VYSOKÁ ŠKOLA POLYTECHNICKÁ JIHLAVA

Katedra technických studií

Knihovna animací pro ovládací prvky grafické knihovny SWT

Autor práce: Robert Havránek

Vedoucí práce: Ing. Petr Stružka, Ph.D.

Jihlava 2021

Vysoká škola polytechnická Jihlava

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Autor práce:	**Robert Havránek**
Studijní program:	Elektrotechnika a informatika
Obor:	Aplikovaná informatika
Název práce:	**Knihovna animací pro ovládací prvky grafické knihovny SWT.**
Cíl práce:	Cílem bakalářské práce je prozkoumat návrhy animací grafických ovládacích prvků napříč operačními systémy a grafickými knihovnami, popsat existující řešení pro knihovnu SWT a navrhnout vlastní řešení animací pro knihovnu SWT. Implementovat systém, který bude schopný animovat existující či nově vytvořené ovládací prvky. Animace se budou týkat otevření okna, rolování v tabulce, přesun ovládacího nebo jiného prvku, zobrazení obrázku, změna velikosti prvku a změna barvy. Implementováno bude více druhů animací například: pro přesun prvku - lineární či postupně akcelerující rychlost a pro zvětšení prvku - lineární či odrážející efekt. Dále vytvořit vzorovou aplikaci, na které se demonstruje funkčnost implementovaného řešení a prozkoumat, zda je možné vytvořit mechanismus, který bude zjišťovat jaké ovládací prvky jsou umístěny v okně a vkládal do nich animace bez nutnosti zásahu do původní aplikace.

Ing. Petr Stružka, Ph.D.
vedoucí bakalářské práce

doc. Ing. Zdeněk Horák, Ph.D.
vedoucí katedry
Katedra technických studií

Abstrakt

Práce se zabývá průzkumem různých druhů animací ovládacích prvků napříč operačními systémy a grafickými knihovnami. Dále prozkoumává existující řešení animací pro knihovnu SWT. Součástí práce je návrh a implementace vlastní knihovny animací, která rozšíří knihovnu SWT. Knihovna bude schopná animovat existující či nově vytvořené ovládací prvky. Funkčnost bude demonstrována na vzorové aplikaci. Práce bude dále prozkoumávat, zda je možné vytvořit mechanismus, který bude animovat ovládací prvky v aplikaci, bez nutnosti zásahu do kódu aplikace.

Klíčová slova

Animace; ovládací prvek; knihovna; SWT; Java

Abstract

The work deals with the research of various types of animation controls across operating systems and graphics libraries. It also explores existing animation solutions for the SWT library. Part of the work is the design and implementation of its own library of animations, which will expand the SWT library. The library will be able to animate existing or newly created widgets. The functionality will be demonstrated on a sample application. The work will further explore whether it is possible to create a mechanism that will animate the controls in the application, without the need to intervene in the application code.

Key words

Animation; widget; library; SWT; Java

Prohlašuji, že předložená bakalářská práce je původní a zpracoval/a jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem v práci neporušil/a autorská práva
(ve smyslu zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů, v platném znění, dále též „**AZ**“).

Souhlasím s umístěním bakalářské práce v knihovně VŠPJ a s jejím užitím k výuce nebo k vlastní vnitřní potřebě VŠPJ.

Byl/a jsem seznámen/a s tím, že na mou bakalářskou práci se plně vztahuje **AZ**, zejména § 60 (školní dílo).

Beru na vědomí, že VŠPJ má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom/a toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem VŠPJ, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených vysokou školou na vytvoření díla (až do jejich skutečné výše), z výdělku dosaženého v souvislosti s, užitím díla či poskytnutím licence.

V Jihlavě dne 26. dubna 2021

…….
Podpis studenta

Poděkování

Chtěl bych poděkovat vedoucímu bakalářské práce Ing. Petrovi Stružkovi, Ph.D. a Ing. Juraji Ondruškovi za vedení, ochotu a cenné rady, které mi v průběhu zpracování bakalářské práce dali. Dále bych rád poděkovat své rodině a přátelům za podporu po celou dobu studia.

Obsah

Úvod 10
1 Úvod do problematiky 12
1.1 Podporované platformy 12
1.2 Ovládací prvky knihovny SWT 13
2 Analýza stávajících řešení 14
2.1 Animace v grafických knihovnách 15
2.1.1 Trident 15
2.1.2 AnimateFX 17
2.1.3 VisualEffects 18
2.2 Animace v operačních systémech 19
2.2.1 Windows 10 19
2.2.2 Ubuntu 21
2.2.3 Android 22
2.3 Popis animací 24
2.3.1 Akce animací 24
2.3.2 Efekty animací 25
2.4 Knihovny 28
2.5 Časovače 29
3 Navrhované řešení 30
3.1 Návrh tříd 30
3.2 Návrh aplikačního rozhraní 32
3.3 Použití knihovny Trident 33
3.4 Propojení knihoven 34
3.5 Návrh ukázkové aplikace 34
3.6 Přidání animací bez zásahu do zdrojové aplikace 36
3.7 3-bodová BSD Licence 36
4 Implementace 37
4.1 Implementace systému animací 37
4.1.1 Abstraktní rodičovská třída akce 37
4.1.2 Podtřídy akce 38
4.1.3 Třída pro výběr animací 41
4.1.4 Rozhraní pro zpětná volání 41
4.2 Implementace efektů animací 42
4.2.1 Zrychlující se efekt 42

4.2.2 Odrážející efekt 42
4.2.3 Efekt třepání 43
4.2.4 Pulzující efekt 43
4.2.5 Zřetězený efekt 43
4.3 Příklady použití 44
4.4 Dokumentace pomocí JavaDoc 45
Závěr 47
Seznam použité literatury 48
Přílohy 49

Seznam obrázků

Obrázek 1: Ukázka SWT v různých OS (Eclipse) 12
Obrázek 2: Ukázka aplikací, vytvořených pomocí Radiance (Grouchnikov) 15
Obrázek 3: Zapnutí animací ve starém nastavení Windows 10 (vlastní) 20
Obrázek 4: Zapnutí animací v moderním nastavení Windows 10 (vlastní) 20
Obrázek 5: Ubuntu 20.10 (vlastní) 21
Obrázek 6: Ukázka OS Android s nadstavbou MIUI 12 (vlastní) 22
Obrázek 7: Hierarchie knihoven v Javě (Vertex Academy) 28
Obrázek 8: Zjednodušený diagram tříd (vlastní) 31
Obrázek 9: Struktura knihovny Trident (vlastní) 33
Obrázek 10: Hierarchie tlačítka (vlastní) 34
Obrázek 11: Ukázková aplikace (vlastní) 35
Obrázek 12: Položky a metody ChangerBase (vlastní) 37

Seznam tabulek

Tabulka 1: Seznam nejdůležitějších ovládacích prvků v SWT (Eclipse) 13
Tabulka 2: Třídy akcí animací (vlastní) 38

Seznam grafů

Graf 1: Průběh lineární animace (vlastní) 25
Graf 2: Průběh odrážející animace (vlastní) 25
Graf 3: Průběh postupně zrychlující animace (vlastní) 26
Graf 4: Průběh pulsující animace (vlastní) 26
Graf 5: Průběh animace natahovací gumy (vlastní) 27
Graf 6: Průběh třepající se animace (vlastní) 27

Seznam zkratek

API	Application Programming Interface
AWT	Abstract Window Toolkit
BSD	Berkeley Software Distribution
FAQ	Často kladené otázky
GNOME	GNU Network Object Model Environment
GNU	GNU's Not Unix
GUI	Grafické uživatelské rozhraní
IBM	International Business Machines Corporation
IDE	Vývojové prostředí
JDK	Java Development Kit
JRE	Java Runtime Environment
OOP	Objektově orientované programování
OS	Operační systém
RGB	Red-green-blue
SVG	Scalable Vector Graphics
SWT	Standard Widget Toolkit
WYSIWYG	What you see is what you get

Úvod

Animace se staly běžnou součástí v oblasti softwaru ve velkém množství webových a mobilních aplikací a operačních systémů. Desktopové aplikace si zatím tento typ vylepšení uživatelského prostředí moc neosvojily. Většina grafických knihoven, jako SWT, takovéto možnosti animací ovládacích prvků nenabízí.

Momentálně existuje jen velmi málo rozšíření podpory animací pro knihovnu SWT. Tyto rozšíření jsou tvořeny samotnými uživateli a obvykle se týkají pouze určitého ovládacího prvku. Obvykle jsou tyto rozšíření dost uzavřené a je problém je využít, dohledat zdrojový kód nebo najít nějakou dokumentaci.

V teoretické části této práce se zaměřím na knihovnu SWT a její ovládací prvky. Porovnám jednotlivá řešení animací v rámci knihovny SWT, ostatních knihoven a operačních systémů. Dále zde popíšu různé druhy animací a jejich využití.

V praktické části se zaměřím na návrh a detaily implementace knihovny, která bude schopna animovat existující či nově vytvořené ovládací prvky. Popíšu fungování jednotlivých animací a zaměřím se i na problematiku ohledně licencí a dokumentace. Dále popíšu postup při vytváření ukázkové aplikace.

Motivace

Vždy mě bavila tvorba grafického uživatelského rozhraní a mám předchozí zkušenosti s prací s grafickými knihovnami jako Java Swing a Windows Forms (.NET framework), ale dříve jsem se na tvorbě nebo rozšíření takovéto knihovny nepodílel. Proto mě zaujalo toto téma práce od firmy NXP, týkající se rozšíření grafické knihovny SWT o animace.

Vedle rozšíření svých znalostí o další užitečné grafické knihovny, tvorbu knihoven v Javě a fungování animací, jsem zároveň chtěl vytvořit funkční řešení, které se bude využívat přímo v praxi a usnadní práci mnoha lidem.

Cíl práce

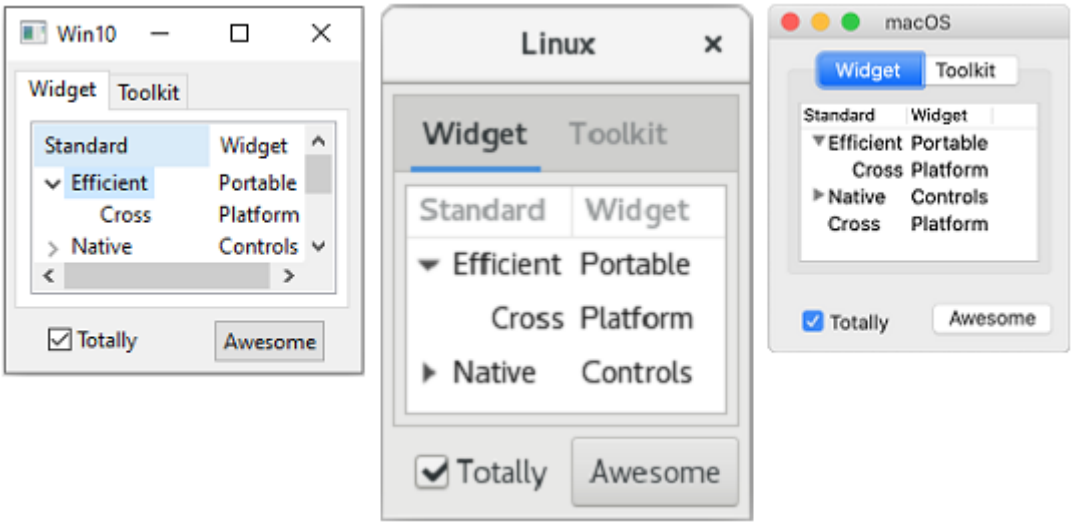
Cílem práce je prozkoumat návrhy animací grafických ovládacích prvků napříč operačními systémy a grafickými knihovnami, popsat existující řešení pro knihovnu SWT a navrhnout vlastní řešení animací pro knihovnu SWT.

Implementovat systém, který bude schopný animovat existující či nově vytvořené ovládací prvky. Animace se budou týkat otevření okna, rolování v tabulce, přesun ovládacího nebo jiného prvku, zobrazení obrázku, změna velikosti prvku a změna barvy. Implementováno bude více druhů animací například: pro přesun prvku – lineární či postupně akcelerující rychlost a pro zvětšení prvku – lineární či odrážející efekt.

Dále vytvořit vzorovou aplikaci, na které se demonstruje funkčnost implementovaného řešení a prozkoumat, zda je možné vytvořit mechanismus, který bude zjišťovat jaké ovládací prvky jsou umístěny v okně a vkládal do nich animace bez nutnosti zásahu do původní aplikace.

