koncový stav hodnoty vlastnosti. V ukázce 28 je objekt animován tak, že se posune z počátečního umístění na hodnotu 470 po ose x.
- **begin** představuje čas, určující, kdy se animace spustí. Nemusí se jednat nutně o časovou hodnotu, možnou hodnotou je i klíčové slovo *click*, které představuje událost kliknutí na objekt. Tím se animace spustí po provedení události. Uvedením kombinace (*click+5s*) se animace spustí 5 sekund po kliknutí na objekt. Je možné určit i více podmínek spuštění (*mouseover;10s*). V tomto případě se animace spustí 10 sekund po načtení stránky nebo po přejetí myší přes objekt [18].
- **dur** definuje dobu trvání animace
- **fill** definuje, co se stane s objektem po skončení animace. Klíčové slovo *remove* vrátí objekt do původního stavu a klíčové slovo *freeze* ponechá objekt v konečné poloze.
- **repeatCount** určuje celým číslem počet opakování animace. Uvedením klíčového slova *indefinite* se animace bude provádět neustále.

<animateTransform> se využívá pro animaci transformací. Transformace jsou natolik specifické, že pro jejich animace nestačí použít pouze element <*animate*>. Způsob použití je totožný s elementem <animate>, jen se navíc musí přidat atribut *type*, který popisuje typ transformace. V ukázce kódu ve výpisu 29 je animovaná rotace obdélníku o 360°s dobou trvání 10 sekund a nekonečným opakováním.

```
<rect x="20" y="20" width="100" height="40" style="stroke: #ff00ff; fill :none;" >
   <animateTransform attributeName="transform"
                     type="rotate"
                     from="0 100 100" to="360 100 100"
                     begin="0s" dur="10s"
                     repeatCount="indefinite"
   />
</rect>
```

Výpis 29: Ukázka animace SVG pomocí <animateTransform>

<animateColor> realizuje animace z jedné barvy do druhé. Hodnota atributu *from* představuje počáteční barvu, uvedenou například ve formátu **rgb**. Atribut *to* definuje konečnou barvu. Pomocí atributu *attributeName* ve výpisu 30 se definuje, že animovat se bude barva výplně.

```
<circle cx="80px" cy="80px" r="50px" fill ="red" stroke="black">
   <animateColor
         begin="click"
         attributeName=" fill "
         dur="2s"
         from="rgb(255,0,0)"
         to="rgb(0,0,255)"
          fill ="freeze"
   />
</ circle >
```

Výpis 30: Ukázka animace barvy SVG objektu pomocí <animateColor>

<animateMotion> slouží pro pohyb objektu po ploše. Objekt se pohybuje po dráze, která je definovaná atributem *path*. Je tak možné sestavit libovolně složité křivky, po kterých se bude objekt pohybovat. Způsob definice cesty je již popsán v kapitole "Základní tvary"na straně 24.

<set> slouží k nastavení vlastnosti na určitou hodnotu v určitém čase. K pochopení bude nejlépe sloužit ukázka 31, ve které se po 5 sekundách po načtení stránky zvětší skokově poloměr kružnice na 100 jednotek. Neproběhne tady žádná plynulá animace.

```
<circle cx="30" cy="30" r="100" style="stroke: none; fill : #0000ff;">
   <set attributeName="r" attributeType="XML" to="100" begin="5s"></set>
</ circle >
```

Výpis 31: Ukázka skokové změny vlastnosti SVG objektu

6 Návrh a implementace nástroje pro webovou animaci

Jedním z hlavních cílů této aplikace bylo přinést prvky z desktopových aplikací podobného zaměření, do prostředí prohlížeče s pomocí využití technologie HTML5. Aplikace se skládá pouze z klientské části, která je napsaná v JavaScriptu.

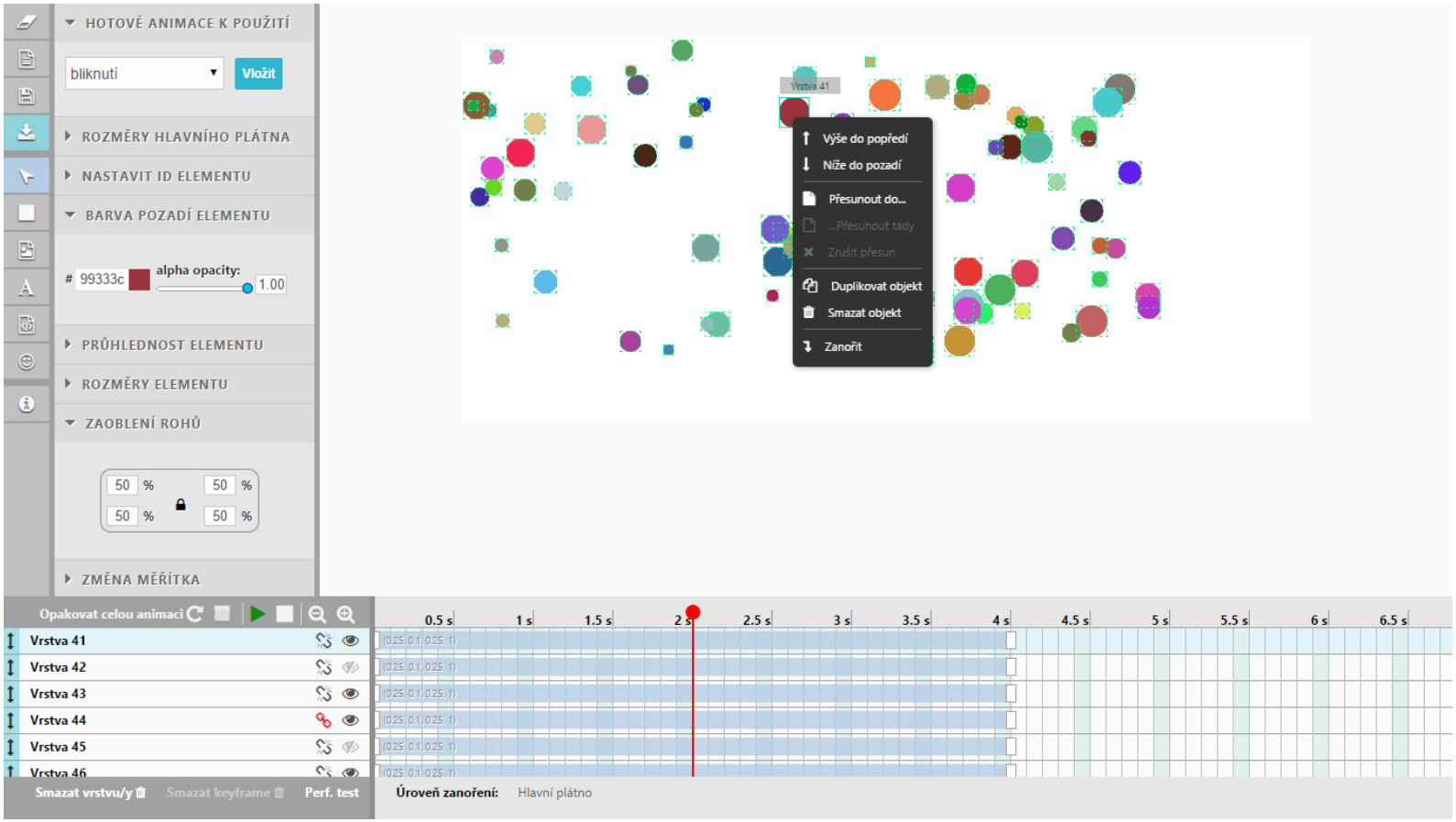
K vytvoření nástroje pro tvorbu animací bylo nutné prozkoumat stávající řešení a navrhnout vhodné rozvržení a způsob práce s nástrojem, s ohledem a využitím stávajících řešení. Důvodem bylo to, aby uživatel, který poprvé aplikaci použije, rámcově věděl, kde se nachází, co s nástrojem může provádět a co znamenají jednotlivé prvky aplikace. Proto bylo využito rozmístění těchto prvků, které je typické pro aplikace podobného zaměření. Uživatel, který má zkušenosti s jinou aplikací, jako je například *Google Web Designer*, tak ví, co má očekávat od jednotlivých funkcí a jak je použít.

Na obrázku 3 je celkový pohled na aplikaci, která se z větší částí skládá z *plátna*, na který se umísťují jednotlivé prvky k animaci. Vkládání prvků a manipulace s nimi umožňuje *panel nástrojů* na levé straně. Jednotlivé ikony znamenají odlišné funkce aplikace – **vytvoření nového projektu**, **uložení a načtení** animace, nástroj pro **manipulaci prvků** na plátně, nástroj pro **vytvoření nového kontejneru**, nástroj pro **vložení obrázku**, nástroj pro **vložení textu**, nástroj pro **vložení SVG prvku**, **galerii jednoduchých SVG tvarů** a **vygenerování animace**. Vedle panelu nástrojů je *panel vlastností*, který umožňuje nastavit některé CSS vlastnosti prvků, jako je **velikost**, **barva pozadí**, **průhlednost**, **border-radius**, **3D rotace**, **zkosení**, zvětšení nebo zmenšení objektu pomocí metody **scale** a posun objektu pomocí **3D translate**. Všechny tyto vlastnosti jsou následně animovány podle toho, jak se mění v čase. Navíc je v panelu vlastností možné na již vloženou vrstvu aplikovat uloženou animaci.

