Vývoj aplikací s využitím JavaFX

Developing applications with JavaFX

bakalářská práce

František Sedláček

Vedoucí bakalářské práce: RNDr. Jaroslav Icha

Jihočeská Univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra informatiky

2011

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Pedagogická fakulta
Katedra informatiky
Akademický rok: **2008/2009**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **František SEDLÁČEK**

Studijní program: **B1802 Aplikovaná informatika**

Studijní obor: **Výpočetní technika**

Název tématu: **Vývoj aplikací s využitím JavaFX**

Zásady pro vypracování:

Cílem práce je vytvořit kolekci aplikací, které budou demonstrovat technologické postupy a možnosti, které nabízí pro vývoj multimediálních aplikací nový skriptovací jazyk JavaFX. V poslední době jsme svědky značného zájmu, kterému se těší používání tzv. skriptovacích jazyků. Od vytváření krátkých skriptů spouštěných z příkazového řádku se zájem přesouvá na jazyky, které umožňují vytvářet i multimediální aplikace s bohatým grafickým rozhraním. Programovací jazyk JavaFX od firmy SUN je příkladem takového jazyka, který se po svém uvedení v v minulém roce setkal se značným zájmem. JavaFX poskytuje jednotný vývojový a distribuční model pro vytváření a distribuci aplikací s bohatým grafickým rozhraním určených pro různou škálu klientských zařízení.
Při zpracování zadaného tématu bude řešitel vycházet z následujících dílčích cílů.

1. Na základě uvedených zdrojů popíše autor stručně technologii JavaFX a doprovodí tento popis konkrétní praktickou ukázkou.
2. Autor navrhne strukturu typů aplikací, které bude ve své práci vyvíjet.
3. Autor popíše vlastní vývoj modelových aplikací s tím, že tento popis bude zahrnovat jak návrh grafického rozhraní a aplikační logiky, tak i vlastní implementaci.
4. Pro vývoj aplikací se předpokládá použití vývojového prostředí NteBeans.

Rozsah grafických prací:

Rozsah pracovní zprávy: **60**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

1. **JavaFX Technology At a Glance [online]. 2008 [cit. 2009-04-01]. Dostupný z WWW: <http://java.sun.com/javafx/index.jsp>.**
2. **JavaFX Technologies At a Glance [online]. 2008 [cit. 2009-04-01]. Dostupný z WWW: <http://java.sun.com/javafx/technologies/>.**
3. **JavaFX Reference - Documentation, Tutorials, & APIs [online]. 2008 [cit. 2009-04-01]. Dostupný z WWW: <http://java.sun.com/javafx/reference/docs.jsp>.**
4. **NetBeans IDE 6.5 Features [online]. 2008 [cit. 2009-04-01]. Dostupný z WWW: <http://www.netbeans.org/features/javafx/index.html>.**
5. **Welcome to the site of People who have Passion for Java Technology! [online]. 2009 [cit. 2009-04-18]. Dostupný z WWW:<http://www.javapassion.com/>.**

Vedoucí bakalářské práce: **RNDr. Jaroslav Icha**
Katedra informatiky

Datum zadání bakalářské práce: **20. dubna 2009**

Termín odevzdání bakalářské práce: **30. dubna 2010**

L.S.

doc. PhDr. Alena Hošpesová, Ph.D.
děkanka

PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 20. dubna 2009

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě Pedagogickou fakultou elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne

Abstrakt

Tato bakalářská práce se zabývá vývojem aplikací pro softwarovou platformu JavaFX. Je zde představena platforma JavaFX včetně informací o vzniku a historii. Dále jsou vysvětleny nejdůležitější rysy jejího programovacího jazyka JavaFX Script. Následuje popis vývoje jednotlivých ukázkových aplikací, které demonstrují možnosti softwarové platformy JavaFX a práci s programovacím jazykem.

Jednotlivé ukázkové aplikace byly pro tuto práci navrženy tak, aby doplnily studijní materiál při výuce programovacího jazyka Java. Konkrétně se jedná o počítačovou hru, aplikaci pro odpočítávání času, převaděč měn podle aktuálních kurzů a animovaný reklamní banner. Čtenář by měl pomocí této práce získat dostatek informací pro objasnění, jakým způsobem a jaký typ aplikací lze pro platformu JavaFX vytvářet.