1 Úvod do problematiky

SWT (Standard Widget Toolkit) je open source knihovna grafických ovládacích prvků pro programovací jazyk Java. Původně byla vyvinuta firmou IBM, ale nyní se o ní stará nadace Eclipse Foundation. Tato knihovna je alternativou k AWT a Swing, ale na rozdíl od nich používá nativní systémové knihovny, takže její ovládací prvky mají stejný vzhled jako ostatní prvky operačního systému. Knihova SWT ve svém základu nepodporuje animaci svých grafických ovládacích prvků. (Eclipse)



Obrázek 1: Ukázka SWT v různých OS (Eclipse)

1.1 Podporované platformy

Jelikož SWT využívá nativní systémové knihovny, tak musí být portován pro každou platformu zvlášť (na rozdíl od AWT nebo Swing). Tím že používá různé nativní knihovny na různých operačních systémech, tak se také mohou vyskytnout specifické chyby pro nějakou platformu. (Eclipse FAQ, 2009)

- Win32
- Linux Motif and GTK
- AIX Motif
- HPUX Motif
- MacOS Carbon and Cocoa
- Photon
- Pocket PC
- Cocoa

1.2 Ovládací prvky knihovny SWT

V následující tabulce je seznam nejdůležitějších ovládacích prvků SWT (widgets).

Název	Třída	Popis
Browser	Browser	Prohlížení HTML dokumentů
Arrow	Button	Tlačítko se šipkou
Check	Button	Zaškrtávací pole (výběr M:N)
Radio	Button	Přepínač (výběr 1:N)
Push	Button	Klasické tlačítko
Toggle	Button	Výběr objektů
Text	Single	Jednořádkové pole pro zadávání textu
Text	Multi	Víceřádkové pole pro zadávání textu
Canvas	Canvas	Povrch pro kreslení grafů
Combo	Combo	Výběr z otevíracího seznamu
Composite	Composite	Může zabalovat více ovládacích prvků
DateTime	DateTime	Rozbalovací kalendář
Group	Group	Rámeček s titulkem
Label	Label	Zobrazuje text nebo obrázek
Link	Link	Hypertextový odkaz
List	List	Výběr ze seznamu
Menu	Menu	Kontextové menu
ProgressBar	ProgressBar	Znázorňuje progres vykonané operace
Shell	Shell	Okno SWT aplikace
Slider	Slider	Posuvník okna
Scale	Scale	Výběr hodnoty pomocí posuvníku
Spinner	Spinner	Výběr hodnoty pomocí šipek
Toolbar	Toolbar	Panel nástrojů

Tabulka 1: Seznam nejdůležitějších ovládacích prvků v SWT (Eclipse)

2 Analýza stávajících řešení

Desktopové aplikace obsahují pouze malé množství animací na rozdíl od mobilních nebo webových aplikací. Hlavní důvod se nachází v nedostatečné podpoře animací v grafických knihovnách ovládacích prvků nebo v samotném operačním systému. Další důvod může být ten, že v mobilních aplikacích si animace najdou lepší využití, kvůli dotykovému ovládání a poskytování zpětné vazby na něj.

Jestliže budeme chtít využít animace v nejpoužívanějších knihovnách ovládacích prvků pro desktopové aplikace (JavaFX, Swing, SWT, Windows Form, …), budeme muset využít nějakou neoficiální knihovnu nebo utilitu, která je o tuto možnost rozšíří. Většina těchto knihoven je svými autory vydávána pod Open-source licencí a lze k nim dohledat zdrojové kódy, bezplatně je používat a případně i modifikovat kód. Další možností, jak přidat animace k těmto knihovnám je implementace vlastního rozšíření, který přidá podporu pro tuto funkcionalitu.

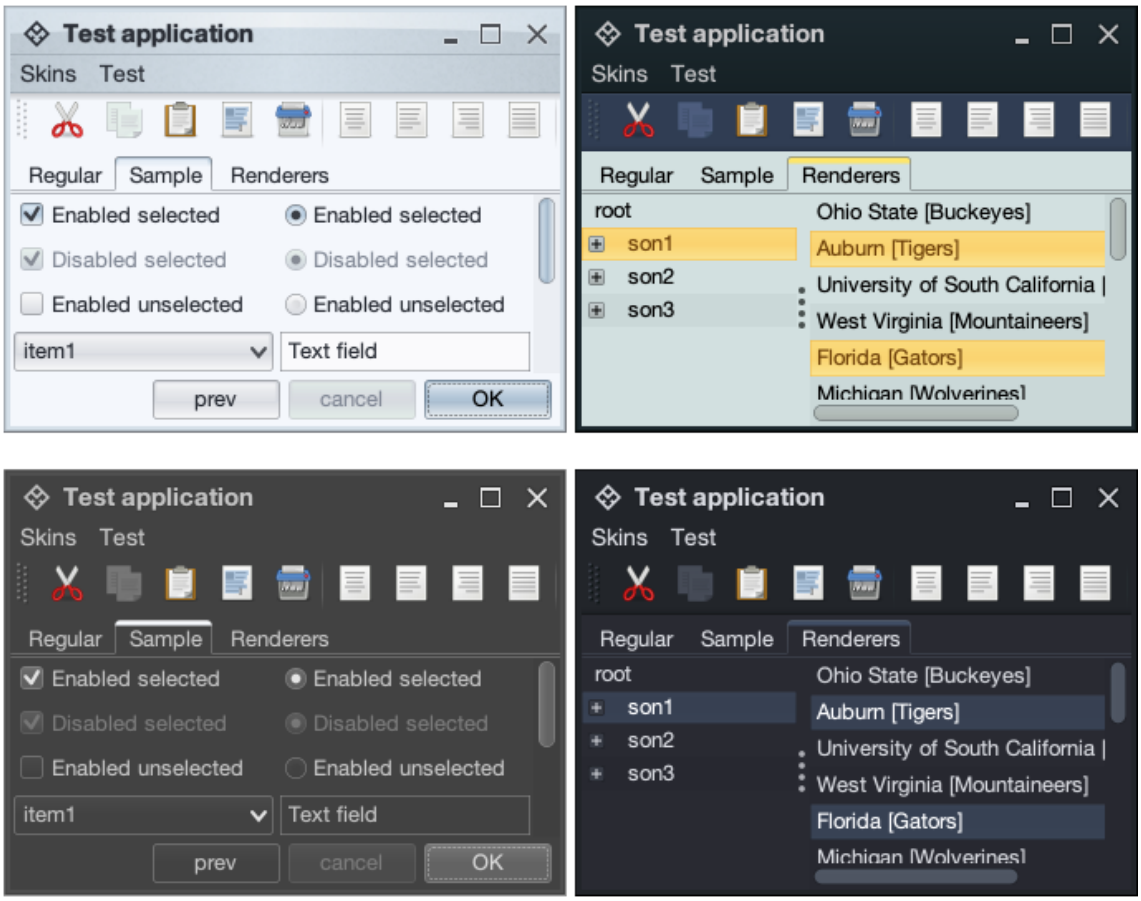
2.1 Animace v grafických knihovnách

Ve většině grafických knihoven ovládacích prvků nejsou v základu animace podporovány, ale dají se najít různé knihovny, které je o tuto funkcionalitu rozšíří.

2.1.1 Trident

Animační knihovna pro Javu od Kirilla Grouchnikova, která se od roku 2018 stala součástí projektu Radiance, na které pracuje stejný autor. Dříve podporovala i SWT, ale když se knihovna spojila s projektem Radiance, tak autor plnou podporu SWT odebral a dá se využívat jen s omezenou funkčností.

Radiance je kolekce knihoven pro psaní moderních, elegantních a rychlých Swing aplikací. Je vytvořena v Javě 9 a funguje v Javě 9 nebo novějších verzích. (Grouchnikov, 2018)



Obrázek 2: Ukázka aplikací, vytvořených pomocí Radiance (Grouchnikov)

Radiance se skládá z těchto knihoven:

Neon

Neon poskytuje API pro práci s obrázky a texty které se mění s rozlišením displeje. Škálování Neon ikon je udržuje ostré a vizuálně dokonalé pro ikony aplikací. API písem lze použít k vykreslení konzistentního textového obsahu napříč celou řadou podporovaných platforem. (Grouchnikov, 2018)

Photon

Photon umožňuje použití vektorových ikon v aplikacích Swing. Obsah SVG lze načíst za běhu asynchronně z různých místních a vzdálených zdrojů nebo překódovat offline do tříd Java / Kotlin, které používají operace plátna Java2D, které nevyžadují nákladné runtime režie závislostí třetích stran. (Grouchnikov, 2018)

Trident

Trident poskytuje výkonné a flexibilní animační API, které se pohybují od jednoduchých případů s jednou vlastností až po složité scénáře, které zahrnují více animací. Trident ovládá všechny animace v knihovnách Radiance. (Grouchnikov, 2018)

Substance

Substance poskytuje komplexní sadu API pro měnění vzhledu aplikací Swing, které splňují ty nejnáročnější požadavky na moderní design. Dodává se s vestavěnou podporou pro všechny základní komponenty Swing a flexibilním API pro vykreslování komponent třetích stran / aplikací. (Grouchnikov, 2018)

Flamingo

Flamingo poskytuje robustní sadu dalších komponent Swing, které lze použít jako stavební kameny pro vytváření moderních a bohatých aplikací Swing. Kromě poskytování mocného příkazového tlačítka a komponent pruhu navigační tabulky, Flamingo zabalí kontejner Office Command Bar (páska), který může hostit složitou hierarchii flexibilní a škálovatelné ovládací plochy aplikace. (Grouchnikov, 2018)

2.1.2 AnimateFX

Animační knihovna od uživatele vystupující na githubu pod přezdívkou Typhon0. Rozšiřuje knihovnu JavaFX, která je, jak už z názvu vypovídá, pro jazyk Java. Tato knihovna obsahuje přes 70 animací, které může programátor využít k vylepšení svého GUI. (Typhon0, 2017)

Druhy animací

- Odrážení
- Mizení
- Blikání
- Otáčení
- Rozsvícení
- Pulsování
- Překlápění
- Třepání
- Sunutí
- Přibližování
- Oddalování
- Další speciální animace

Způsob použití

Používání této knihovny je jednoduché. Stačí si vytvořit instanci ovládacího prvku z knihovny JavaFX a poté instanci efektu, na která se zavolá metoda `play`.

```
Text text = new Text("AnimateFX");
new Bounce(text).play();
```

2.1.3 VisualEffects

Knihovna VisualEffects umožňuje animace pro ovládací prvky Windows Form. Zajišťuje mnoho efektů pro změnu velikosti, místa, barvy a průhlednosti ovládacích prvků. Pomocí implementace rozhraní IEffect můžete snadno vytvářet nové druhy efektů. Dále knihovna zajišťuje kombinaci efektů např. změna barvy a přesun prvku zároveň. (Sampietro, 2014)

Druhy animací

- Odrážení
- Mizení
- Změna barvy
- Otáčení
- Překlápění
- Sunutí
- Přibližování
- Oddalování

Způsob použití

Knihovna se používá zavoláním metody `Animate` na ovládacím prvku. Této metodě musíme předat 5 povinných argumentů: akce, efekt hodnota, které má dosáhnout, doba animace v milisekundách a zpoždění startu animace v milisekundách.

Dále jí můžeme předat dva nepovinné argumenty: opačná animace a počet cyklů

```
test.Animate(
                new XLocationEffect(),
                EasingFunctions.BounceEaseOut,
                500,
                2000,
                0
            );
```

Přesunutí ovládacího prvku test po ose X o 500 px doprava, s využitím odrážejícího efektu a dobou trvání 2 s.

2.2 Animace v operačních systémech

Animace se rozšířily i v moderních operačních systémech, ať už se jedná o mobilní nebo desktopové operační systémy. V desktopových operačních systémech se většinou animace používají jen v malém množství a občas je složité je vůbec postřehnout. Hlavní úloha těchto animací je navodit dojem větší plynulosti při ovládaní operačního systému. V mobilních zařízeních zase můžeme pozorovat animací více, kvůli zlepšení komfortu dotykového ovládání.

2.2.1 Windows 10

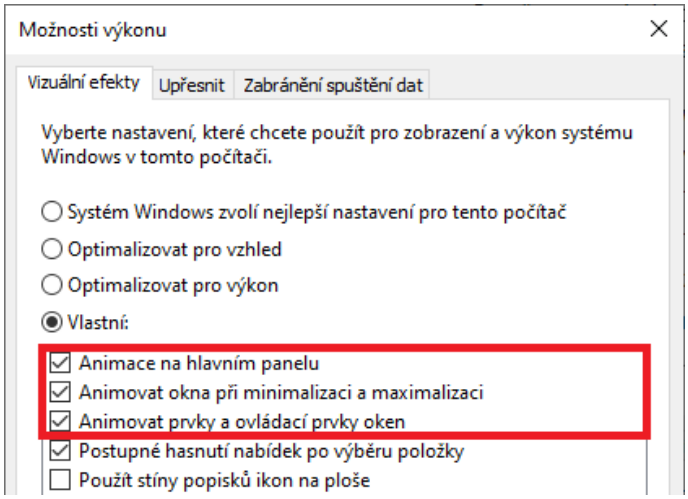
Operační systém Windows 10 obsahuje pár animací, které jsou hlavně vidět při vyvolávání nabídek z hlavní lišty. Většinou se jedná o animace s lineárním efektem, ale například při změně měsíce v kalendáři můžeme pozorovat i efekt, který se postupně zpomaluje. Tyto animace můžeme pozorovat na následujících prvcích hlavního panelu.