V dolní části se nachází *časová osa*. Na časové ose se nachází seznam objektů s klíčovými snímky. Posunutím ukazatele na ose a změnou některé z vlastností objektu se vytvoří nový klíčový snímek na aktuálním místě. Osa poskytuje jednoduché ovládání animace a další nastavení, jako je opakování animace, mazání vrstev, mazání snímků a změna měřítka osy. Na základě uživatelského testování byly následně v časové ose přidány možnosti pro zneviditelnění jednotlivých vrstev a pro hromadnou editaci všech klíčových snímků vrstvy.

6.1 Použité nástroje

Výstupem této bakalářské práce by měla být webová aplikace pracující v prohlížeči. Pro implementaci byl tedy vybrán JavaScript, konkrétně jeho rozšíření **TypeScript**. Jako vývojový nástroj byl použit **Microsoft Visual Studio 2013**. Hlavním důvodem pro toto rozhodnutí bylo propojení dvou užitečných vlastností – *IntelliSence* ve Visual Studiu a *typová kontrola* TypeScriptu. Našeptávání ve Visual Studiu bylo velmi nápomocné a užitečné. Implementace nástroje v čistém JS by byla mnohem složitější. A navíc nebylo nutné instalovat žádné další rozšíření.



Obrázek 3: Vzhled aplikace

6.1.1 TypeScript

K implementaci aplikace pro tvorbu webových animací byl vybrán TypeScript. TypeScript je open-source programovací jazyk vytvořený společností Microsoft jako odpověď na obdobný jazyk Dart od společnosti Google. Na rozdíl od Dart-u je TypeScript rozšířením JavaScriptu. To znamená, že jakýkoliv kód v JavaScriptu je spustitelný i v TypeScriptu. Dart oproti tomu jde cestou úplného oddělení, a i když je možné výsledný kód překompilovat do JavaScriptu, dlouhodobým cílem je nativní implementace jazyka Dart do prohlížečů [19].

Čistý JavaScript má velkou výhodu v tom, že je jednoduchý. Tato výhoda se stane nevýhodou v případě, kdy se JavaScript použije v rozsáhlejších projektech. TypeScript jako rozšíření JavaScriptu poskytuje silnou **typovou kontrolu**, která zajistí kontrolu datových typů proměnných při kompilaci kódu. Pro primitivní datové typy zavádí anotace *number*, *string* a *boolean*. Dalším pomocníkem pro velké projekty je podpora **tříd**, **modulů** a **rozhraní**, které přinášejí prvky objektově orientovaného programování. Lze tak využít **dědičnost** nebo i **generické datové typy**. TypeScript podporuje **hlavičkové soubory**, které obsahují informace o typech a rozhraních již existujících knihoven. Je tak možné používat cizí knihovny napsané v JavaScriptu, jako například jQuery. A navíc je TypeScript jednoduchý na naučení pro toho, kdo už JavaScript alespoň rámcově umí.

6.2 Možnosti aplikace

V aplikaci je možné animovat několik typu objektů, jejichž vložení je možné provést pomocí jednotlivých funkcí v panelu nástrojů:

kontejner vloží prvek v podobě HTML elementu ** **, který se dál v této práci bude označovat jako "kontejner", protože umožňuje seskupovat další prvky. Po dvojkliku na kontejner se změní kontext plátna a další přidané objekty se přidávají právě do tohoto kontejneru. Přehled zanoření je možné kontrolovat pomocí drobečkové navigace v dolní části osy.

obrázek se vkládá z disku uživatele využitím technologie *Drag and Drop*. Po nahrání obrázku si aplikace uloží textovou reprezentaci souboru, kódovanou pomocí **Base64**. Výhodou je, že se nemusela implementovat serverová část aplikace, starající se o ukládání obrázků. Nevýhodou je v tomto případě to, že není příliš efektivní vkládat velmi velké obrázky. To bylo vyřešeno automatickým zmenšením obrázku po nahrání na velikost plátna.

text , kterému lze měnit vlastnosti *font-size*, barvu textu *color* a rodinu písma *font-family* z několika připravených možností.

SVG se vkládá pomocí formuláře pro vložení XML kódu, který reprezentuje SVG obrázek. Alternativou je nahrát SVG soubor z disku. Podobně jako u bitmapových obrázků, je

i zde pro uložení textové reprezentace SVG použito kódování Base64. Další práce s SVG je stejná jako s běžným obrázkem. Lze ho přesouvat, zvětšovat i zmenšovat bez ztráty kvality. Pro zobrazení se používá element <**img**>.

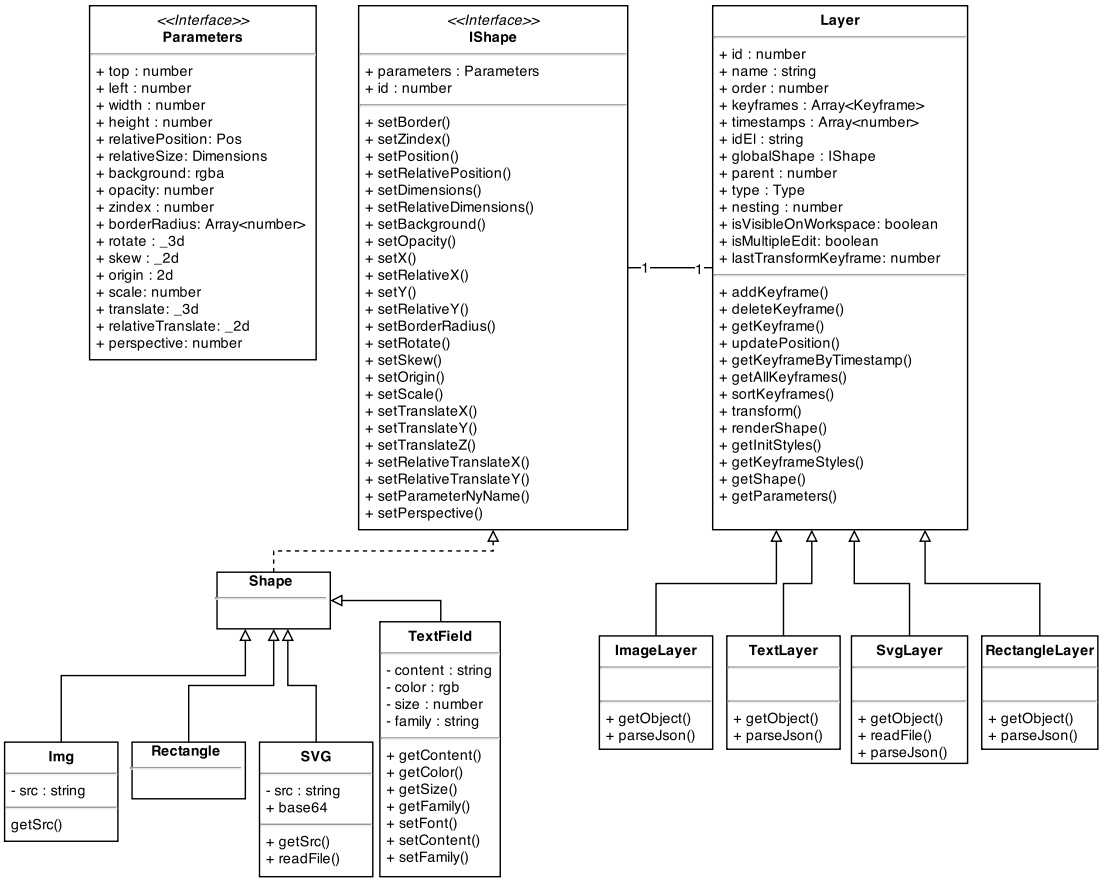
Galerie SVG uchovává připravené jednoduché tvary SVG, které slouží k rychlému vložení na plátno.

6.3 Implementace

Pro implementaci bylo využito předností TypeScriptu a aplikace byla rozdělena do tříd. Po spuštění hlavní třídy se vytvoří instance tří tříd – *Timeline*, *Workspace* a *ControlPanel*. Tyto třídy zajišťují funkčnost jednotlivých prvků aplikace a jsou vzájemně propojeny. Na obrázku 4 je třídní diagram vrstvy a objektu, který se umísťuje na plátno. Jednotlivé vrstvy jsou uloženy do datové struktury typu pole. Instance jedné třídy *Layer* obsahuje jeden globální objekt typu *IShape*, který představuje počáteční stav objektu při vytvoření vrstvy. Tento globální objekt je užitečný zejména v případech, kdy je potřebné uložit vlastnost objektu, který se nemění, nebo se mění v rámci všech klíčových snímků. Jako příklad může sloužit objekt typu *Img*, který uchovává obrázek v textové formě. Tato informace se v rámci klíčových snímku nemění, a proto je s ní manipulováno v globálním objektu. Jednotlivé klíčové snímky jsou uloženy v instanci třídy *Layer* jako pole objektů typu *Keyframe*. Každý objekt *Keyframe* obsahuje právě jednu instanci implementující interface *IShape*, která obsahuje parametry popisující, jak daný objekt vypadá na plátně v daném čase. Časový údaj je spolu s časovou funkcí uložen v objektu *Keyframe*.