Abstract

The paper deals with development of applications for the software platform called JavaFX. The JavaFX platform is presented here, including information on the origins and history. Next are explained the most important features of the JavaFX Script programming language. Then follows a description of the development of sample applications that demonstrate the possibilities of software platforms and work with the JavaFX programming language.

Each sample application for this work were designed to supplement the study material for learning the Java programming language. That sample applications are specifically, the computer game, the countdown timer, currency converter based on current rates and animated banner. The reader should use this study to obtain sufficient information to explain how and what type of applications can be made for JavaFX platform.

Klíčová slova

JavaFX, software, platforma, vývoj, aplikace, zařízení

Keywords

JavaFX, software, platform, development, application, device

Poděkování

Rád bych poděkoval RNDr. Jaroslavu Ichovi za vstřícnost a cenné rady při vedení mé bakalářské práce.

Obsah

1 Úvod

Tato práce se zabývá vývojem tzv. RIA aplikací (Rich Internet Applications) pro softwarovou platformu JavaFX. První část práce popisuje technologii JavaFX a seznamuje čtenáře s architekturou, historií a principy fungování. Dále následuje představení programovacího jazyka JavaFX Script, který se používá pro vývoj aplikací JavaFX (Zdroj [16]). Předpokládá se, že čtenář je obeznámen s problematikou programování v jazyku Java, a proto je v této práci předložen pouze výčet odlišností a popis nových konstrukcí jazyka JavaFX Script.

Další část práce se již věnuje rozboru vývoje ukázkových aplikací. Popis jednotlivých aplikací je členěn na návrh tématu a činnosti aplikace, návrh uživatelského rozhraní, dále následuje popis vlastní implementace, zhodnocení teoretické přenositelnosti aplikace a shrnující závěr.

Práce rozebírá čtyři aplikace, které byly navrženy tak, aby každá představila trochu jinou oblast JavaFX. Motivace pro tvorbu konkrétních aplikací je vysvětlena v části návrhu každé aplikace. Některé postupy a programové konstrukce jsou společné pro více než jednu aplikaci. Ty unikátní jsou patřičně zdůrazněny. V textu práce jsou používány části zdrojového kódu pro názornější vysvětlení jednotlivých konstrukcí a postupů. Zdrojový kód aplikací, který je součástí přílohy této práce, je patřičně okomentován a ke každé aplikaci byla vytvořena dokumentace Javadoc. V rámci této práce jsem také vytvořil internetové stránky, kde lze jednotlivé aplikace spustit jako webové. Tyto stránky jsou umístěny na adrese http://javafx.xoe.cz.

2 JavaFX

2.1 Softwarová platforma JavaFX

2.1.1 Úvod

Softwarová platforma JavaFX byla vyvinuta společností SUN Microsystems. Na začátku nesl tento projekt označení „F3“ a byl pouze jakýmsi návrhem technologie, která by sloužila běhu RIA aplikací, byla by schopna výrazným způsobem podporovat multimedia a snadnost tvorby graficky přitažlivých aplikací.

Projekt „F3“ měl na starosti vývojář společnosti SUN Microsystems Christopher Oliver (Zdroj [10]). Označení „F3“ bylo brzy opuštěno a projekt dostal sebevědomější název, JavaFX. V květnu 2007 tak byla na konferenci Java vývojářů představena odborné veřejnosti již rodící se platforma JavaFX. První oficiální verze JavaFX 1.0 však spatřila světlo světa až v prosinci 2008. Jednotlivé verze během měsíců postupně získávaly nové schopnosti. Pro úplnost bych jen uvedl, že tato bakalářská práce se zaměřuje na využití platformy JavaFX verze 1.3 resp. 1.3.1 z roku 2010. (Zdroj [8])

Od začátku byla platforma JavaFX pro vývojáře navržena tak, aby se mohli soustředit především na kreativní část práce a nemuseli se starat například o to, jakým způsobem budou jednotlivé grafické prvky aplikace vykreslovány apod. Tuto filozofii si ukážeme na příkladech jednotlivých aplikací.