- Nabídka start
- Kontextové menu
- Panel oznámení
- Kalendář
- Nastavení sítě, zvuku a klávesnice

Dále Windows 10 umožňuje animovat okna při otevření a zavření nebo při maximalizaci a minimalizaci. Zde používá pro změnu velikosti a opacity rovněž lineární efekt animace. Některé animace používá tento operační systém i v jeho moderních aplikacích a nastavení. Tyto animace jsou nejvíce patrné na posuvnících okna a přepínačích nastavení.

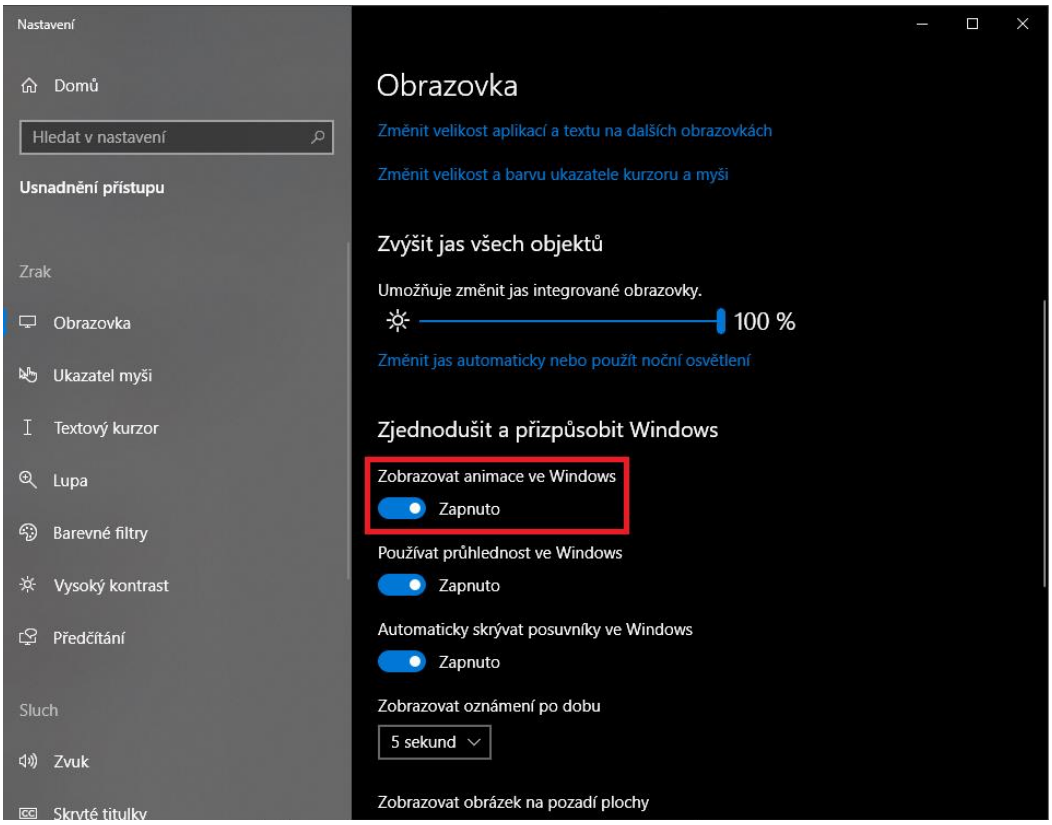
Ve Windows 10 je možně zapnout animace následujícím způsobem:

Staré nastavení: Ovládací panely -> Systém -> Upřesnit nastavení systému -> Upřesnit -> Výkon -> Nastavení



Obrázek 3: Zapnutí animací ve starém nastavení Windows 10 (vlastní)

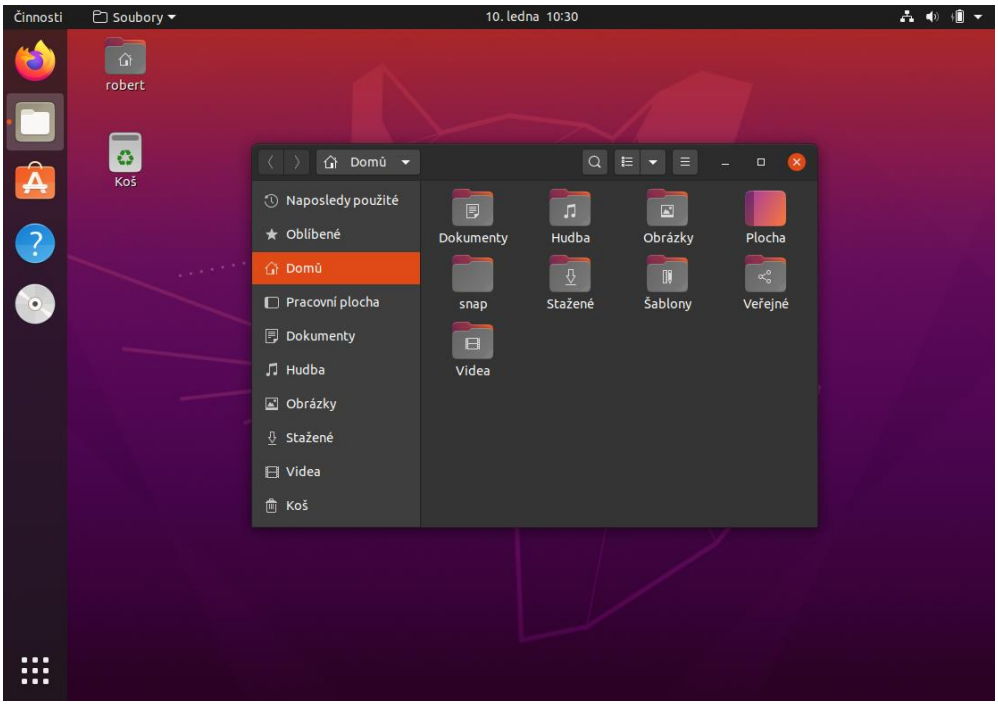
Moderní nastavení: Start -> Nastavení -> Usnadnění přístupu -> Obrazovka



Obrázek 4: Zapnutí animací v moderním nastavení Windows 10 (vlastní)

2.2.2 Ubuntu

Ubuntu staví na osvědčeném jádře Linux, které patří mezi nejrychleji se vyvíjející softwarové projekty vůbec. Linuxový základ posiluje stabilitu systému a umožňuje běh s většinou současného i staršího hardwaru. Díky praktikám dodržovaným v Linuxovém ekosystému je Ubuntu také velmi robustní a odolné vůči virům a přímým útokům z vnějšího prostředí. (Ubuntu 2021)



Obrázek 5: Ubuntu 20.10 (vlastní)

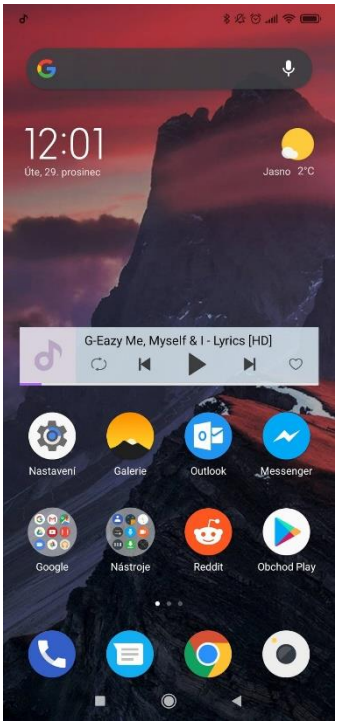
Ubuntu ve svých nejnovějších verzích používá GNOME (GNU Network Object Model Environment) jako prostředí pracovní plochy. GNOME tvoří uživatelské rozhraní operačního systému a pokrývá základní aplikace jako jsou: souborový manažer, prohlížeč obrázků, přehrávač hudby a videí, mapy, počasí, nastavení systému, kalkulačka, terminál a mnoho dalších základních aplikací.

GNOME ve svém grafickém uživatelském rozhraní nabízí animace pro otevření a zavření oken, zvětšení a zmenšení oken, notifikace, přepínání obrazovek a mnohé další. Většinou se jedná o animace s lineárním efektem.

Animace se dají i upravovat, a to buď pomocí konfiguračních souborů nebo pomocí volně dostupných doplňků, které se dají stáhnout přímo ze stránek GNOME.

2.2.3 Android

Android dnes na chytrých telefonech používá více než 70 % uživatelů. Za takto vysokým podílem na trhu stojí otevřenost Androidu. Každý výrobce chytrých telefonů nebo jiných zařízení si může Android upravovat, když dodrží stanovené podmínky. To znamená, že je systém většinou od každého výrobce jiný a tím se mohou lišit i animace, které výrobce implementoval do svého spouštěče (launcher), tedy aplikace, která obstarává domovskou obrazovku, grafická témata, widgety, seznamy aplikací a další základní grafické prvky operačního systému. Naštěstí Android obsahuje knihovny, které programátorům umožňují animovat ovládací prvky uživatelského rozhraní.



Obrázek 6: Ukázka OS Android s nadstavbou MIUI 12 (vlastní)

Když chceme animovat bitmapu, tak použijeme speciální knihovnu Drawable graphics. Tato knihovna je použitelná například na načítací animace, které se skládají z více obrázků a chceme zajistit jejich plynulý přechod.

Pro přesouvání nebo měnění průhlednosti prvků se používá knihovna animací poskytovaná balíčkem android.animation, který je dostupný už od Androidu verze 3.0. Tato knihovna aktualizuje vlastnosti vytvořených prvků a průběžně je překresluje, jak se vlastnosti mění. Například když se změní vlastnost polohy, tak se prvek přesune po obrazovce a když se změní vlastnost průhlednosti, tak se prvek objeví nebo zmizí. (Android developer)

Dále můžeme pomocí animací vytvářet plynulé přechody z jednoho rámce do druhého. Tato knihovna je dostupná od Androidu verze 4.4 a nese název Transition. Stačí ji předat počáteční a koncové rozložení a jaký typ animace se má použít. Poté se provede animace mezi dvěma rozloženími. Takto můžeme efektivně změnit uživatelské rozhraní a docílit efektivnější navigace v aplikaci.

2.3 Popis animací

Animace se skládají ze dvou hlavních částí: akce a efektu. S kombinací akcí a efektů můžeme docílit různých animací, ale ne všechny jsou zrovna vhodné k reálnému použití. Například pokud chceme posunout okno lineárně doleva, tak akce je přesun vlevo a efekt je lineární.

2.3.1 Akce animací

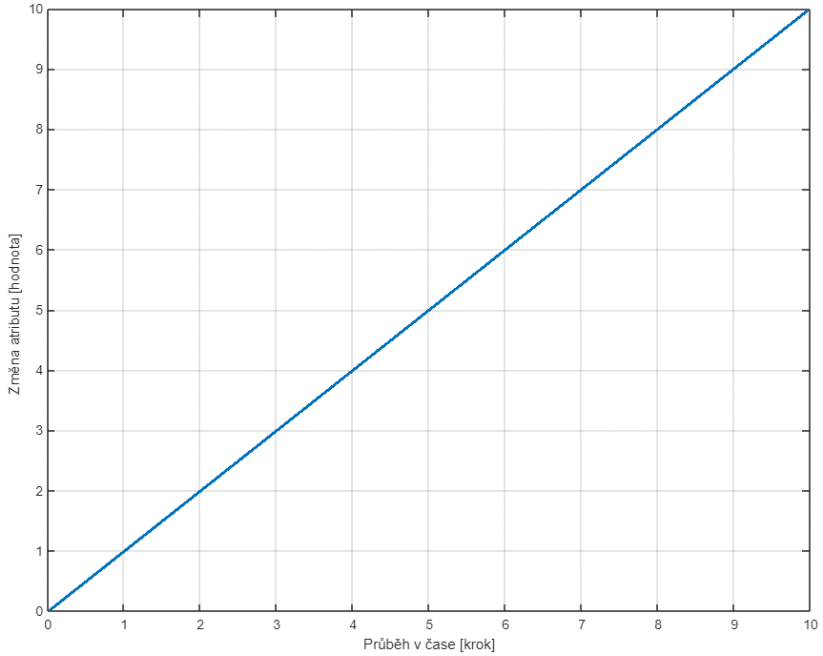
Akce animací můžeme rozdělit do těchto základních skupin:

- Přesunutí – doleva, doprava, nahoru, dolu
- Otočení – po ose x, po ose y
- Překlopení – po ose x, po ose y
- Změna velikosti – přiblížení, zvětšení, oddálení, zmenšení
- Změna barvy
- Změna průhlednosti – objevení, mizení

2.3.2 Efekty animací

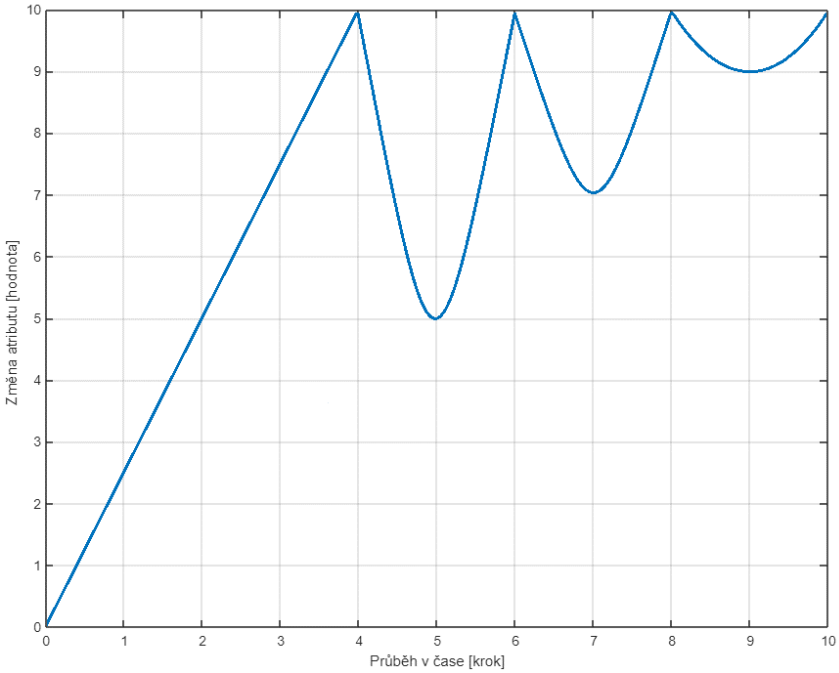
Efekty jsou další základní součástí systému animací, bez kterých by animace nemohly fungovat. Na následujících grafech jsou zobrazeny průběhy jednotlivých efektů animací. Atribut zastupuje souřadnice bodu, RGB složku nebo průhlednost.