6.3.1 Vykreslení a animace objektu

Objekt na plátno lze vložit několika různými způsoby v závislosti na typu objektu. V zjednodušené ukázce kódu ve výpisu 32 je ukázáno vložení kontejneru. Vkládaný objekt je následně uložen do instance třídy *Layer*. Instance vrstvy je uložena do pole, se kterým celá aplikace následně pracuje.



Obrázek 4: Třídní diagram pro objekty na scéně

```
private onDrawSquare(e: JQueryEventObject) {
    // vytvori se novy objekt na platne
    var new_object: JQuery = $('<div>').addClass('shape-helper tmp-shape');
    // zjisti se aktualni pozice kliknuti na platno
    var click_y: number = e.pageY - this.workspaceContainer.offset().top;
    var click_x: number = e.pageX - this.workspaceContainer.offset().left;
    //na novy objekt se aplikuji CSS styly
    new_object.css({ 'top': click_y, 'left': click_x, 'background': 'rgba(' + this.color + ')' });

    //Udalost posouvani mysi
    this.workspaceWrapper.on('mousemove', (event: JQueryEventObject) => {
        // Vypocitaji se nove souradnice v zavislosti na pohybu kurzoru
        var move_x: number = e.pageX - this.workspaceContainer.offset().left;
        var move_y: number = e.pageY - this.workspaceContainer.offset().top;
        var width: number = Math.abs(move_x - click_x);
        var height: number = Math.abs(move_y - click_y);
        var new_x: number, new_y: number;
        new_x = (move_x < click_x) ? (click_x - width) : click_x;
        new_y = (move_y < click_y) ? (click_y - height) : click_y;
        //na objetk na platne se aplikuji nove souradnice
        new_object.css({ 'width': width, 'height': height, 'top': new_y, 'left': new_x });
    }).on('mouseup', (event: JQueryEventObject) => {
        this.workspaceWrapper.off('mousemove');
    });
}
```

Výpis 32: Funkce pro vložení kontejneru táhnutím myší

V situacích vyžadující překreslení celé scény, jako je například zanoření do jiného kontejneru, se postupně iteruje každá vrstva z pole a zavolá se na ní metoda renderShape(), která přijímá parametry *container* (jQuery objekt představující kontejner, do kterého se má daný tvar vykreslit), *position* (aktuální pozice na časové ose), *currentScope* (představuje ID vrstvy, která je aktuálním kontejnerem) a *helper* (představuje pomocný element, který slouží k manipulaci vrstvy). Parametr *currentScope* slouží k zjištění, zda je nutné vykreslovat pro danou vrstvu i již zmíněný pomocný ** ** element. Tyto pomocné elementy se vykreslují jen v aktuálním kontejneru. Metoda renderShape() následně pomocí pozice na časové ose vyhledá příslušný časový snímek a podle parametrů vykreslí objekt na plátno.

Jiným způsobem pracuje další metoda pro vykreslování na plátno - transform(). Tato metoda nepřekresluje objekty, ale pouze upravuje CSS vlastnosti objektů, které jsou metodě předávány v parametru *shape* a *helper*. Parametr *shape* představuje objekt, který má být transformován a parametr *helper* jeho pomocný ** ** element. Volání této metody je ukázáno ve výpisu 33. V metodě se podle aktuální pozice na časové ose vyhledají nejbližší klíčové snímky a pomocí **interpolace** se zjistí aktuální hodnota všech parametrů v intervalu ohraničeném vyhledanými klíčovými snímky. Vypočtené hodnoty vlastností se následně aplikují na objekt. Metoda transform() je volána při každé změně pozice ukazatele na časové ose. Tímto způsobem je dosažen efekt animace.

Animace není nic jiného, než vizualizace vlastnosti, která se mění v čase. Samotný přepočet hodnot vlastností se provádí s přihlédnutím na časovou funkci, která je v po-

době Bézierovy křivky uložena u každého klíčového snímku. Možnost nastavit tuto časovou funkci je důležité, protože tímto může uživatel ovlivňovat rychlost, zrychlení nebo zpomalení animace. Kdyby byla změna pouze lineární, takový pohyb by byl mechanický, postrádající dynamiku a zrychlení.

```
public transformShapes(showHelpers: boolean = true) {
    var currentTimestamp: number = this.app.timeline.pxToMilisec();
    var layers: Array<Layer> = this.app.timeline.layers;
    // iterace vsech vrstev
    layers.forEach((layer, index: number) => {
        //nalezeni objektu a jeho pomocneho elementu na platne
        var shape: JQuery = this.workspaceWrapper.find('.shape[data-id="' + layer.id + '"]');
        var helper: JQuery = this.workspaceWrapper.find('.shape-helper[data-id="' + layer.id + '"]
            ');

        if (layer.isVisibleOnWorkspace) {
            //pokud ma byt dana vrstva viditelna, zobraz ji a proved transformaci
            shape.show();
            if (showHelpers) {
                helper.show();
            }

            //pokud je dana vrstva aktualni kontejner, misto pomocneho elementu najdi <div>
            // zajistujici bile pozadi pod objektem s transparentnim pozadim
            if (layer.id == this.scope) {
                helper = this.workspaceContainer.parent().find('.base-fff');
            }

            //ID aktualne oznacene vrstvy,
            //slouzi k hodnot vlasnosti v panelu vlastnosti pro oznacenou vrstvu
            var currentLayerId = this.workspaceWrapper.find('.shape-helper.highlight').first().
                data('id');

            layer.transform(currentTimestamp, shape, helper, currentLayerId, this.app,
                showHelpers);
        } else {
            //pokud vrstva viditelna byt nema, schovej objekt a pomocny element
            shape.hide();
            helper.hide();
        }
    });
}
```

Výpis 33: Aktualizace vlastností objektu

6.3.2 requestAnimationFrame

K dosažení efektu animace v prostředí JavaScriptu je nutné vždy po určité době překreslit scénu. K tomu se může použít časovací smyčka, vytvořená pomocí metod *setInterval()* nebo *setTimeout()*. Lepší způsob překreslování nabízí další z nových technologií HTML5, funkce **requestAnimationFrame**, která umožňuje plynuleji a s větším výkonem vykreslovat animace. Hlavní výhoda spočívá v tom, že prohlížeč optimalizuje různé součásti

animací do jednoho cyklu překreslování. Další výhodou je zastavení animace, pokud je prohlížeč minimalizován nebo je přepnuto do jiného okna, což s sebou nese snížení zátěže. Funkce ve výchozím nastavení používá k vykreslování scény 60 FPS.

Pro implementaci byla použita technika *aktualizace plátna v závislosti na uplynulém čase*, která spočívá ve výpočtu času v milisekundách, který uplynul od posledního vykreslení. Následně se plátno vykreslí v pozici, která je závislá na tomto rozdílu časů.

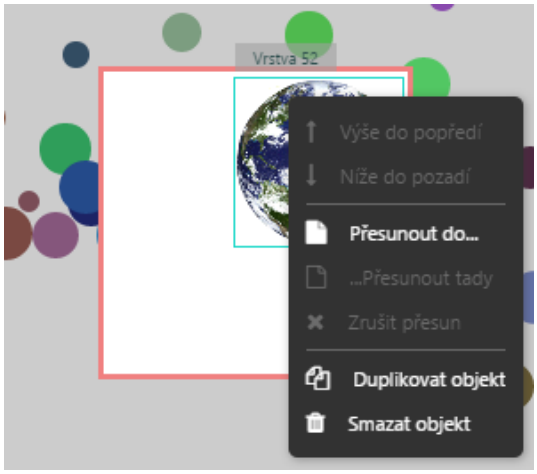
Ve výpisu 34 je zkrácená část kódu, která se stará o chod animace na plátně. Kromě funkce **requestAnimationFrame()**, využívá i funkce **cancelAnimationFrame()**, která slouží k zastavení animace. Pozice ukazatele je při každém překreslení aktualizována v závislosti na době, která uplynula od posledního překreslení. Podle pozice ukazatele jsou následně aktualizovány hodnoty vlastností objektů na plátně.

```
var time;
this.start = new Date();
var draw = () => {
    this.playInterval = requestAnimationFrame(draw);
    var now = new Date().getTime();
    var dt = now - (time || now);
    time = now;
    //absoluteMax zde predstavuje dobu cele animace
    //absoluteMaxPx predstavuje misto na casove ose, kde animace konci
    this.pointerPosition += (absoluteMaxPx / absoluteMax) * dt;
    if (this.pointerPosition >= absoluteMaxPx)
        cancelAnimationFrame(this.playInterval);
    this.pointerEl.css('left', this.pointerPosition - 1);
    //prekresleni objektu na scene
    this.app.workspace.transformShapes(false);
}
draw();
```

Výpis 34: Použití requestAnimationFrame k překreslování scény

6.3.3 Seskupování prvků

Seskupování objektů do skupin je jedna z důležitých vlastností aplikace. Bylo nutné vyřešit způsob, jakým se vložené objekty mohou seskupovat do jednoho objektu, se kterým se následně bude pracovat jako s jednou vrstvou. Dále bylo nutné implementovat související funkce pro přesouvání objektů z jednoho kontejneru do druhého a mazání kontejneru a příslušných vrstev obsažených v daném objektu. Pro uchování právě aktivního plátna byla použita proměnná **scope**, která uchovává jedinečné ID vrstvy. Tato vrstva představuje aktivní kontejner, ve kterém se pracuje a do kterého se vkládají nové objekty. Je důležité zmínit, že uchovávat jiné objekty může pouze objekt typu kontejner. Počet zanoření není omezen a lze tak do kontejneru vkládat i další kontejnery. Zanoření se provádí dvojklikem na kontejner a návrat o úroveň zpět dvojklikem mimo aktuální kontejner.