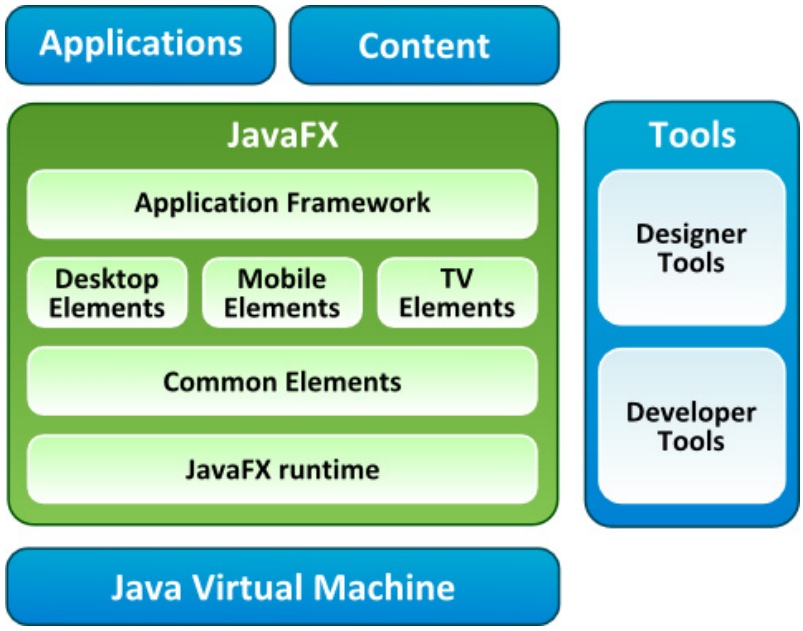
Naopak mezi nevýhody lze určitě počítat ten fakt, že jednotlivé verze JavaFX mezi sebou nejsou binárně kompatibilní. To znamená, že například aplikace napsaná a přeložená pro platformu ve verzi 1.2 nelze spustit na platformě JavaFX 1.3. Tato poněkud neobvyklá komplikovanost svědčí o určité nevyzrálosti a mládí platformy jako takové. (Zdroj [15])

Obrázek 2.1: Slogan společnosti SUN Microsystems vyjadřující přenositelnost aplikací Java a JavaFX na různá klientská zařízení, (Zdroj [32])

2.1.2 Architektura platformy

Platforma JavaFX se skládá z několika částí a pro své fungování zároveň vyžaduje určité prostředí. Architektura platformy JavaFX je ukázána na obrázku 2.2. Z tohoto schématu je vidět, že se JavaFX skládá z následujících částí:

- *Application Framework* – Aplikační rámec poskytující základní prostředky pro fungování aplikací
- *Desktop Elements* – API specifické pro běh aplikací na PC
- *Mobile Elements* – API obsahující knihovny pro běh na mobilních zařízeních
- *TV Elements* – API specifické pro platformu televizních zařízení
- *Common Elements* – API, které je společné pro všechna cílová zařízení
- *JavaFX Runtime* – Multiplatformní běhové prostředí



Obrázek 2.2: Architektura platformy JavaFX (Zdroj [30])

Z obrázku 2.2 vyplývá, že celá platforma JavaFX závisí na fungování *Java Virtual Machine*, což znamená, že pro spouštění aplikací JavaFX je v systému nutná přítomnost běhového prostředí Java (Zdroj [12]). Dále z tohoto spojení plyne fakt, že aplikace JavaFX mohou využívat knihoven standardní Javy. Tím pádem API JavaFX nemusí obsahovat znovu

úplně ty stejné knihovny jako Java. V aplikacích JavaFX je tedy bez problémů možné využívat například standardní Java třídu `Math` a její metody.

Dále je z obrázku 2.2 patrné, že existují nástroje pro vývoj aplikací JavaFX a také nástroje pro přípravu grafiky. JavaFX má dokonce vlastní formát pro kompresi grafických souborů. Tento formát se označuje jako FXZ a je podporován některými grafickými editory. (Zdroj [18])

Ze softwarových nástrojů jsem vyzkoušel pouze vývojový nástroj NetBeans, který, dle mého názoru, nabízí dostatečný komfort pro vývoj JavaFX aplikací a nabízí mj. i možnost zvolit cílový spouštěcí profil aplikace. Aplikaci lze tedy vyzkoušet jako desktopovou, webovou, mobilní a nebo televizní.

2.1.3 JavaFX API

Rozhraní pro programování aplikací JavaFX je rozděleno na dva profily. První se označuje jako `common` a je společný pro všechna cílová zařízení. Druhý je nazván `desktop` a v podstatě přidává k profilu `common` další třídy, jejichž fungování by bylo jinde než na desktopu problematické. Jedná se především o třídy některých vizuálních efektů, které buď pro profil `common` vůbec nejsou definovány nebo jejich použití nemá žádný výsledný efekt.