Lineární efekt



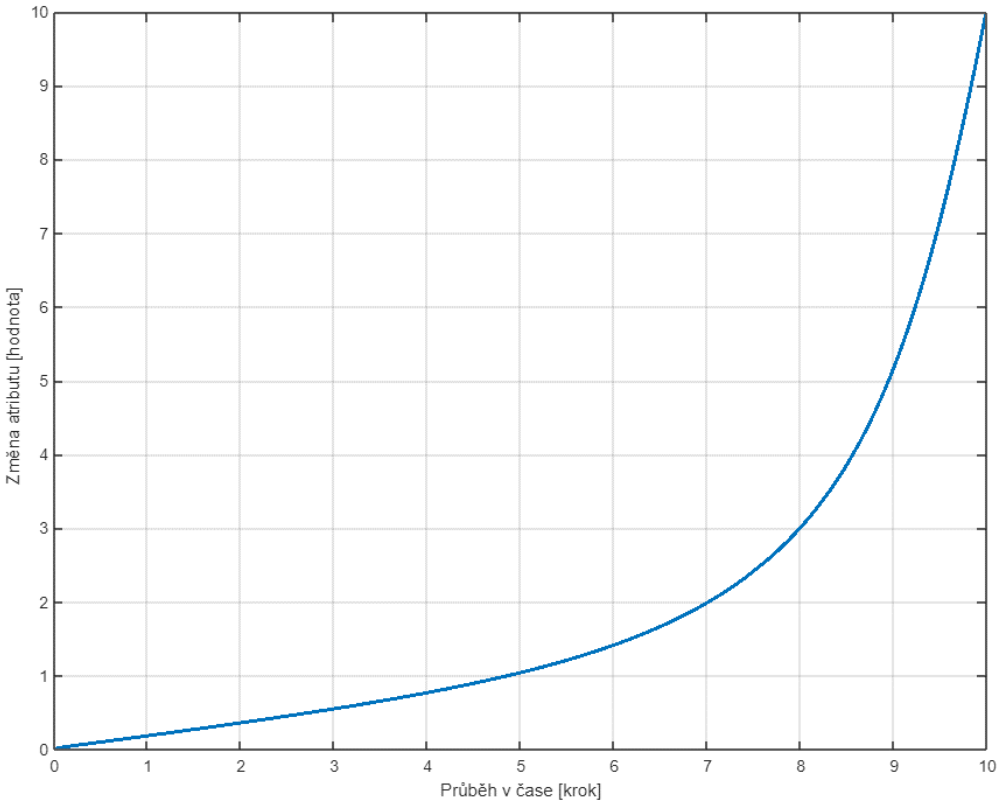
Graf 1: Průběh lineární animace (vlastní)

Odrážející efekt



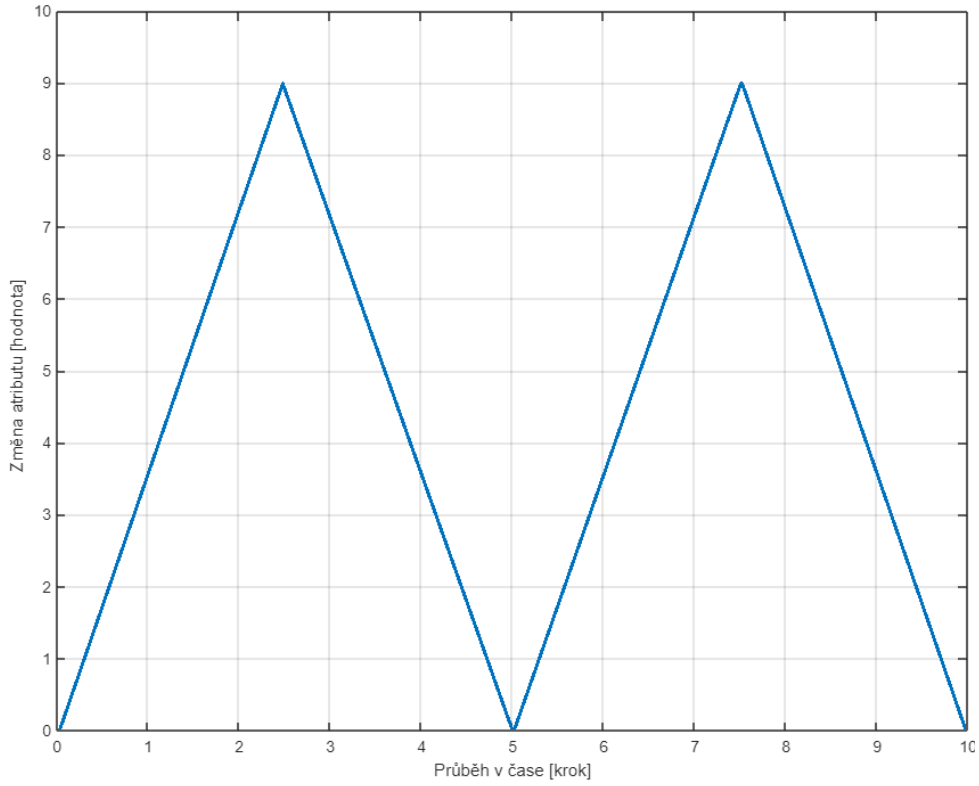
Graf 2: Průběh odrážející animace (vlastní)

Postupně zrychlující efekt



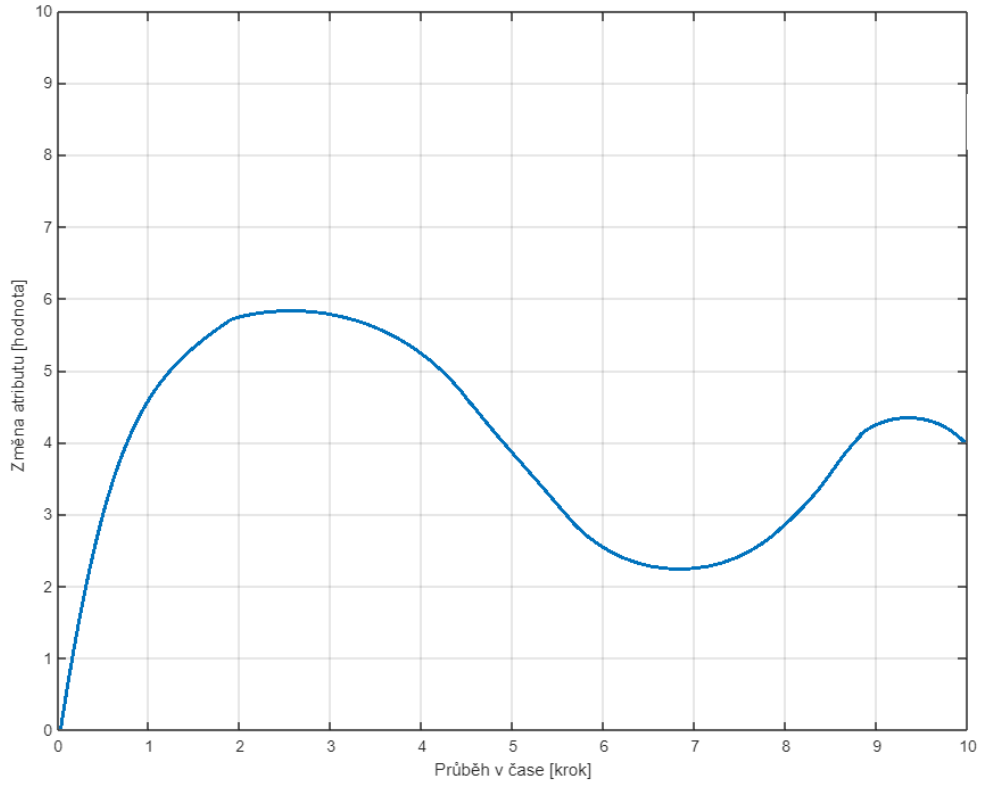
Graf 3: Průběh postupně zrychlující animace (vlastní)

Pulsující efekt



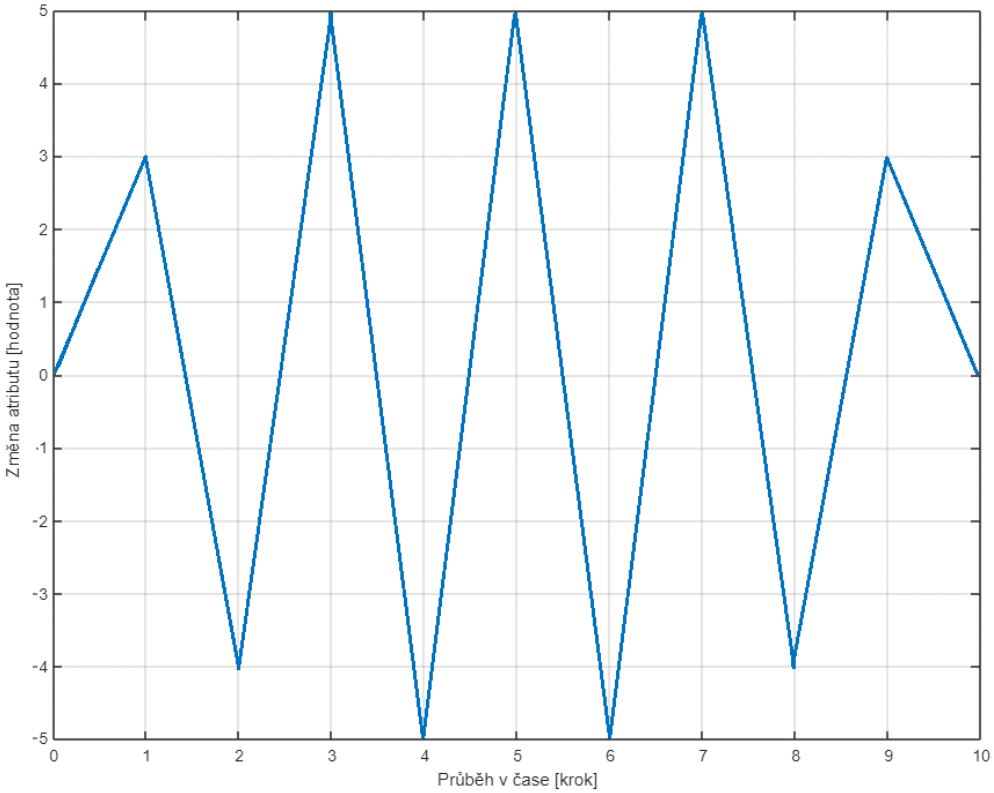
Graf 4: Průběh pulsující animace (vlastní)

Efekt natahovací gumy



Graf 5: Průběh animace natahovací gumy (vlastní)

Třepající se efekt

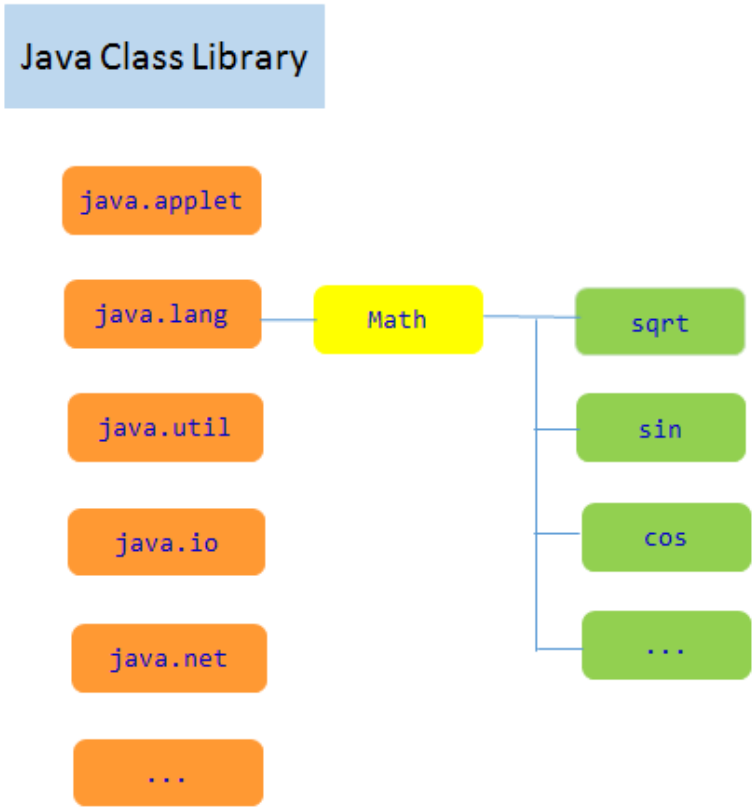


Graf 6: Průběh třepající se animace (vlastní)

2.4 Knihovny

Knihovna je označení pro sadu tříd, metod, struktur a konstant, které mohou být využívané různými programy. Jejich hlavní účel spočívá ve znovupoužitelnosti již napsaného kódu. Knihovna poskytuje své služby pomocí API (aplikační rozhraní), skrz které může programátor využívat různé funkcionality dané knihovny ve svém programu.