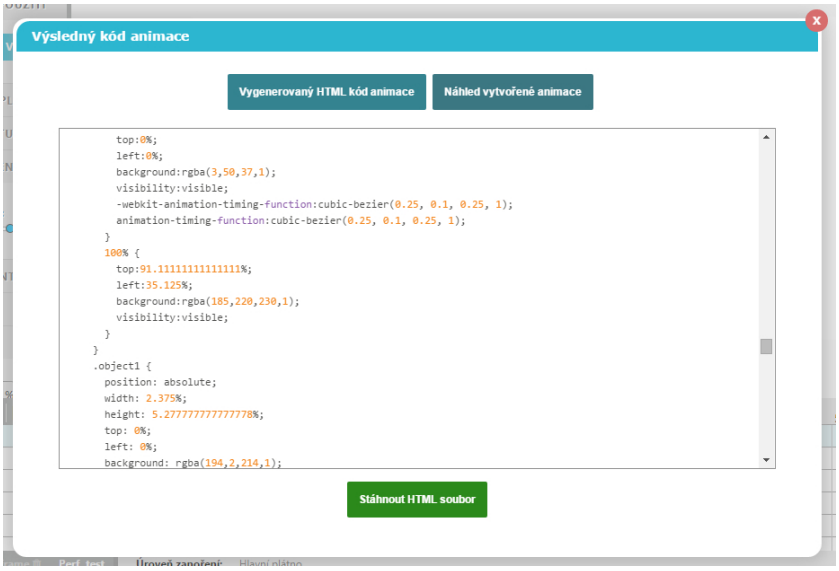
Obrázek 5: Kontejner s kontextovou nabídkou

Na obrázku 5 je znázorněn aktuální kontejner s výrazným červeným ohraničením, který obsahuje jeden objekt. Všechny další objekty, které jsou mimo tento kontejner, jsou v pozadí za šedou vrstvou. Po dvojkliku na tuto šedou vrstvu se aplikace dostane na hlavní plátno. Přesun objektů do jiných kontejnerů nebo do kořenového plátna je realizován pomocí kontextového menu položkami **"Přesunout do..."** a **"...Přesunout tady"**. Nadřazený kontejner dané vrstvy je uchováván ve třídě *Layer* v proměnné **parent**, představující ID nadřazeného kontejneru. Proměnná **nesting** určuje stupeň zanoření objektu. Změnou těchto proměnných se docílí přesunutí objektů do jiných kontejnerů. Proměnná **parent** je dále využívána například k mazání objektu a jeho potomků. Funkce zajišťující mazání je ve výpisu 35 a využívá rekurze.

```
private deleteLayer(index: number) {
    var deletedLayer: Layer = this.layers[index];
    this.layers.splice(index, 1);

    // najdi potomky
    for (var i: number = this.layers.length - 1; i >= 0; i--) {
        if (this.layers[i].parent == deletedLayer.id) {
            this.deleteLayer(i);
        }
    }
}
```

Výpis 35: Mazání vrstev



Obrázek 6: Ukázka okna s výsledným kódem

6.3.4 Vygenerování animace

Efekt animace je na pracovním plátně dosažen pomocí metody transform(), uvedené v kapitole "Vykreslení a animace objektu"na straně 33. Výsledná vygenerovaná animace je dosažena pomocí **CSS3 Keyframes**. O výpis a zobrazení výsledné animace se stará třída *GenerateCode*. Tato třída využívá metody getObject(), getInitStyles () a getKeyframeStyle() ve třídě *Layer*. Zavoláním metody getObject() se získají elementy objektů, getInitStyles () vrací CSS styly objektu a metoda getKeyframeStyle() vrátí CSS styly klíčového snímku. Všechny tyto informace jsou v textové podobě. Poskládáním jednotlivých kousků vznikne souvislý výsledný kód animace. Výsledný kód je optimalizován, takže se v jednotlivých klíčových snímcích zbytečně nevyskytují vlastnosti, které se nemění. Ukázka kódu je zobrazena na obrázku 6. V záložce **Náhled vytvořené animace** je k dispozici živý náhled toho, jak animace vypadá. Je využito elementu <**iframe**>, do kterého byl vložen vygenerovaný kód.

Jednotlivé objekty v kódu jsou generovány s relativní velikostí a pozicí v procentech, díky čemuž je výstupní animace responzivní. Animace se tak přizpůsobí velikosti nadřazeného prvku nebo velikosti okna. A díky podpoře CSS3 Keyframes v mobilních prohlížečích, je animace spustitelná i na mobilních zařízeních. Například prohlížeč dnes nejrozšířenějšího mobilního operačního systém Android plně podporuje CSS3 modul Keyframes od verze 4 [20, 21].

Na elementy ve výsledném kódu lze doplněním párového tagu <**script**> navázat JS kód. Skript může obsluhovat různé události provedené nad daným elementem. Jednotlivé elementy mají pro odlišení jiné názvy tříd. V aplikaci je rovněž možné jednotlivým elementům přiřadit jedinečné ID, pomocí kterého je možné element dohledat v JS kódu, například pomocí metody getElementById("id").

7 Test výkonnosti a UX testování

7.1 UX testování

Pojem UX je zkratkou pro slovní spojení **U**ser e**X**perience, znamenající **uživatelský prožitek**. Cílem uživatelského testování je zjistit, jakým problémům čelí v aplikaci její skuteční uživatelé. Uživatelské testování může být prováděno nad plně funkční finální aplikací nebo nad aplikací ve vývoji. Je dokonce možné testovat i funkční prototypy nebo grafické návrhy.

Při *testování použitelnosti* je nejdříve potřebné analyzovat **cílovou skupinu**. Zjistí se, kdo bude používat aplikaci a jaké jsou potřeby uživatelů. Dalším krokem je vytvoření **scénářů**, které popisují správný postup plnění úkolů. V příloze A - Scénáře k UX testování se nacházejí obrázky, ve kterých zelená cesta představuje scénář. Červená cesta představuje odlišnosti od ideálního scénáře, kterých se respondenti během testování dopustili.

Vytvořené úkoly se zadají respondentům a po testu je provedeno porovnání scénáře a postupu respondenta. Vzorek respondentů by měl odpovídat cílovým skupinám aplikace. Podíl zkušenějších a nezkušených uživatelů by měl být rovnoměrný, protože nezkušený uživatel může narazit na zcela jiné problémy, než zkušený uživatel. Počet respondentů je libovolný, skupina 5 – 8 osob dokáže odhalit až 90% problémů [22]. Samotné uživatelské testování by mělo probíhat jako individuální moderované sezení v UX laboratoři. Reakce a postup testujícího respondenta by měly být **snímány** a respondent by měl nahlas přemýšlet. Moderátor dává úkoly uživateli a klade doplňující otázky. Následuje analýza získaných dat, která by měla ukázat problematické prvky aplikace. Výsledná zpráva by měla nasměrovat k vymyšlení nejvhodnějšího řešení, které by vedlo ke zvýšení použitelnosti [23, 24].

Pro testování bylo vybráno 5 lidí s průměrnou až pokročilou znalostí informačních technologií (běžná kancelářská práce, znalost prostředí nástroje typu Blender). Postup testování byl monitorován zachycením toho,co se děje na monitoru. Pozorovatel u každého respondenta zapisoval poznámky o tom, jak reagoval v různých částech úkolů a jaké měl problémy. Po testu byl postup porovnáván se scénářem a proběhla diskuze o příčinách problému a o tom, jak tyto problémy odstranit. Pro potřeby testování bylo vytvořeno následujících 5 úkolů:

Vložení SVG a manipulace s rychlostí animace Uživatel měl za úkol vložit na plátno bitmapový obrázek a SVG obrázek. Vložené objekty měl následně jednoduchou animací animovat a použitím časové funkce nastavit různé rychlosti objektů. Na závěr měl prozkoumat vygenerovaný kód a výslednou animaci.

Práce s klíčovými snímky Po vložení textu je vytvořena animace, ve které se periodicky mění velikost textu. Animace obsahuje několik klíčových snímků a uživatel má následně pomocí přesunů klíčových snímků změnit chod animace.

Práce s atributy a s vlastnostmi objektu Úkolem je vložit SVG obrázek z připravené galerie a SVG z disku. Ve vytvořené animaci se má vlastnost zkosení animovat ze zadané do nulové hodnoty. Následně je otestována změna názvu vrstvy a změna ID elementu.

Práce s kontejnerem a uložení projektu Úkolem je vytvořit animaci oběhu planety s měsícem kolem Slunce. Je otestováno přepínání mezi kontejnerem a hlavním plátnem a seskupování objektů. Úkol rozšiřuje zadání předešlého úkolu a obsahuje změnu dalších dostupných vlastností. Výsledek animace má být poté uložen do souboru.

Práce s vrstvami Poslední úkol má za cíl otestovat práci s kontextovou nabídkou a manipulaci vrstvami, jako je přesun mezi kontejnery nebo mazání vrstev a klíčových snímků.Uživatel má načíst projekt ze souboru a pracovat s animací. Zjistí se tak, jak intuitivní a logická je celková práce a pohled na časovou osu.