API JavaFX je poměrně rozsáhlé a obsahuje velké množství knihoven. Neuvádím zde proto jejich výčet, ale odkazuji čtenáře na přehled celého API JavaFX ve verzi 1.3.1 ze zdroje [21].

2.1.4 Podpora mobilních zařízení

Výhodou platformy JavaFX je přenositelnost aplikací na širokou škálu klientských zařízení bez nutnosti změny kódu samotných aplikací. To znamená, že aplikace fungující na běžném PC poběží například i na mobilním telefonu a nebo zkrátka všude, kde má k dispozici prostředí JavaFX. Tato úvaha má jediný háček. Tím je nedostatečná podpora mobilních zařízení. V dnešní době (rok 2010, 2011) například neexistuje verze JavaFX pro operační systém Android. Pro mobilní operační systém Windows Mobile existuje pouze JavaFX verze 1.2, která, jak již bylo zmíněno, není binárně kompatibilní s aktuální verzí 1.3. Tyto skutečnosti brzdí JavaFX v rozletu a naopak nahrávají konkurenci. (Zdroj [17])

I přes tuto nevýhodu jsem měl možnost vyvíjené aplikace vyzkoušet alespoň na emulátoru JavaFX pro mobilní zařízení, který funguje jako součást vývojového prostředí NetBeans.

2.2 Jazyk JavaFX Script

2.2.1 Datové typy

Jazyk JavaFX Script je staticky typovaný programovací jazyk (Zdroj [25]). Znamená to, že při překladu musí být kompilátoru znám typ každé proměnné i návratový typ každé funkce. Pokud programátor typ neuvede, kompilátor se jej pokusí odhadnout z kontextu zdrojového kódu (Zdroj [22]). Jestliže se mu to však nepovede, ohlásí chybu překladu. Konkrétní datové typy jazyka JavaFX Script jsou uvedeny v tabulce 1.

Datový typ	Popis	Příklad
`String`	Řetězec znaků	`'abcd'` nebo `"abcd"`
`Integer`	Celá čísla	`1984`
`Number`	Desetinná čísla	`11.7`
`Boolean`	Reprezentuje dvě hodnoty	`true` nebo `false`
`Duration`	Definuje časový úsek	`250ms; 3s; 9m; 1h;`
`Void`	Funkce nevrací hodnotu	`function play(): Void`

Tabulka 1: Přehled datových typů jazyka JavaFX Script

Při deklaraci proměnných se používá zápis, který ukazuje kód 1.

Kód 1 *Deklarace proměnných resp. objektů pomocí JavaFX Script*

```
var promenna = hodnota;
/* nebo... */
var promenna: Typ = hodnota;
/* a konkretne napr... */
var cislo: Integer = 1984;
```

Definování konstant se provádí pomocí klíčového slova `def`. Příklad použití ukazuje zdrojový kód 2.

Kód 2 *Příklad definování konstanty v jazyce JavaFX Script*

```
def mojeKonstanta: Integer = 1984;
```

Dále je třeba také zmínit způsob, jakým se dají proměnné vložit do řetězce a následně třeba vypsat do konzole. Tento způsob je představen v kódu 3.

Kód 3 *Příklad vkládání proměnných do řetězce a výpis do konzole v jazyce JavaFX Script*

```
def x: Integer = 1984;
def s: String = "x ma hodnotu {x}";
println(s);  // vypise "x ma hodnotu 1984"
```

2.2.2 Řídící struktury

Programovací jazyk JavaFX Script umožňuje použití běžných řídících struktur jako v jazyku Java. Existuje zde podmíněný příkaz `if` s alternativním větvením `else`. Neexistuje však větvení programu pomocí `switch` a `case`. Z cyklů jsou podporovány příkazy `while` a `for`, nikoliv však `do-while`. (Zdroj [5])

Kód 4 *Ukázka for cyklu v JavaFX Scriptu*

```
/* pocet pruchodu cyklem je zadan pomoci sekvence */
for (j in [0..4]) {
    delej(j);
}

/* behem jednotlivych pruchodu se v promenne den */
/* postupne vystridaji vsechny objekty ze sekvence dnyVTydnu */
for (den in dnyVTydnu) {
    zobrazNazevDne(den);
}
```

Při druhém použití `for` cyklu v příkladu v kódu 4 nepotřebujeme znát velikost sekvence.