V Javě můžeme najít spoustu různých knihoven už v samotném JDK (Software Development Kit pro Javu), který mimo jiné obsahuje i JRE (prostředí pro běh programů), překladač, debugger, javadoc (tvorba dokumentace) a mnohé další. Program v Javě využívá spoustu základních knihoven (Java Core API), které je nutné mít při překladu a interpretaci programu. Pro přidání knihovny do řešení se používá klíčové slovo import.



Obrázek 7: Hierarchie knihoven v Javě (Vertex Academy)

2.5 Časovače

JDK (Software Development Kit pro Javu) obsahuje knihovny `java.util.Timer` a `java.util.TimerTask`, které pro nás budou nezbytnou součástí pro tvorbu animační knihovny pro SWT.

Třída Java `Timer` je bezpečná pro více vláken a více vláken může sdílet jeden objekt Timer bez nutnosti externí synchronizace. Třída časovače používá `java.util.TaskQueue` k přidávání úkolů v daném pravidelném intervalu a kdykoli může být spuštěno pouze jedno vlákno `TimerTask`, například pokud vytváříte časovač pro spuštění každých 10 sekund, ale provedení jednoho vlákna trvá 20 sekund, pak objekt `Timer` bude stále přidávat úkoly do fronty a jakmile je jedno vlákno hotové, upozorní frontu a začne se provádět další vlákno. (Pankaj, 2013)

Metody

`TimerTask`

- `boolean cancel()`
  - Zruší tuto úlohu
- `abstract void run()`
  - Akce, kterou úloha vykonává
- `long scheduledExecutionTime()`
  - Vrací naplánovaný čas spuštění posledního provedení úlohy

`Timer`

- `void cancel()`
  - Ukončí časovač a zruší všechny aktuálně naplánované úlohy
- `void purge()`
  - Odebere všechny zrušené úlohy z fronty úloh tohoto časovače
- `void schedule(TimerTask task, Date time)`
  - Naplánuje provedení úlohy na zadaný čas
  - Konstruktor je přetížený – jde zde zadat ještě zpoždění a opakování
- `void scheduleAtFixedRate(TimerTask task, Date firstTime, long period)`
  - Naplánuje zadanou úlohu pro opakované provádění s pevnou rychlostí, počínaje v zadaný čas

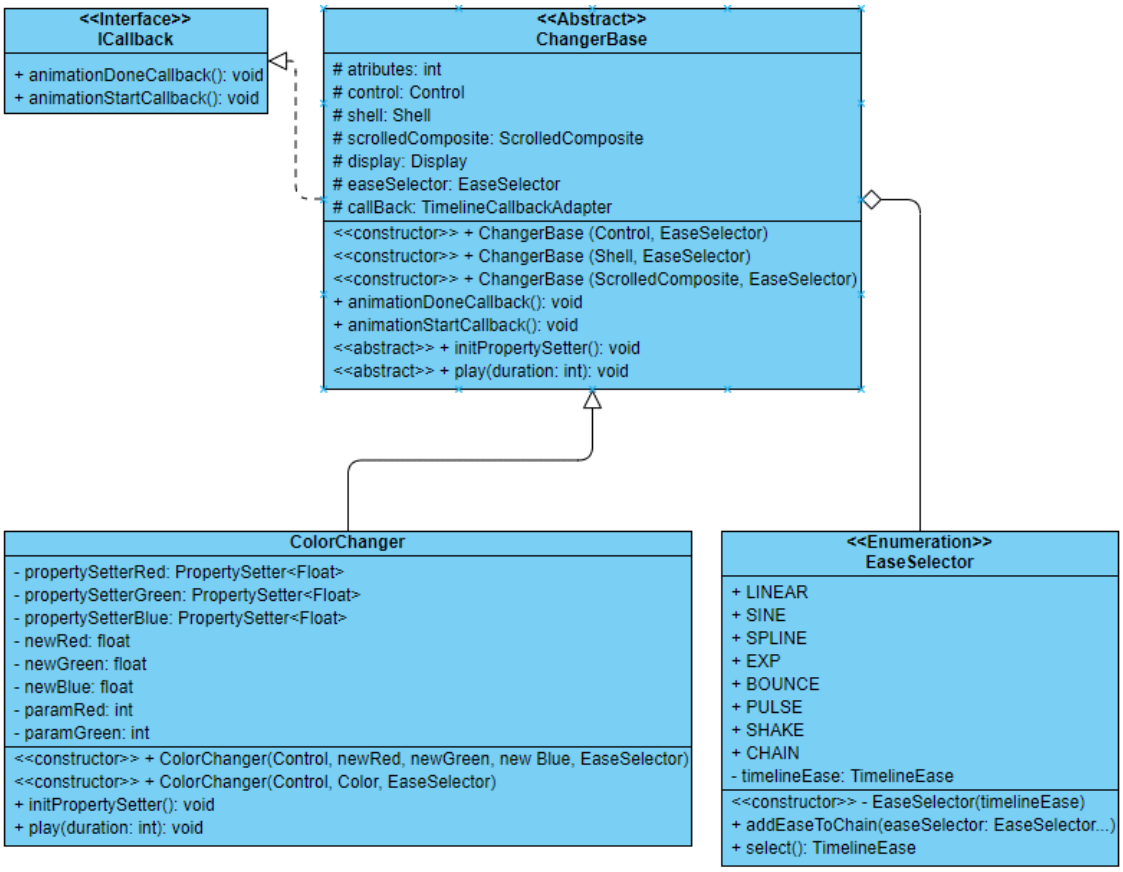
3 Navrhované řešení

V této kapitole popíšu, jak jsem konkrétně postupoval při návrhu knihovny animací pro SWT. Jelikož žádné z předchozích prozkoumávaných řešení nebylo vhodné pro knihovnu SWT, tak jsem se rozhodl navrhnout své vlastní řešení s využitím časových funkcí z knihovny Trident. Při návrhu této knihovny jsem se snažil dodržovat principy OOP (objektově orientovaného programování), které programovací jazyk Java podporuje. Veškeré části knihovny jsem rozdělil na obecné části, které spolu vzájemně komunikují. Těchto návrhů jsem se snažil držet při implementaci knihovny, ale nakonec jsem musel pár věcí upravit.

3.1 Návrh tříd

Při návrhu tříd jsem uvažoval, jak udělat strukturu mé knihovny animací pro SWT, co možná nejpřehledněji a nejefektivněji a kladl jsem důraz na možnosti případného rozšíření. Rozhodl jsem se, že knihovnu rozdělím do dvou balíků (package) z důvodu oddělení logické části a části s efekty animací knihovny.

V prvním balíku budou implementovány jednotlivé akce animací, jako jsou například změna velikosti ovládacího prvku nebo posouvaní se v tabulce. Tyto všechny akce animací budou dědit základní atributy, konstruktory a metody z rodičovské abstraktní třídy, která tyto prvky sdružuje. Jednotlivé třídy akcí animací budou dále obsahovat privátní položky, které jsou typické pro určitý typ akce. Například akce pro změnu barvy bude obsahovat datové položky pro jednotlivé složky barvy RGB, které se nastaví po dokončení animace. Dále zde bude implementováno rozhraní pro zpětné volání (callback), které bude moct uživatel využívat tak, že provede vlastní implementaci těchto metod, které se zavolají v ten okamžik, kdy animace začala anebo úspěšně skončila. Poslední položkou v tomto balíku bude výčtový typ (enum) pro vybírání animací, který bude obsahovat výčty všech efektů animací a metodu `select`, která bude vracet požadovaný efekt. Výčtový typ jsem zvolil hlavně kvůli snadnému přístupu k položkám, které obsahuje. Přes tečkovou notaci `Enum.HODNOTA`, mohu předávat informaci o zvoleném animačním efektu konstruktoru akce animace.



Obrázek 8: Zjednodušený diagram tříd (vlastní)

Na obrázku můžeme vidět zjednodušený diagram tříd balíčku `animations.swt`. V diagramu jsem zahrnul pouze třídu akce pro změnu barvy, abych tento diagram zpřehlednil. Další třídy, které se starají o další akce jako jsou změna pozice, změna průhlednosti, změna velikosti a rolování, jsou navrženy velmi podobně a mají v podstatě odlišné jen privátní položky a konstruktory.

Druhý balík bude zaměřen pouze na efekty animací. Tyto efekty budou implementovat rozhraní `TimelineEase`, které obsahuje metodu map. Metoda `map` přijímá na svém vstupu čas, ve kterém se animace nachází, což musí být hodnota mezi 0 a 1 a vrací taktéž hodnotu mezi 0 a 1, podle toho, jak je efekt animace implementovaný. Například pro lineární efekt animace je implementace této metody velmi snadné. Jednoduše stačí vracet na výstup vstupní hodnotu, čímž se docílí požadovaného lineárního efektu.

3.2 Návrh aplikačního rozhraní

Když jsem navrhoval rozhraní pro programování aplikací (API) pro moji knihovnu animací pro SWT, tak jsem přemýšlel, jak nabídnout uživateli, co nejjednodušší práci s touto knihovnou. Zvažoval jsem různé možnosti, a nakonec jsem se rozhodl, že se animace prvků budou vytvářet následující způsobem:

1. Vytvoření instance třídy některé akce animace
2. Této instanci se při vytváření předá prvek SWT, který se bude animovat, atributy pro nové hodnoty, které bude mít prvek po provedení animace a poslední atribut je efekt animace
3. Když je instance animace vytvořena, tak na ni stačí zavolat metodu play, které se předá doba trvání animace v milisekundách

```
// create instance of PositionChanger
PositionChanger pc = new PositionChanger(btnChangeMyPosition, randomNumber(0,
460), randomNumber(0, 337), EaseSelector.BOUNCE);
// play defined animation with 2s duration
pc.play(2000);
```

Další atributy, které jsou pro akci animace potřeba, je možné zjistit přímo z SWT prvku, který obsahuje metody pro vracení jeho atributů (getters). Pomocí těchto metod si třídy akcí zajistí veškeré další potřebné informace jako jsou například počáteční hodnoty akce.