7.1.1 Vložení SVG a manipulace s rychlostí animace

Nejvýraznější odlišnosti od ideálního scénáře popisuje obrázek 9. První pohled na aplikaci a nabídka nástrojů v podobě ikonek měla za následek záměnu nástroje pro nahrání projektu ze souboru s nástrojem pro vložení obrázku. Řešením bylo vhodně upravit popisky u odkazů a například "*Nahrát ze souboru*"bylo změněno na "*Načíst projekt ze souboru*". Důležitým zjištěním bylo, že většina uživatelů intuitivně kliká pravým tlačítkem myši na různé prvky aplikace v situacích, kdy si neví rady. V původní verzi aplikace byla kontextová nabídka implementována pouze na objekty umístěné na plátně. Pro zlepšení použitelnosti byla kontextová nabídka vytvořena i v časové ose. Nabídka poskytuje možnost smazání a vytvoření snímku nebo přejmenování vrstvy.

Dalším problémem, pozorovaným u všech uživatelů, bylo nastavení časové funkce. Možnost změny křivky se zobrazí po kliknutí na snímek, ale uživatelé prohledávali panel vlastností, protože nevěděli, co mají nastavit. K chybnému chování přispělo i to, že uživatelé postrádali všeobecné informace o tom, jak časová funkce v animaci funguje.

Vygenerování a prohlédnutí animace bylo intuitivní a provedeno bez problémů. Všichni uživatelé věděli, kde hledat výsledek jejich práce. U jednoho uživatele byl vznesen požadavek na přidání tlačítka, které zajistí i stažení výsledného kódu.

7.1.2 Práce s klíčovými snímky

Je důležité zmínit, že k některým výsledkům se lze dostat i několika alternativními způsoby. Není tedy správný jen jediný scénář. Většina uživatelů namísto vytváření snímků dvojklikem na časové ose využila druhou možnost vytvoření snímku - posunutím ukazatele na místo, které neobsahuje žádný snímek a změna některé z vlastností má za následek vytvoření nového snímku. Tento způsob měl jednu nevýhodu v tom, že se snadno přehlédlo vytvoření nového snímku, například po nechtěné změně pozice ukazatele. Proto bylo přidáno potvrzení, které se objeví v případě vytvoření snímku změnou některé z vlastností objektu.

Jeden z uživatelů nechtěně zrušil označení aktuální vrstvy a změna vlastnosti tak neměla žádný vliv. Řešením by bylo deaktivovat editační pole v panelu vlastností, v případě, kdy není označena žádná vrstva. V části úkolu, kde bylo potřebné změnit hodnotu vlastností se ukázalo, že panel nástrojů obsahuje příliš mnoho nastavení a hledat správný parametr je obtížné. Proto byly vlastnosti seskupeny a jsou zobrazovány jen po kliknutí na sekci.

Vložení a editace textu byly pro uživatele intuitivní a bezproblémové. Při přesouvání snímků bylo pro některé respondenty složité odlišit jednotlivé snímky. Rychlým řešením problému by mohlo být poskytnutí informace o tom, které vlastnosti daný snímek mění. Tato informace by musela být umístěna někde v blízkosti klíčového snímku.

Odlišnosti od ideálního scénáře jsou přehledně uvedeny na obrázku 10.

7.1.3 Práce s atributy a s vlastnostmi objektu

Tento úkol byl, až na pár drobných nesrovnalostí, splněn podle scénáře. Kde se uživatelé lišili od scénáře je uvedeno na obrázku 11. Od většiny uživatelů byl vznesen požadavek na možnost nahrání SVG ze souboru, protože kopírování XML kódu do textového pole nebylo moc intuitivní. Změna názvu vrstvy byla naopak intuitivní a bezproblémová i přes to, že není na první pohled nikde znát, že tato možnost je k dispozici.

7.1.4 Práce s kontejnerem a uložení projektu

Cílem tohoto komplexnějšího úkolu bylo vytvořit smysluplnou animaci pomocí dostupných nástrojů. Odlišnosti od ideálního scénáře jsou uvedeny na obrázku 12. Prvním problémem pro většinu uživatelů bylo vytvořit kontejner. Po kliknutí na ikonu nástroje pro vložení uživatelé čekali, co se stane. Nějakou dobu trvalo zjistit, jak se s daným nástrojem pracuje.

S přihlédnutím na výsledky testů byla přidána možnost editace zaoblení rohů pro všechny rohy hromadně. Uživatelé dál nevěděli, jak se pracuje s vlastností transform-origin, proto byl přidán tooltip s nápovědou.

O funkci zanoření do kontejneru a posun o úroveň zpět, uživatelé zpočátku nevěděli. Trvalo nějakou dobu, než objevili způsob přesunu mezi úrovněmi. Nápomocným byla drobečková navigace, která ukazuje aktuální kontejner a úroveň zanoření.

7.1.5 Práce s vrstvami

Přehled odlišností od ideálního scénáře uvádí obrázek 13. I tady někteří uživatelé nevěděli, jak se pracuje s kontejnerem. Snažili se dohledat správnou funkci v kontextovém menu. Proto byla do kontextového menu přidána možnost pro zanoření do kontejneru. Při mazání klíčového snímku jeden z uživatelů použil klávesu delete, která maže pouze vrstvy a nedopatřením tak smazal celou vrstvu. Proto přibyla možnost mazat klávesou delete i jednotlivé snímky.

Odhalit způsob řazení vrstev trval delší dobu a nakonec byla změna pořadí provedena pomocí kontextového menu. Uživatelé nevěděli, že měnit pořadí lze i táhnutím

vrstev na časové ose. Proto byla graficky odlišena plocha pro posouvání jednotlivých vrstev.

7.1.6 Závěrečné zhodnocení

Testování použitelnosti pomohlo odhalit místa, která dělala uživatelům největší problémy. Menší problémy, které bránily nebo ztěžovaly provést danou část úkolu, byly napraveny. Jedná se například o přidání kontextových menu na různé prvky aplikace, přidání chybových hlášek a popisků nebo změna grafických prvků aplikace.

Konkrétní změnou byla mírná grafická změna ukazatele, který v nové podobě vybízí uživatele ovládat pohyb táhnutím. Uživatelé měli tendenci zkoumat vytvořenou animaci pomocí ovládacích tlačítek *"Přehrát animaci"* a *"Zastavit animaci"*. Méně častá byla jednodušší a rychlejší možnost - ovládání animace kliknutím na žádaný časový úsek nebo táhnutím ukazatele.

Obecným problémem se stala práce s časovou osou. Uživatelé nevěděli, co představují jednotlivé snímky a často vznikaly nepřehledné situace. Z velké části byly tyto problémy způsobeny neznalostí prostředí aplikace a neznalostí způsobu pracování s klíčovými snímky. Jakmile byl uživatel poučen o fungování aplikace, dopadly další testy lépe. Uživatelé by jistě ocenili i uživatelský manuál. Například aplikace podobného typu Google Web Designer má vypracovanou podrobnou nápovědu doplněnou o názorné ilustrované ukázky [25]. Další zpřehlednění časové osy by se dalo dosáhnout oddělením jednotlivých animovaných vlastností objektu do jednotlivých řádků a zobrazením podvrstev ve stromové struktuře.

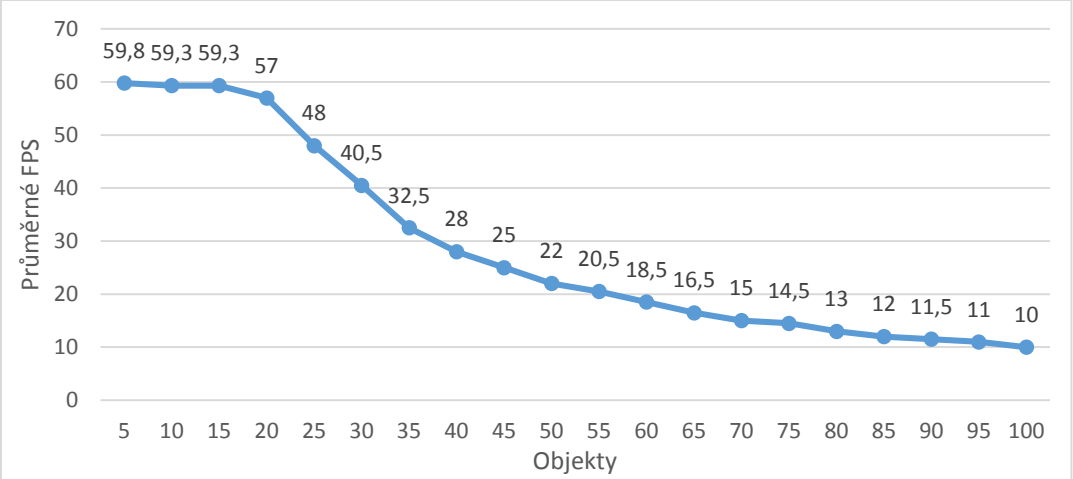
7.2 Výkonnostní testování

Výkonnostní testování bylo provedeno nad animací na plátně, která je ovládána pomocí ukazatele na časové ose, a nad výslednou animací, která je dostupná u vygenerovaného kódu v *iframe*. Cílem testování byla rychlost zobrazování, neboli **snímková frekvence**, která je udávána v počtu snímků za sekundu (FPS). Jako přijatelná hodnota pro lidské oko, kdy je pohyb ještě plynulý, se udává 25 až 30 FPS.