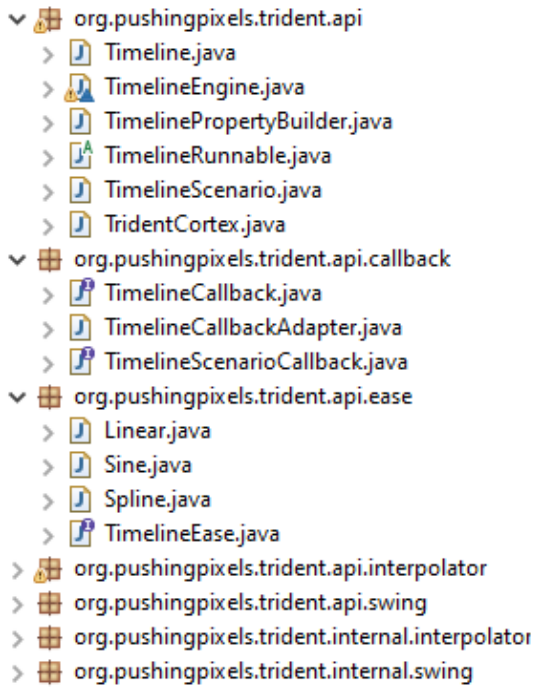
Pro zajištění operací, které jsou závislé na začátku nebo konci animace, bude mít uživatel knihovny možnost využít zpětná volání. Tyto zpětná volání bude moci implementovat přepsáním funkcí, které se nachází v rozhraní objektu. Přepsání těchto metod bude moci provést už při vytváření objektu animace pomocí notace `@Override`.

```
OriginChanger oc = new OriginChanger(scrolledComposite, x, y,
selector(comboScrolling.getSelectionIndex())) {
	@Override
	public void animationDoneCallback() {
		setScrollFlag(true);
	}

	@Override
	public void animationStartCallback() {
		setScrollFlag(false);
	}
};
```

3.3 Použití knihovny Trident

Při průzkumu řešení animací v grafických knihovnách jsem narazil na knihovnu Trident od autora Kirilla Grouchnikova. Tato knihovna dříve podporovala i SWT, ale autor ji sdružil s dalšími svými projekty, které se zabývají grafickými ovládacími prvky, do projektu Radiance a nyní se zaměřuje pouze na Java Swing. Po detailnějším průzkumu knihovny Trident jsem zjistil, že by se dali použít jeho časové funkce, které byly naprogramovány pro obecné použití a nevázaly se tedy na Swing. Po diskusi s vedoucím práce jsem se rozhodl knihovnu Trident využívat ve vlastním řešení. Využity v knihovně budou hlavně časové funkce, které se postarají o správný průběh animace. Tím pádem se nebudou muset používat časovače a veškerá režie vláken připadne právě na Trident. Dále zde bude využito rozhraní pro efekty animací a tři efekty, které jsou v knihovně Trident implementovány.

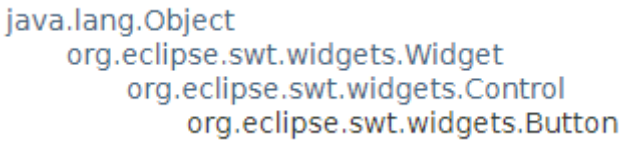


Obrázek 9: Struktura knihovny Trident (vlastní)

Knihovna Trident je vydána pod licencí BSD 3-Clause, takže není problém ji využívat či upravovat při splnění licenčních podmínek. Při rozhodování zda tuto knihovnu využívat ve vlastním řešení hrálo roli i to, že mohu prokázat orientaci v cizím kódu a schopnost využít takto rozsáhlou knihovnu.

3.4 Propojení knihoven

Knihovna animací pro SWT bude závislá na dalších dvou knihovnách, a to jsou SWT a Trident. Z knihovny SWT budu pracovat převážně s rodičovskou abstraktní třídou skoro všech SWT komponent, která se jmenuje Control. Třída obsahuje metody pro nastavení velikosti, pozice a barvy. Z této třídy dále dědí různé ovládací a grafické prvky.



Obrázek 10: Hierarchie tlačítka (vlastní)

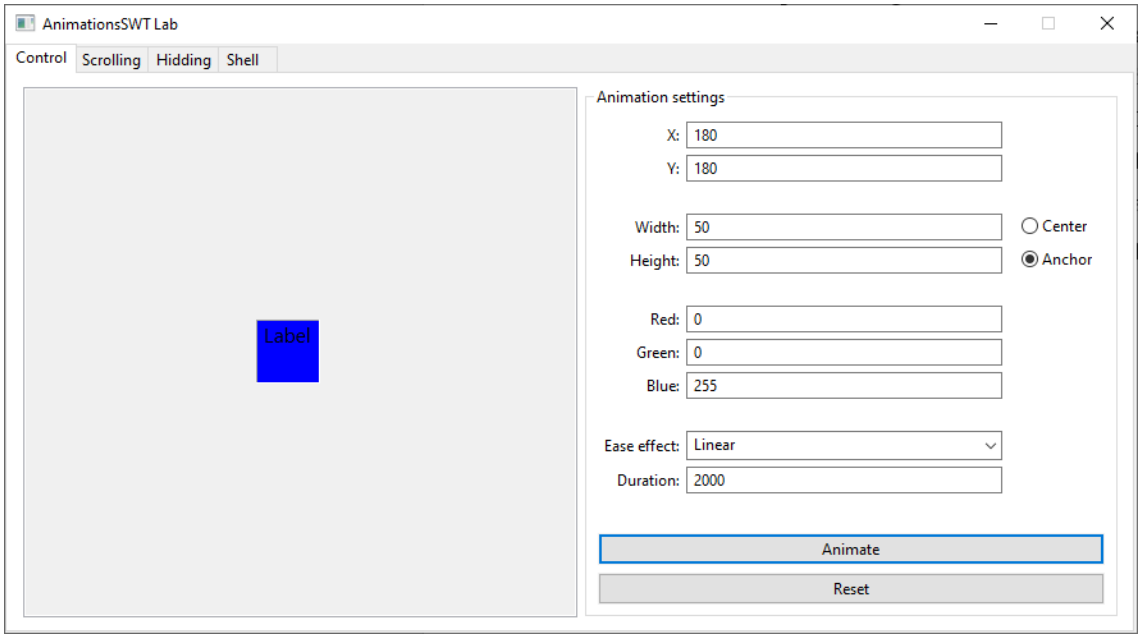
Dále z knihovny SWT využiji komponentu `ScrolledComposite` a `Shell`. `ScrolledComposite` se bude používat na animace, které se týkají rolování obsahu této komposity. Tato třída má implementovanou metodu `setOrigin`, která se stará o přemisťování rolovací lišty a obsahu komposity. Shell se bude využívat pro animace, které pracují s průhledností okna. Pomocí metody `setAlpha` se dá oknu nastavit průhlednost.

Z knihovny Trident budu využívat třídu `Timeline`, které předám informace o animaci společně s `PropertySetter`. Informace o animaci se skládají z následujících hodnot: vlastnost, která se má interpolovat, čas vykonávání animace, efekt animace a zpětné volání animace. Vlastnost ještě obsahuje informaci, s jakým `PropertySetter` a rozmezím hodnot se mají interpolovat. `PropertySetter` se stará o nastavování požadovaných hodnot buď do proměnných anebo může rovnou využít nějakou nastavovací metodu z SWT. Dále budu používat rozhraní `TimelineEase` pro implementaci vlastních efektů animací.

3.5 Návrh ukázkové aplikace

Pro ověření funkčnosti knihovny a testovací účely bylo potřeba vytvořit ukázkovou aplikaci, která obsahuje různé případy užití animací. Při návrhu aplikace jsem se snažil udělat čisté a přehledné grafické uživatelské rozhraní (GUI) pomocí knihovny SWT. Do aplikace jsem chtěl zakomponovat ty nejdůležitější animace a nechat na uživateli, aby si je mohl nastavit dle jeho preferencí. Při návrhu aplikace jsem používal plugin WindowBuilder do Eclipse IDE, což je WYSIWYG editor, který automaticky generuje

zdrojový kód SWT prvků, které si uživatel přidá a rozmístí v GUI tohoto pluginu. Abych udělal aplikaci co nejpřehlednější, tak jsem rozdělil obsah mezi jednotlivé záložky. Chtěl jsem demonstrovat základní animace typu změna velikosti, změna pozice a změna barvy, rolování v tabulce, používání zpětných volání a změnu průhlednosti okna.



Obrázek 11: Ukázková aplikace (vlastní)

V záložce Control se nachází základní animace pro jakýkoliv prvek SWT, který dědí ze třídy `Control`. Může se zde měnit pozice, výška a šířka, barevné složky, vybírat efekt animace a nastavit čas za který se animace vykoná. U změny velikosti je možnost si vybrat, jestli se má objekt zmenšovat či zvětšovat s uchyceným levým horním rohem nebo jestli má měnit velikost do všech stran.

Na záložce Scrolling se demonstruje rolování v tabulce za pomocí kolečka myši nebo stiskem tlačítka. U rolování lze nastavit efekt animace, délka jednoho rolu a přepínání vertikálního či horizontálního posuvníku.

V další záložce Hidding je ukázka chování animací v rozložení typu `Grid layout` s pomocí zpětného volání. Uživatel si nastaví efekt a délku animace a následně může skrývat či odkrývat prvky, které jsou zabaleny do `Composite`. Jakmile se animace provede, tak zavolá metodu zpětného volání, která je implementována tak, že `Composite` změní velikost podle počtu viditelných prvků.

Poslední záložka se věnuje otevírání a zavírání nového modálního okna. Okno, které je reprezentováno třídou Shell, má jako jediný SWT prvek možnost nastavení průhlednosti.

Proto jsem přidal i tuto možnost do demonstrující aplikace. Uživatel může nastavit efekt animace a její čas.

Ukázky vzhledu jednotlivých záložek aplikace jsou k nalezení v (Příloha A)

3.6 Přidání animací bez zásahu do zdrojové aplikace

Přidání animací bez nutnosti zásahu do zdrojového kódu původní aplikace jde jen velmi obtížně. Jedna z možností je úprava knihovny SWT a zabudování animací přímo do metod, které se starají o nějaké změny prvku. Takto upravená knihovna by se při překladu aplikace použila místo původního SWT. V tomto kódu je implementace metody pro nastavení umístění prvku typu `Control`.

```
public void setLocation (int x, int y) {
            checkWidget();
            x = DPIUtil.autoScaleUp(x);
            y = DPIUtil.autoScaleUp(y);
            setLocationInPixels(x, y);
}
```

Úpravou této metody lze snadno dosáhnout požadovaného chování, ale takto upravená knihovna by přinášela samé nevýhody a animace by se volala na každý prvek typu `Control`.

3.7 3-bodová BSD Licence

Jeden z požadavků od vedoucího bakalářské práce na knihovnu animací pro SWT byl, že knihovna musí být vyvíjena pod svobodnou licencí, aby se mohlo mé řešení volně využívat. Po konzultaci ve škole mi byla doporučena 3-bodová BSD licence, která tyto požadavky splňuje. Tato licence udává, že musí být ve zdrojovém kódu a v binárních souborech uvedeno jméno autora a informace o licenci. Dále zprošťuje autora a přispěvatele právní zodpovědnosti za škody, které vzniknou používáním produktu s touto licencí. Plné znění licence, která je využita v bakalářské práci, je uvedená v (Příloha B).

Licence je využívána ve všech souborech, které jsem vytvořil a obsahují zdrojový kód. Dále je licence obsažena v textovým souboru licence.txt, který je vložen do kořenových adresářů projektů (AnimationsSWT a AnimationsSWTdemo).

4 Implementace

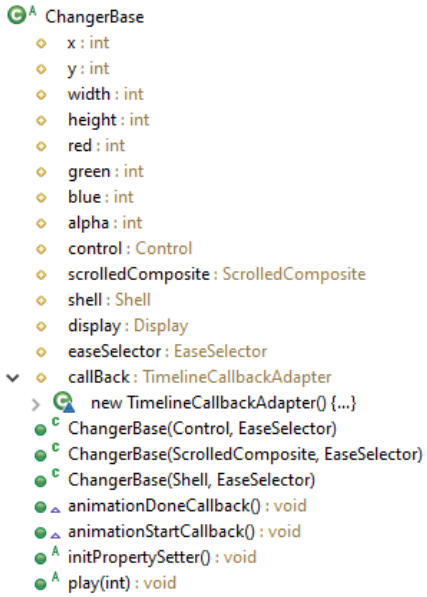
Na základě návrhu knihovny jsem ji a všechny potřebné součásti implementoval. Implementaci tohoto problému rozdělím na dvě části. Nejprve se budu věnovat implementaci systému animací, který bude sloužit zejména pro akce animací a poté se zaměřím na implementaci animačních efektů.

4.1 Implementace systému animací

Systém animací obsahuje třídy pro jednotlivé akce animací, třídu pro vybírání animací a rozhraní pro uživatelská zpětná volání.

4.1.1 Abstraktní rodičovská třída akce

Jedna z hlavních tříd pro animace `ChangerBase` je abstraktní třída, která implementuje rozhraní zpětných volání. Obsahuje tři přetížené konstruktory, které využívají třídy, které z ní dědí. Konstruktory jsou přetížené podle typu ovládacího prvku. Může se jednat o typ `Shell`, `ScrolledCompozite` nebo `Control`. Třída obsahuje mimo jiné i chráněné položky atributů ovládacího prvku.



Obrázek 12: Položky a metody ChangerBase (vlastní)

Pro zajištění zpětných volání je zde implementována instance třídy `TimelineCallbackAdapter`, ve které přepisuji metodu `onTimelineStateChange` a přidávám do ní své vlastní zpětné volací metody `animationStartCallback`

a `animationDoneCallback`, které jsou přebírány z rozhraní `ICallback`. Tyto metody nemají v základním stavu žádnou implementaci, takže je na volbě uživatele, jestli je bude chtít používat a případně přepsat u svého objektu.

```
/**
* Instance of TimelineCallbackAdapter to handle user callback
*/
protected TimelineCallbackAdapter callBack = new TimelineCallbackAdapter() {
	@Override
	public void onTimelineStateChanged(TimelineState oldState, TimelineState
	newState, float durationFraction, float timelinePosition) {
		if (newState == TimelineState.PLAYING_FORWARD) {
			display.asyncExec(()->animationStartCallback());
		} else if (newState == TimelineState.DONE) {
			display.asyncExec(()->animationDoneCallback());
		}
	}
};
```

Další jsou tu dvě klíčové metody `initPropertySetter` a `play`, které obstarávají nastavování nových hodnot a vykonávání animace. Tyto metody se liší u každého potomka, proto jsem metody nastavil jako abstraktní a implementuji je až u jednotlivých akcí animací.

4.1.2 Podtřídy akce

Tyto podtřídy dědí z `ChangerBase` a obsluhují už konkrétní animace. V následující tabulce popíšu jednotlivé třídy akcí, název jejich tříd a na jaký SWT prvek se aplikují.

Název třídy	Akce animace	SWT prvek
`AlphaChanger`	Změna průhlednosti okna	`Shell`
`ColorChanger`	Změna barvy pozadí	`Control`
`OriginChanger`	Rolování v kompositu	`ScrolledComposite`
`PositionChanger`	Změna pozice	`Control`
`SizeChanger`	Změna velikosti	`Control`

Tabulka 2: Třídy akcí animací (vlastní)

Na třídě `SizeChanger` demonstruji, jak akce animací funguje. Nejprve se deklarují privátní proměnné, které se budou následně v metodách využívat. Proměnné `newSizeWidth` a `newSizeHeight` budou sloužit k uchování hodnoty, která se má nastavit po konci animace. `PropertySetter` pro výšku a šířku se bude později ve třídě využívat k měnění velikosti objektu. Dále je zde pomocná proměnná `paramWidth`, která bude ukládat hodnotu šířky v probíhající animaci a proměnná center, podle které se

nastavuje, jestli změna velikosti bude vycházet ze středu objektu nebo bude ukotvený levý horní roh.

```
private PropertySetter<Float> propertySetterWidth;
private PropertySetter<Float> propertySetterHeight;
private int paramWidth;
private float newSizeWidth;
private float newSizeHeight;
private boolean center;
```

Následně je vytvořen konstruktor této tříd, který pomocí metody `super` provolá i konstruktor rodičovské třídy `ChangerBase` a tím inicializuje proměnné.

```
public SizeChanger(Control control, int newSizeWidth, int newSizeHeight, boolean
center, EaseSelector easeSelector) {
	super(control, easeSelector);
	this.newSizeWidth = newSizeWidth;
	this.newSizeHeight = newSizeHeight;
	this.center = center;
}
```

Dále se musí doimplementovat metody, které jsou klíčové pro provádění animací. První z nich je `initPropertySetter`. Tato metoda nastaví akci, která se bude provádět v průběhu animace. První `PropertySetter` postupně při animaci zapisuje hodnoty do pomocné proměnné `paramWidth`. U třídy `SizeChanger` je tato metoda trochu komplikovanější než u ostatních, protože SWT u metody `setSize` nebo `setBounds` mění velikost objektů tak, že je levý horní okraj ukotven na své pozici. Rozhodl jsem se přidat funkcionalitu, která umožní prvek roztahovat do všech stran současně. Tato funkce se zapíná parametrem `center`. Následně se kontroluje, jak je proměnná center nastavena a podle ní se zavolá druhý `PropertySetter` s metodou `setBounds` nebo `setSize`. Pokud je `center` nastaven na `true`, tak se provede metoda `setBounds`, která kromě změny velikosti dokáže změnit zároveň pozici prvku. K výpočtu nové pozice polohy prvku se použije následující vzorec.

$$nová\ pozice = \frac{\text{původní pozice} + (\text{původní velikost} - \text{velikost v průběhu animace})}{2}$$

Jestliže je `center` nastaven na `false`, tak se provede metoda `setSize`, které se předá pomocná proměnná šířky a hodnota výšky v průběhu animace.

```
@Override
public void initPropertySetter() {
	this.propertySetterWidth = (obj, fieldName, valueWidth) -> {
		paramWidth = (int)valueWidth.intValue();
	};
	if(center) {
		this.propertySetterHeight = (obj, fieldName, valueHeight) -> {
			display.asyncExec(()->
			control.setBounds(x + (width - paramWidth) / 2, y + (height -
			(int)valueHeight.intValue())/2, paramWidth,
			(int)valueHeight.intValue()));
		};
	} else {
		this.propertySetterHeight = (obj, fieldName, valueHeight) -> {
			display.asyncExec(()->
			control.setSize(paramWidth, (int)valueHeight.intValue()));
		};
	}
}
```

Dále je třeba implementovat ještě jednu metodu, která je u rodičovské třídy abstraktní, a to je metoda `play`. Tato metoda přebírá argument `duration`, který se používá pro nastavení času trvání animace v milisekundách. V této metodě se nejprve zavolá `initPropertySetter` a poté se vytvoří nová `Timeline`. Této instanci se musí pomocí metod přidat hodnoty, které se mají interpolovat. Hodnoty se skládají z: od jaké hodnoty má začít, do jaké hodnoty má dojít a jaký `PropertySetter` má využívat. Jako další se nastavuje doba trvání animace a ta se nastaví jednoduše z argumentu metody. Pro provedení úspěšné animace je také zapotřebí nastavit efekt animace. Ten je nastaven pomocí metody `setEase`, kterému se předá jako argument nová instance efektu. Zpětná volání jsou zajištěna metodou `addCallback` a metoda `build` vytvoří požadovanou animaci, která se následně spustí metodou `play`.

```
@Override
public void play(int duration) {
	initPropertySetter();
	Timeline timeline = Timeline.builder()
		.addPropertyToInterpolate(Timeline.<Float>property("valueWidth")
				.from((float)width)
				.to(newSizeWidth)
				.setWith(propertySetterWidth))
		.addPropertyToInterpolate(Timeline.<Float>property("valueHeight")
				.from((float)height)
				.to(newSizeHeight)
				.setWith(propertySetterHeight))
		.setDuration(duration)
		.setEase((TimelineEase)easeSelector.select())
		.addCallback(callBack)
		.build();
	 timeline.play();
}
```

4.1.3 Třída pro výběr animací

`EaseSelector` je výčtový typ a má v sobě nadefinováno, jaké efekty animací existují a pro každou položku je definována hodnotu, která se využívá v akcích animací.

```
LINEAR(new Linear()),
SINE(new Sine()),
SPLINE(new Spline()),
EXP(new Exponential()),
BOUNCE(new Bounce()),
PULSE(new Pulse()),
SHAKE(new Shake()),
CHAIN(new Chain(EaseSelector.BOUNCE, EaseSelector.LINEAR));
```

Dále obsahuje konstruktor, který nastaví požadovaný efekt do `TimelineEase` a metodu pro přidávání jednotlivých efektů do řetězového efektu animace. Tato metoda přijímá pole `EaseSelector` a nastaví sekvenci požadovaných efektů. Poslední metoda v tomto výčtovém typu je metoda `select`. Tato metoda se musí zavolat, když se požaduje nová instance nějakého efektu.

```
private TimelineEase timelineEase;
private EaseSelector(TimelineEase timelineEase) {
	this.timelineEase = timelineEase;
}
public EaseSelector addEaseToChain(EaseSelector... easeSelector) {
	timelineEase = new Chain(easeSelector);
	return EaseSelector.CHAIN;
}
public TimelineEase select() {
	return timelineEase;
}
```

4.1.4 Rozhraní pro zpětná volání

Toto rozhraní obsahuje pouze dvě metody `animationDoneCallback` a `animationStartCallback`, které se volají v rodičovské třídě akce animací a můžou být implementovány uživatelem knihovny přepsáním těchto metod.

4.2 Implementace efektů animací

Veškeré efekty animací implementují rozhraní `TimelineEase`, které obsahuje metodu `map`. Podrobněji jsem metodu map popsal v návrhové kapitole. Efekty animací můžeme rozdělit na dva typy. První efekt začíná v 0 a končí v 1, takže po vykonání akce se prvek nějakým způsobem změní. Druhý efekt změny začíná v 0 a zase v 0 končí. Tento typ efektu slouží například pro upozornění na nějakou změnu anebo když uživatel nechce, aby se prvek po skončení akce změnil.

4.2.1 Zrychlující se efekt

Tento efekt byl velmi jednoduchý na implementaci. Nejprve jsem chtěl použít exponenciální funkci, ale potřeboval jsem docílit toho, aby funkce vracela hodnoty v rozsahu 0 až 1. Proto je funkce x^2 ideální řešení.

```
@Override
public float map(float durationFraction) {
	return (float)Math.pow(durationFraction, 2);
}
```

4.2.2 Odrážející efekt

Odrážející efekt jsem implementoval pomocí rozdělení časové osy na 5 částí. V lichých částech jsem hodnoty zvyšoval, aby na konci části vracela funkce 1 a v sudých částech rozdělené časové osy jsem je naopak snižoval, čímž je docílen efekt skákání.

```
@Override
public float map(float durationFraction) {
	if(durationFraction <= 0.2) {
		return durationFraction * 5;
	} else if (durationFraction <= 0.4){
		return (float) (-2 * durationFraction + 1.4);
	} else if (durationFraction <= 0.6){
		return (float) (2 * durationFraction - 0.2);
	} else if (durationFraction <= 0.8){
		return (float) (- durationFraction + 1.6);
	} else {
		return (float) durationFraction;
	}
}
```

4.2.3 Efekt třepání

Tento efekt vrací na konci animace 0, takže se vrátí do původního stavu. Tato metoda je naimplementována tak, že je časová osa rozdělena na části kde návratová hodnota roste a na části kde klesá. Tyto dvě části jdou po sobě a 5krát se opakují.

```
@Override
public float map(float durationFraction) {
	int flag = (int) (durationFraction * 10) % 2;
	int b;
	if(flag == 0) {
		b = (int) (durationFraction * 10);
		return 10 * durationFraction - b;
	} else {
		b = (int) (durationFraction * 10) + 1;
		return - 10 * durationFraction + b;
	}
}
```

4.2.4 Pulzující efekt

Pulzující efekt funguje na podobném principu jako efekt třepání.

4.2.5 Zřetězený efekt

Tento efekt je speciální tím, že se může skládat z více efektů. Obsahuje vlastní konstruktor, který pomocí `varargs` přijímá pole efektů animací. V metodě `map` se nejprve zjistí počet efektů, které byli tomuto efektu předány. Poté pro tento počet efektů rozdělí časovou osu a hledá, jaký efekt má zrovna použít. Dále vrací přetransformovaný výsledek, který získá zavolání metody map na požadovaný efekt. Zřetězený efekt funguje správně jen na efekty, které na konci animace vrací hodnotu 1.

```
EaseSelector[] easeSelector;
public Chain(EaseSelector... easeSelector) {
	this.easeSelector = easeSelector;
}

@Override
public float map(float durationFraction) {
	int length = easeSelector.length;
	float fraction = (float)1 / length;
	for(int i = 0; i < length; i++) {
		if(durationFraction <= fraction * (i + 1)) {
			return easeSelector[i].select().map((float) (durationFraction -
			(i * fraction)) * length) / length + (i * fraction);
		}
	}
	return durationFraction;
}
```

4.3 Příklady použití

Pro lepší pochopení, jak se pracuje s mojí knihovnou animací pro SWT, jsem vytvořil pár jednoduchých příkladů. Toto řešení jsem zvolil, protože mě osobně už mnohokrát podobné příklady pomohly, když jsem se učil pracovat s novou technologií. Příklady jsem se snažil implementovat co nejjednodušeji a pokrýt co nejvíce případů použití. Příklady jsem pokryl následující použití:

- Změna pozice prvku
- Změna velikosti prvku
- Změna barvy prvku
- Rolování ve `ScrolledComposite`
- Práce se zpětným voláním
- Animované otevírání a zavírání okna
- Skrývání obsahu a vylučování z layoutu

Všechny příklady použití jsem implementoval do nového projektu, abych je oddělil od knihovny. V následují ukázce zdrojového kódu jde vidět použití animace na rolování ve `ScrolledComposite`, která si přidá `MouseWheelListener`, který zaručuje zavolání funkce `mouseScrolled`, když se pootočí s kolečkem na myši. Dalším krokem je zjistit jakým směrem se kolečko otočilo. Toto lze zjistit zavoláním metody `e.count`, která vrací 3 nebo -3 podle směru otočení (u Windows OS). Hodnotu, kterou získáme, ještě vynásobíme nějakou konstantou, abychom získali rozumnou hodnotu posunu, o který se bude animovat. Nakonec už stačí vytvořit instanci animace a spustit ji metodou play.

```
// add event listener
scrolledComposite.addMouseWheelListener(new MouseWheelListener() {
	public void mouseScrolled(MouseEvent e) {
		//count scroll distance
		int steps = 80;
		int x = (scrolledComposite.getOrigin().x - steps * e.count);
		int y = (scrolledComposite.getOrigin().y - steps * e.count);

		// create instance of OriginChanger
		OriginChanger oc = new OriginChanger(scrolledComposite, x, y,
		EaseSelector.BOUNCE);
		// play defined animation with 0.5s duration
		oc.play(500);
	}
});
```

4.4 Dokumentace pomocí JavaDoc

JavaDoc je utilita, která se využívá k automatickému generování dokumentace k programu. Využívá speciální druh komentáře, který se zapisuje takto:

```
/**
* THIS IS JAVADOC
*/
```

JavaDoc vygeneruje html soubory, které reprezentují jednotlivé třídy a odkazy mezi nimi. Jelikož jsou výstupem JavaDocu html soubory, tak je možné používat v komentářích html tagy. Kromě toho je možné předávat JavaDocu různé informace pomocí anotací.