Testy proběhly na počítači s *Intel(R) Core(TM) i5-2410M CPU 2.30Ghz* s *4GB RAM* paměti a grafickou kartou *NVIDIA GeForce GT 520M*. Aplikace byla spuštěná v prohlížeči *Google Chrome* ve verzi 42 na operačním systému *Windows 8.1 Pro*. K zachycení FPS byl použit vestavěný nástroj pro měření FPS, který je součástí *Nástroje pro vývojáře* v prohlížeči Google Chrome.

7.2.1 Animace na plátně

Samotné testování probíhalo tak, že se postupně přidávaly objekty na plátno s předem definovanou animací, u které se měřila snímková frekvence. Na obrázku 7 je graf udávající průměrné FPS v závislosti na počtu objektů. Je důležité zmínit, že k zajištění plynulé animace o 60 FPS má každé vykreslení přibližně 16 milisekund. Za tuto dobu se musí stihnout přepočítat a následně aktualizovat všechny animované vlastnosti [26]. Jestliže



Obrázek 7: Graf s výsledky výkonnostního testování v aplikaci

tyto operace trvají déle, FPS klesá. Z grafu 7 lze vyčíst, že animace je ještě plynulá při 45 objektech. Výsledek testu potvrzuje, že jQuery se pro animaci elementů nehodí. Nicméně, aplikace je použitelná i s více objekty, protože při práci na plátně není nutné vysoké FPS.

Při prvních testech byly naměřeny dokonce ještě horší hodnoty. Optimalizací kódu, který se stará o interpolaci a aktualizaci animovaných vlastností, se dosáhlo značného zlepšení. Tabulka 1 uvádí, že FPS se u animace o počtu do 60 objektů zvětšilo i několikanásobně . Optimalizace spočívala v tom, že výpočet nových hodnot vlastnosti proběhl pouze v případě, že došlo ke změně dané vlastnosti. V dalším kroku byly vypuštěny některé výkonnostně náročnější fragmenty kódu.

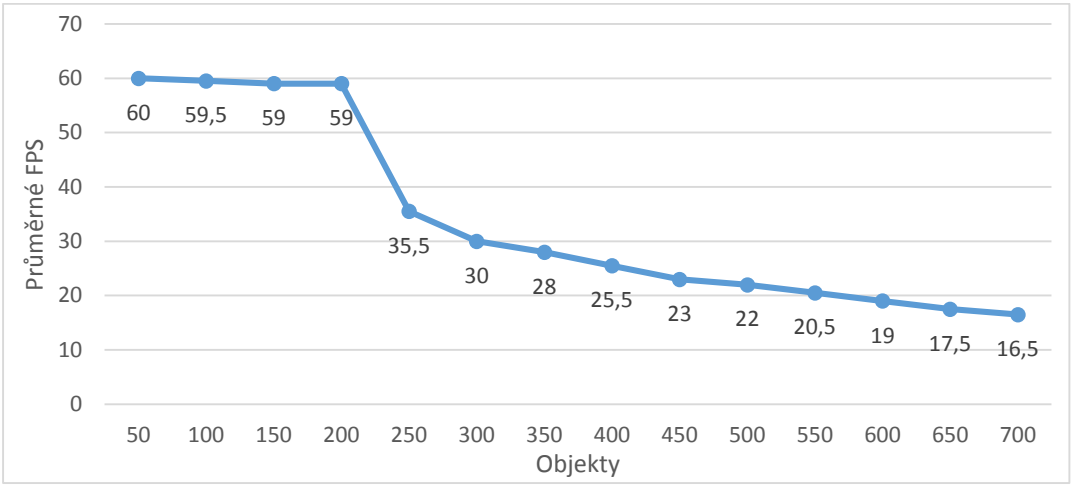
Například příkaz $('.shape-helper').css('left', '-=1'), který slouží ke korekci polohy pomocného elementu, zhoršil snímkovou frekvenci až o 10 FPS. Jako další vhodné řešení se nabízí použití sady nástrojů **GSAP (GreenStock Animation Platform)**, která je známá svým vysokým výkonem ve webových animacích v prostředí JavaScriptu [27].

7.2.2 Výsledná CSS animace

Výsledná animace je již vytvořena pomocí CSS3, a tak jsou výsledky testu oproti JS animaci výrazně lepší. Z grafu na obrázku 8 je možné vyčíst, že animace je plynulá ještě při přibližně 400 animovaných objektech. Nicméně, stažená animace v HTML souboru, čítající 400 objektů, byla při samostatném spuštění plynulá a bylo naměřeno 57 až 59 FPS.

	FPS	
počet objektů	**před optimalizací**	**po optimalizaci**
5	29,9	59,8
10	21,6	59,3
15	15,4	59,3
20	12,2	57
25	10	48
30	8,3	40,5
35	7,4	32,5
40	6,3	28
45	5,4	25
50	5	22
55	4,5	20,5
60	4,3	18,5

Tabulka 1: Porovnání FPS před a po optimalizaci



Obrázek 8: Graf s výsledky výkonnostního testování výsledné animace

8 Závěr

Výsledkem této bakalářské práce je nástroj pro tvorbu webových animací, který umožňuje lehce přidávat různé typy objektů a vlastnosti těchto objektů následně animovat. Součástí aplikace je časová osa, která přehledně zobrazuje jednotlivé klíčové snímky a umožňuje zobrazit průběh animace v čase.

V rámci této práce byly rovněž popsány HTML5 technologie a CSS3 moduly, které byly využity k implementaci nástroje. Byly rovněž představeny jiné nástroje, určené k tvorbě animací a byly rozebrány rozdíly různých technologií, určených pro tvorbu animací.

Stávající řešení určitě poskytuje prostor pro vylepšení a rozšiřování, zejména zlepšení rychlosti animace přímo v aplikaci. Do budoucna by se mohl nástroj rozšířit o několik dalších animovatelných vlastností a také by mohl být vylepšen způsob manipulace s SVG objekty, jako například editace a vytváření vlastních SVG přímo v aplikaci. Velmi užitečnou vlastností by byl import již vytvořených animací z HTML a CSS kódu.

Podařilo se vytvořit online nástroj, který umožňuje jednoduchým způsobem animovat uživatelem vložené objekty bez hlubších znalostí CSS3 animace. Tato práce se snažila ukázat, že možnosti technologií popsaných v této práci, jsou dnes natolik široké, že je lze bez problémů použít k vytvoření komplexních animací i bez použití jiných nástrojů, jako například Java. Současné trendy a masivní využití HTML5 a CSS3 v praxi to dokazují.

9 Reference

[1] Co je to Google Web Designer?. In: *Google* [online]. 2015 [cit. 2015-04-05]. Dostupné z: https://support.google.com/webdesigner/answer/3184833?hl=cs

[2] *Stylie* [online]. [cit. 2015-04-16]. Dostupné z: `http://jeremyckahn.github.io/stylie/`

[3] *CSS Keyframes Animation Generator* [online]. [cit. 2015-04-16]. Dostupné z: `http://cssanimate.com/`

[4] *BannerFlow* [online]. [cit. 2015-04-16]. Dostupné z: `http://www.bannerflow.com/`

[5] HTML5. *World Wide Web Consortium (W3C)* [online]. 2014 [cit. 2015-04-07]. Dostupné z: `http://www.w3.org/TR/html5/`

[6] Support tables for HTML5, CSS3, etc. *Can I use* [online]. [cit. 2015-04-07]. Dostupné z: `http://caniuse.com/#cats=HTML5`

[7] Browser Statistics. *W3Schools Online Web Tutorials* [online]. [cit. 2015-04-07]. Dostupné z: `http://www.w3schools.com/browsers/browsers_stats.asp`

[8] GOLDSTEIN, Alexis, Louis LAZARIS a Estelle WEYL. *HTML5 a CSS3 pro webové designéry*. Brno: Zoner Press, 2011. ISBN 978-80-7413-166-0.

[9] BALAN, Deepu. RGBa vs Opacity: The Difference Explained. In: *Deepu Balan – Scribblings of a cyber geek – Web Design Blog* [online]. 2010 [cit. 2015-04-16]. Dostupné z: `http://deepubalan.com/blog/2010/03/29/rgba-vs-opacity-the-difference-explained/`

[10] STOREY, Dudley. *Pro CSS3 Animation*. New York: Apress Media, 2012. ISBN 978-1-4302-4722-7.

[11] BRADLEY, Steven. *CSS animations and transitions for the modern Web*. USA: Adobe Press, 2014. ISBN 01-339-8050-2.

[12] GOLDSTEIN, Alexis. *Learning CSS3 animations and transitions: A hands-on guide to animating in CSS3 with transforms, transitions, keyframe animations, and JavaScript*. Upper Saddle River, NJ: Addison-Wesley, 2012. ISBN 03-218-3960-9.