Nejdůležitější anotace:

- `@author` – autor zdrojového kódu
- `@version` – verze zdrojového kódu
- `@param` – vstupní parametry metody
- `@return` – výstup metody
- `@throws` – výjimky
- `@see` – odkaz na třídu nebo metodu

Ve své knihovně jsem se snažil zdokumentovat všechny třídy, metody i proměnné. U tříd jsem psal krátký a výstižný popis a anotace pro autora a verzi.

```
/** Class for change position of SWT control element
 *
 * @author Robert Havranek
 * @version 1.2
*/
public class PositionChanger extends ChangerBase{

}
```

K proměnným jsem napsal pouze stručný popis, k čemu daná proměnná slouží a u metod jsem popsal jejich funkcionalitu, vstupní parametry a co metoda vrací.

```
/**
* Target coordinate y after animation
*/
private float newPositionY;

/**
* Constructor for class ChangePosition - initialization instance of ChangePosition
*
* @param control SWT control element
* @param newPositionX Coordinate x of the element after the animation is completed
* @param newPositionY Coordinate y of the element after the animation is completed
* @param ease Ease effect of animation
*/
public PositionChanger(Control control, int newPositionX, int newPositionY,
EaseSelector easeSelector) {
	super(control, easeSelector);
	this.newPositionX = newPositionX;
	this.newPositionY = newPositionY;
}
```

Závěr

V této bakalářské práci na téma Knihovna animací pro ovládací prvky knihovny SWT se vydařilo teoreticky shrnout problematiku, která se týká SWT a animací. Na začátku práce je pozornost věnována hlavně knihovně SWT, kterou mám za úkol vylepšit právě o systém animací.

V následující části jsem úspěšně provedl rozbor existujících knihoven animací nejen pro knihovnu SWT a zjistil jsem, jaké nabízejí možnosti a mohl se tak inspirovat pro budoucí implementaci mého vlastního řešení. Bohužel jsem zjistil, že jediná dostupná animační knihovna pro SWT (Trident) se přestala samostatně vyvíjet a autor oficiálně ukončil podporu SWT. Po komunikaci s autorem jsem zjistil, že podporu ukončil, protože spojoval více knihoven grafických ovládacích prvků a chtěl se zaměřit pouze na modernější Swing, který může sloužit jako alternativa k SWT. Dále jsem se zaměřil, na animace v operačních systémech a zjistil jsem, že mobilní platformy mají nad desktopovými výrazně navrch.

V poslední teoretické části práce jsem udělal rozbor fungování animací a základní analýzu knihovny Timer a TimerTask. Tyto rozbory mi dále posloužily pro návrh a implementaci mého řešení.

Návrhová část této práce podrobně rozebírá návrhy jednotlivých tříd, API a vzorové aplikace. také je zde rozebíráno proč a jakým způsobem je nakonec zakomponována knihovna Trident v mé animační knihovně.

Dále jsem rozebíral způsob, kterým se dají přidat animace do aplikace i bez zásahu do zdrojového kódu původní aplikace a popsal 3-bodovou BSD licenci, kterou používám ve zdrojových kódech, jelikož jeden z požadavků od vedoucího práce byl, aby knihovna byla vydána pod svobodnou licencí.

V další části se už zabývám samotnou implementací systému animací a animačních efektů. Podařilo se implementovat nové efekty animací i různé animační akce, které se dají spolu kombinovat. Také je zde popsáno, jak využívám JavaDoc dokumentaci.

Všechny nastavené cíle se podařilo naplnit. Výstupem této práce je tedy plně funkční knihovna pro animování ovládacích prvků knihovny SWT. Navíc jsem ještě udělal jednoduché ukázky použití jednotlivých funkčností knihovny.

Seznam použité literatury

ANDROID DEVELOPER. Animation [online]. Android, 2020 [cit. 2020-12-22]. Dostupné z: https://developer.android.com/training/animation/overview

ECLIPSE FAQ. *What is SWT?* [online]. Eclipse Foundation, 2009 [cit. 2020-11-28]. Dostupné z: https://wiki.eclipse.org/FAQ_What_is_SWT%3F

ECLIPSE. *SWT Library* [online]. Eclipse Foundation [cit. 2020-11-12]. Dostupné z: https://www.eclipse.org/swt/

GROUCHNIKOV, Kirill. *Radiance libraries* [online]. github, 2018 [cit. 2020-11-28]. Dostupné z: https://github.com/kirill-grouchnikov/radiance

PANKAJ. *Java Timer TimerTask example* [online]. JournalDev, 2013 [cit. 2020-1-8]. Dostupné z: https://www.journaldev.com/1050/java-timer-timertask-example

SAMPIETRO, Mauro. *Control Animation in Winforms* [online]. CodeProject.com, 2014 [cit. 2020-11-28]. Dostupné z: https://www.codeproject.com/Articles/827808/Control-Animation-in-Winforms

TYPHON0. *AnimateFX library* [online]. github, 2017 [cit. 2020-11-28]. Dostupné z: https://github.com/Typhon0/AnimateFX

UBUNTU. *Desktop* [online]. Canonical Ltd, 2021 [cit. 2021-1-10]. Dostupné z: https://www.ubuntu.cz/desktop/

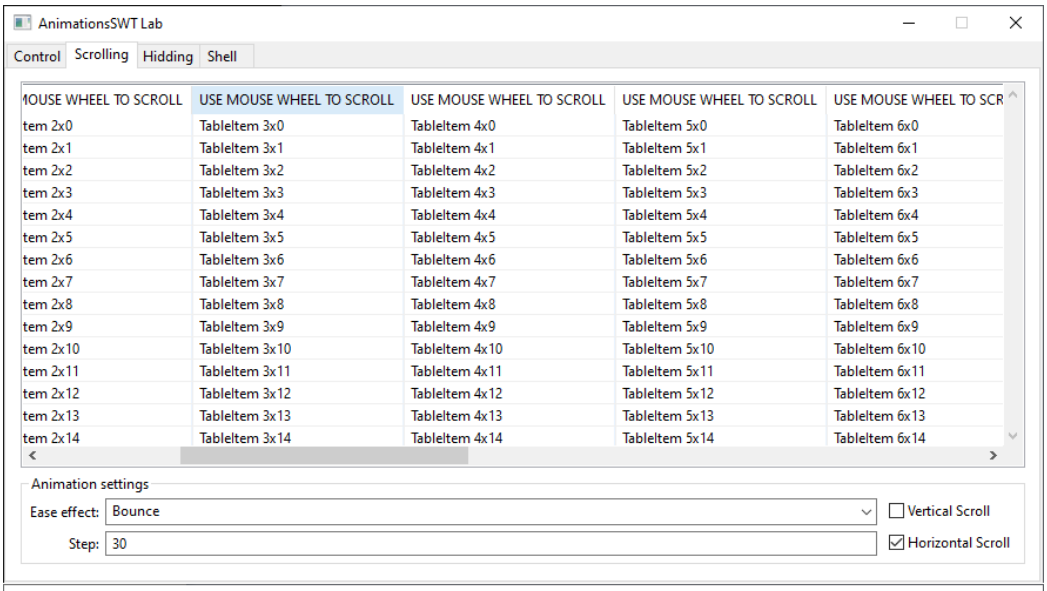
VERTEX ACADEMY. *What is library* [online]. Vertex Academy, 2016 [cit. 2021-1-9]. Dostupné z: https://vertex-academy.com/tutorials/en/what-is-java-library/

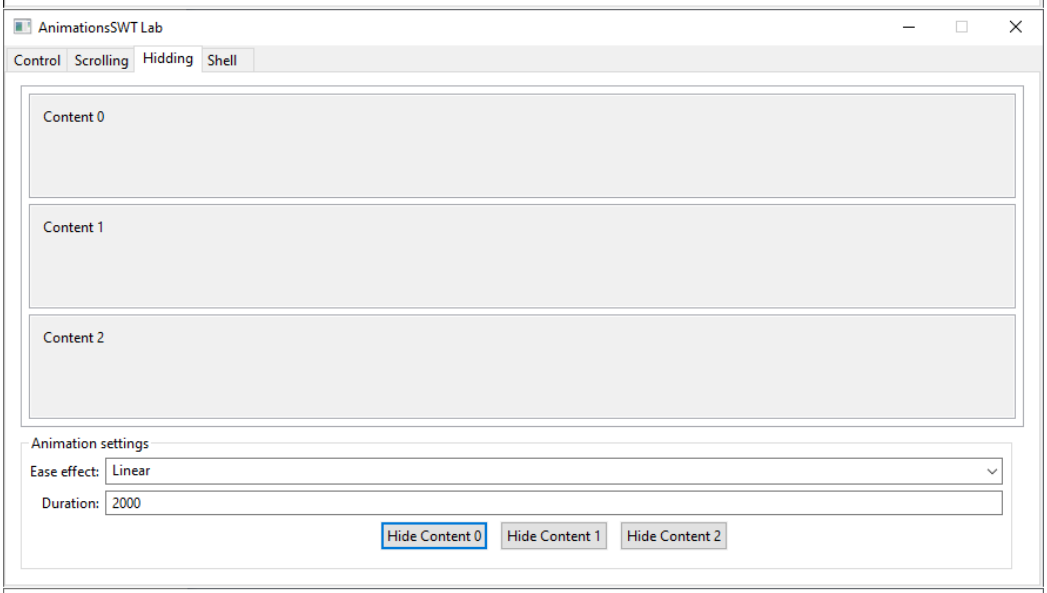
Přílohy

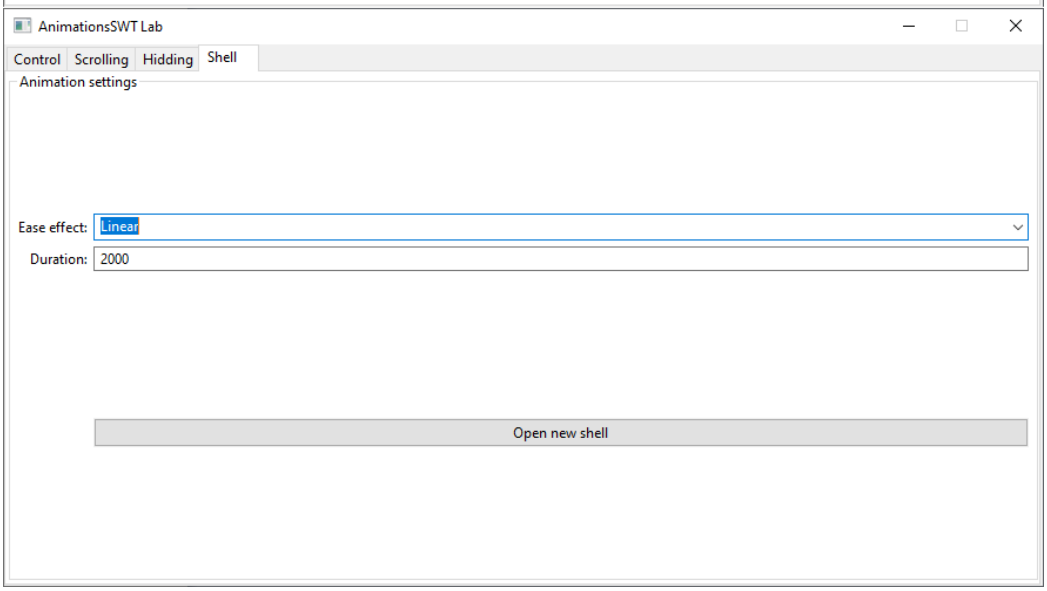
Příloha A **Jednotlivé záložky demonstrující aplikace**

Příloha B **BSD licence použita v bakalářské práci**

Příloha A


AnimationsSWT Lab
Control
Scrolling
Hidding
Shell
USE MOUSE WHEEL TO SCROLL
TableItem 3x0
TableItem 4x0
TableItem 5x0
TableItem 6x0
Animation settings
Ease effect:
Bounce
Step:
30
Vertical Scroll
Horizontal Scroll


AnimationsSWT Lab
Control
Scrolling
Hidding
Shell
Content 0
Content 1
Content 2
Animation settings
Ease effect:
Linear
Duration:
2000
Hide Content 0
Hide Content 1
Hide Content 2


AnimationsSWT Lab
Control
Scrolling
Hidding
Shell
Animation settings
Ease effect:
Linear
Duration:
2000
Open new shell

Příloha B