[13] Why are CSS3 transitions worth using?. In: BRADSHAW, Rich. *CSS Transitions, Transforms and Animation Tutorial* [online]. 2013 [cit. 2015-04-09]. Dostupné z: `http://css3.bradshawenterprises.com/blog/jquery-vs-css3-transitions/`

[14] RAO, Siddharth. CSS3 vs jQuery Animations. *Dev.Opera* [online]. 2012 [cit. 2015-04-28]. Dostupné z: https://dev.opera.com/articles/css3-vs-jquery-animations/

[15] SVG (basic support). *Can I use* [online]. [cit. 2015-04-12]. Dostupné z: http://caniuse.com/#search=svg

[16] G - SVG. *Mozilla Developer Network* [online]. 2015 [cit. 2015-04-12]. Dostupné z: https://developer.mozilla.org/cs/docs/Web/SVG/Element/g

[17] MARSH, Charles. Goodbye, Layout Invalidation: Animating SVGs With CSS Transforms. In: *Charlie Marsh* [online]. 2014 [cit. 2015-04-12]. Dostupné z: http://www.crmarsh.com/svg-performance/

[18] SVG Animation. JENKOV, Jakob. *Tutorials.jenkov.com* [online]. [cit. 2015-04-12]. Dostupné z: http://tutorials.jenkov.com/svg/svg-animation.html

[19] NISKANEN, Sampo. CoffeeScript vs. TypeScript vs. Dart. *Code for Hire* [online]. 2013 [cit. 2015-04-13]. Dostupné z: http://codeforhire.com/2013/06/18/coffeescript-vs-typescript-vs-dart/

[20] Mobile Devices Statistics. *W3Schools* [online]. © 1999-2015 [cit. 2015-04-21]. Dostupné z: http://www.w3schools.com/browsers/browsers_mobile.asp

[21] CSS3 Animation. *Can I use* [online]. 2015 [cit. 2015-04-21]. Dostupné z: http://caniuse.com/#search=css-animation

[22] PLOTĚNÝ, Luboš. Uživatelské testování webu NAVRCHOLU.cz. In: *Slideshare* [online]. 2009 [cit. 2015-04-18]. Dostupné z: http://www.slideshare.net/dobryweb/uivatelsk-testovn-webu-navrcholucz-1828805

[23] RUBIN, Jeffrey. *Handbook of usability testing: How to plan, design, and conduct effective tests*. 2nd ed. Indianapolis: Wiley, 2008. ISBN 978-0-470-18548-3.

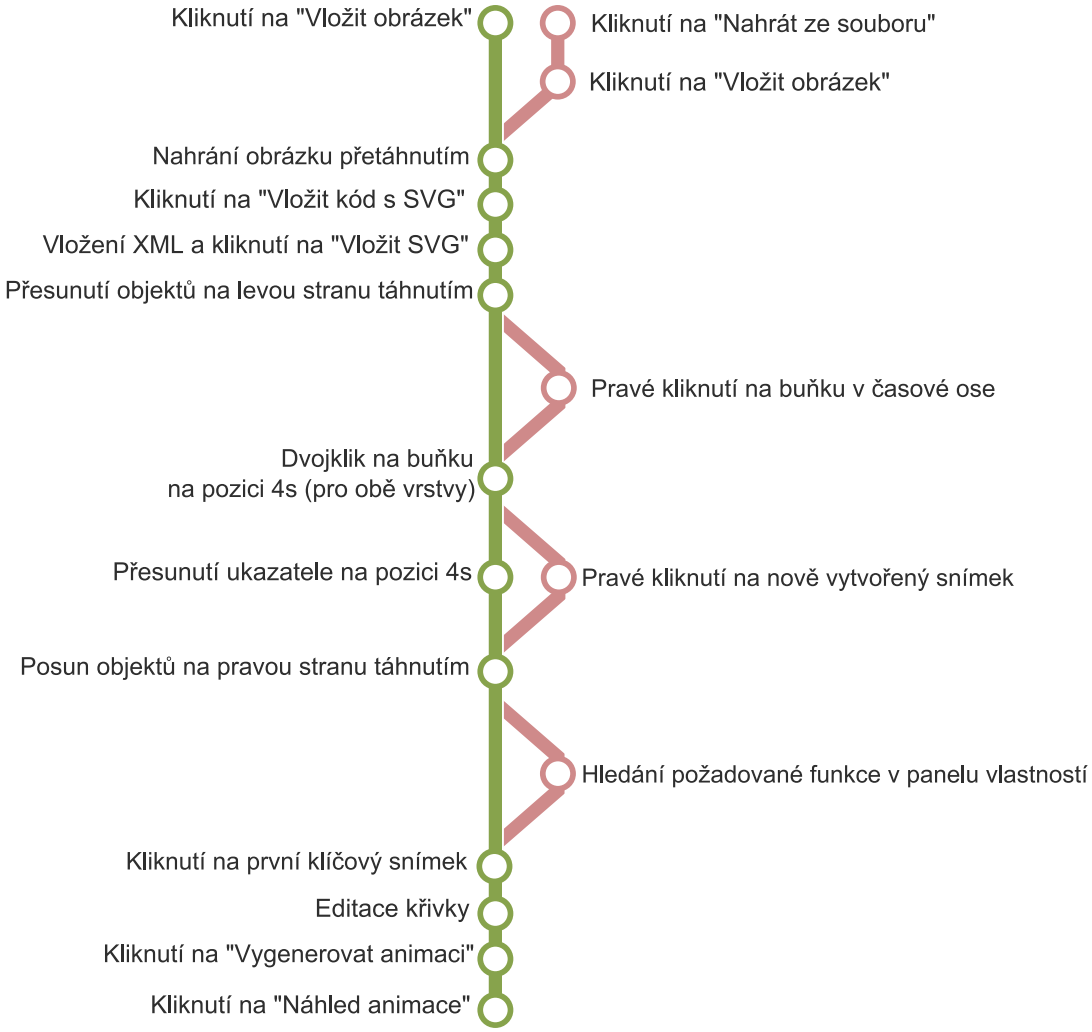
[24] Uživatelské testování použitelnosti. [online]. [cit. 2015-04-18]. Dostupné z: http://www.dobryweb.cz/uzivatelske-testovani

[25] Nápověda Google Web Designer. In: *Nápověda Google Web Designer* [online]. © 2015 [cit. 2015-04-18]. Dostupné z: https://support.google.com/webdesigner

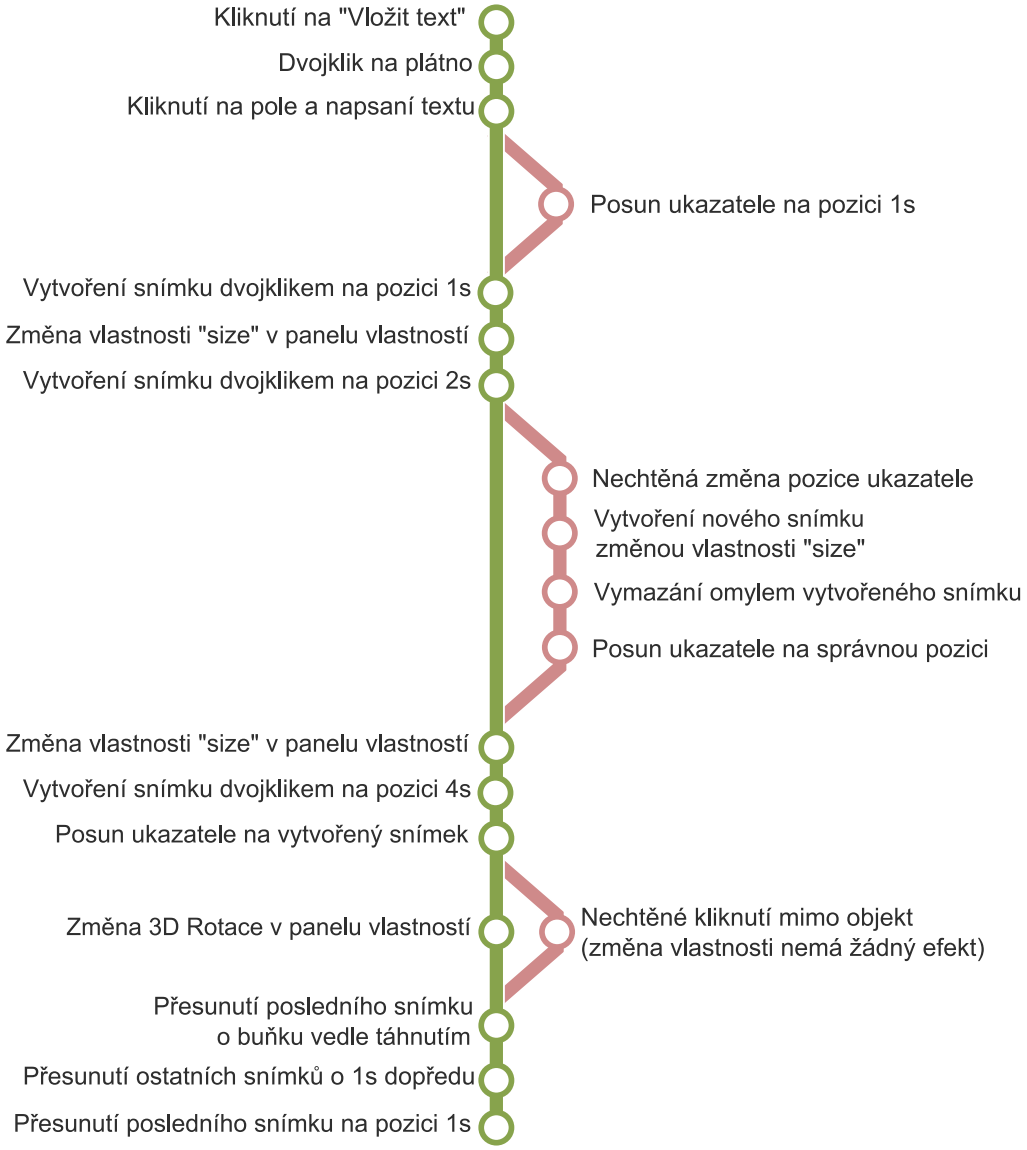
[26] Optimising for 60fps everywhere: Performance and speed in JavaScript and CSS. In: *GoSquared Engineering* [online]. 2014 [cit. 2015-04-25]. Dostupné z: https://engineering.gosquared.com/optimising-60fps-everywhere-in-javascript

[27] *GreenSock | GSAP* [online]. © 2015 [cit. 2015-04-25]. Dostupné z: http://greensock.com/gsap

A Scénáře k UX testování

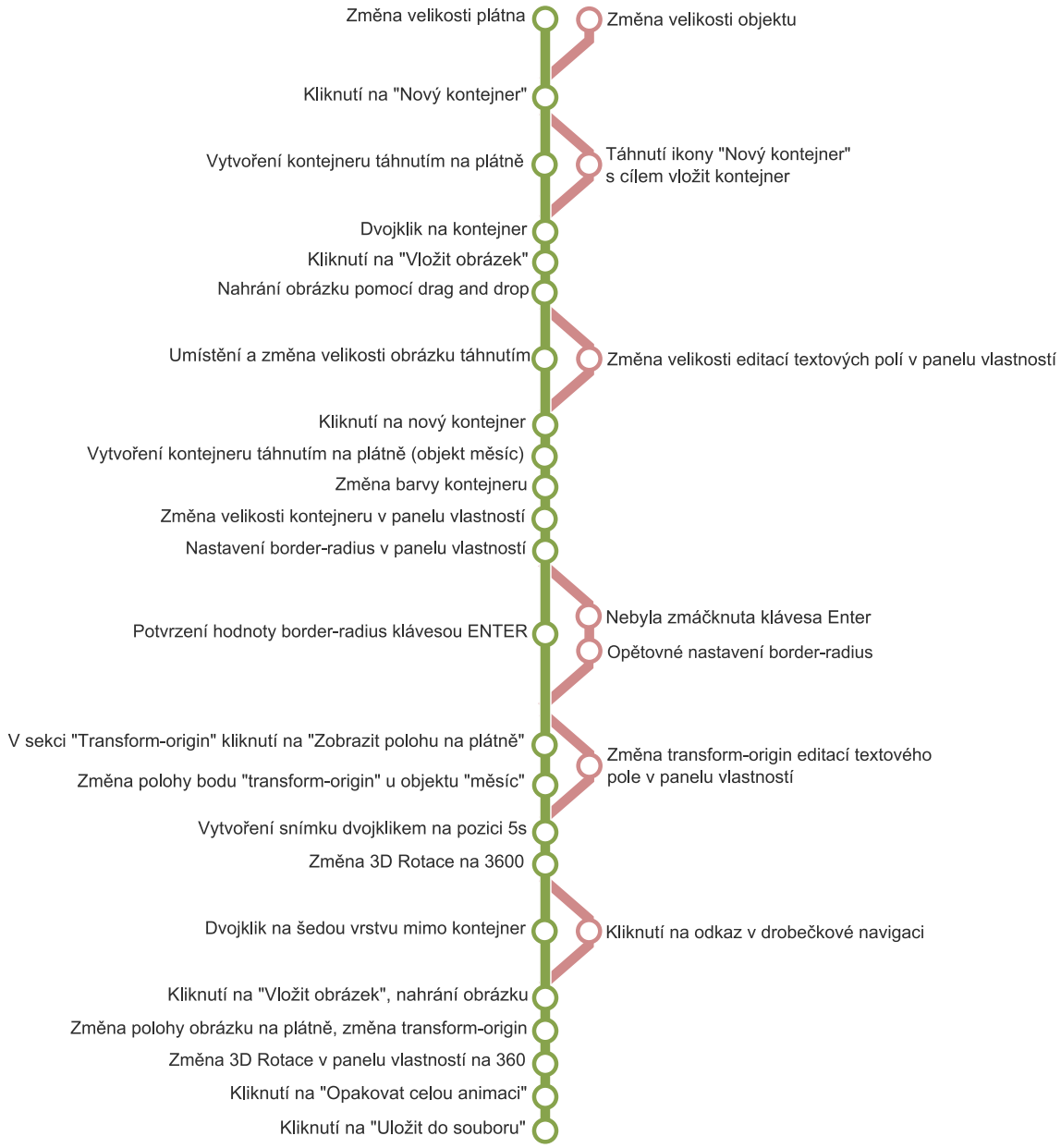


Obrázek 9: Scénář 1 - Vložení SVG a manipulace s rychlostí animace

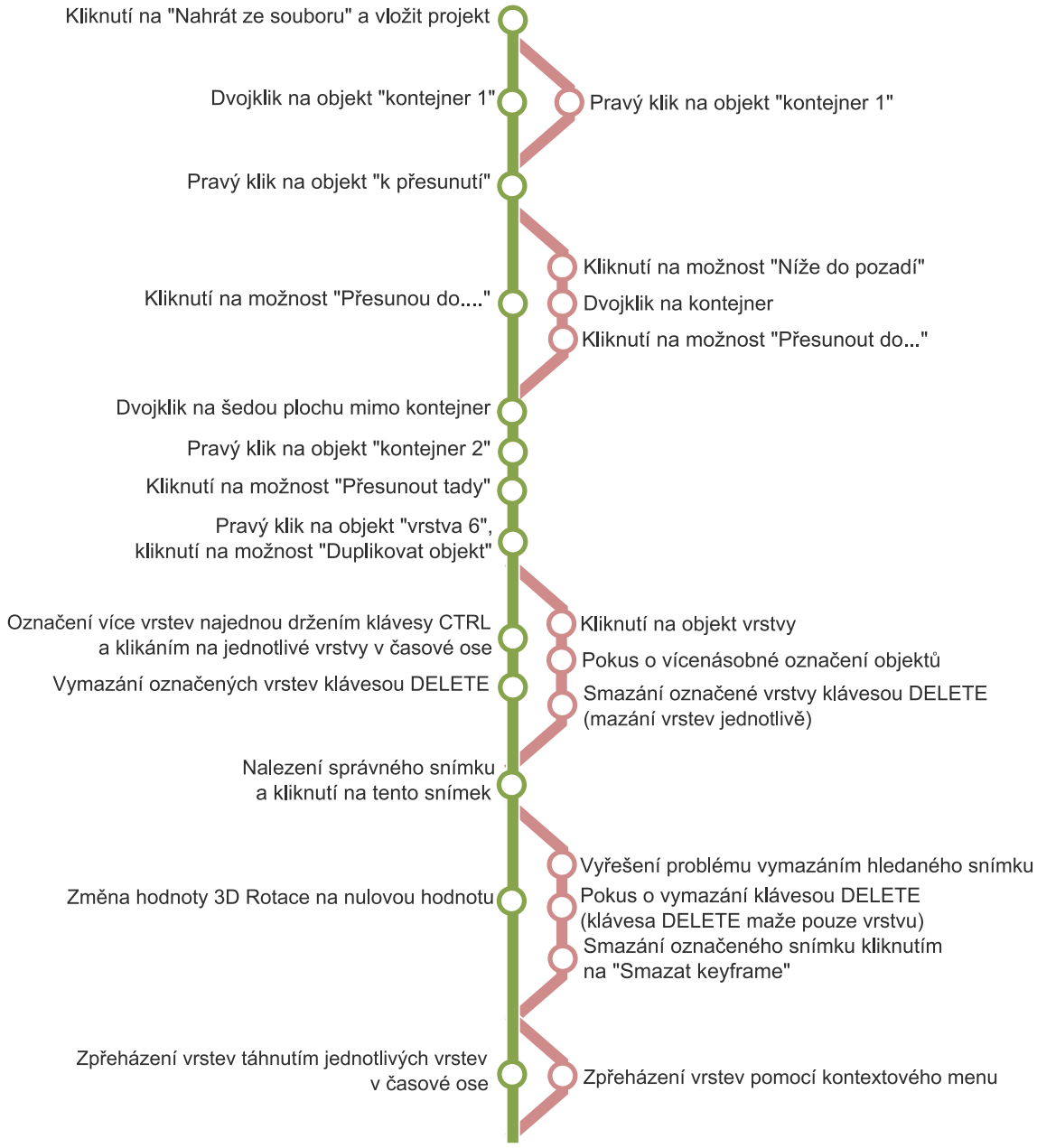


Obrázek 10: Scénář 2 - Práce s klíčovými snímky

Obrázek 11: Scénář 3 - Práce s atributy a s vlastnostmi objektu



Obrázek 12: Scénář 4 - Práce s kontejnerem a uložení projektu



Obrázek 13: Scénář 5 - Práce s vrstvami

B Obsah CD

- **bp.pdf** - text této práce
- **sources** - zdrojové kódy aplikace
- **app** - spustitelná aplikace
- **examples** - Příklady animací a projektů, vytvořených pomocí nástroje