2.2.3 Data binding

Zajímavou vlastností programovacího jazyka JavaFX Script je schopnost tzv. data bindingu. Jedná se v podstatě o svázání dvou proměnných. V kódu 5 je vidět příklad svázání dvou proměnných, kdy je proměnná `x` inicializována hodnotou 25 a proměnná `y` je díky svázání deklarována jako „věčný“ trojnásobek hodnoty proměnné `x`. (Zdroj [1], str. 66)

Kód 5 *Příklad svázání dvou proměnných pomocí data bindingu v JavaFX Scriptu*

```
var x = 25;
var y = bind 3 * x;
/* Nyni je hodnota promenne y rovna 75 */
x = 30;
/* Nyni je hodnota promenne y rovna 90 */
```

2.2.4 Triggers

Triggers neboli spouštěče jsou jakousi obdobou data bindingu. Spočívají v definování určitého bloku kódu programu, který se má provést při změně obsahu konkrétní proměnné. Spouštěče se definují pomocí klíčových slov `on replace`, po kterých následuje kód, který se má při změně obsahu proměnné provést. Dále je také možné zjistit, z jaké na jakou hodnotu se proměnná změnila. Je třeba ještě podotknout, že poprvé je kód spouštěče vyvolán přímo v místě, kde je proměnná se spouštěčem deklarována. Pro názornost je uveden příklad v podobě kódu 6. (Zdroj [24])

Kód 6 *Příklad užití spouštěče v JavaFX Scriptu*

```
var x: Integer = 20 on replace {
      println("nastala zmena obsahu promenne");
}
x = 35; //provede se kod s vypisem "nastala zmena..."

var y: Integer = 84 on replace yPred=yPotom {
      println("y pred = {yPred} potom = {yPotom}");
}
y = 19; //provede se kod s vypisem "y pred = 84 potom = 19"
```

2.2.5 Sekvence

Zatímco jazyk Java používá pro ukládání množiny objektů pole (Zdroj [31]), jazyk JavaFX Script představuje tzv. sekvence. Ty lze vytvářet mnoha různými způsoby, jak ukazuje kód 7.

Kód 7 *Příklad inicializace sekvencí v JavaFX Scriptu*

```
var sekvence1: Integer[] = [0..5]; //vznikne sekvence cisel 0,1,2,3,4,5
var sekvence2: Integer[] = [0..<5]; //vznikne sekvence cisel 0,1,2,3,4
var sekvence3: Integer[] = [0..9 step 3]; //vznikne sekvence cisel 0,3,6,9
var sekvence4: Integer[] = [[0..2], 5]; //vznikne sekvence cisel 0,1,2,5
```

Jazyk JavaFX Script má k dispozici několik příkazů pro manipulaci se sekvencemi. Pro vkládání objektů do sekvencí se používá příkaz `insert`. Navíc je možné specifikovat, kam přesně chceme objekt vložit. Příkaz `insert` tak doplňuje některé z klíčových slov `into`, `after` nebo `before`. Pro mazání objektů ze sekvencí se používá příkaz `delete`. Převrácení pořadí jednotlivých objektů v sekvenci provádí příkaz `reverse`. (Zdroj [26])

Kód 8 *Příklad použití příkazů insert, delete a reverse na sekvence v JavaFX Scriptu*

```
var dny: String[] = ["utery","ctvrtek"];
insert "patek" into dny;  // "patek" se vlozi NAKONEC sekvence
insert "streda" after dny[0];  // "streda" se vlozi ZA "utery"
insert "pondeli" before dny[0];  // "pondeli" se vlozi PRED "utery"
delete dny[1];  // odstrani ze sekvence "utery"
dny = reverse dny;  // obraceni: "patek", "ctvrtek", "streda", "pondeli"
delete dny;  // odstraneni cele sekvence
```

2.2.6 Třídy

Při vytváření instancí tříd v jazyku JavaFX Script je používána deklarativní syntaxe. Tento způsob výrazně napomáhá názornosti zdrojového kódu. Samotný zápis deklarace třídy je shodný s jazykem Java. Objevují se zde však odlišnosti, jako je například absence konstruktoru třídy.

Místo konstruktoru třídy lze definovat bloky `init` a `postinit`. Blok `init` je prováděn při inicializaci třídy a blok `postinit` až poté.

Pořadí je tedy obecně takové, že nejprve je proveden blok `init` rodičovské třídy, poté blok `init` potomka, následně blok `postinit` rodičovské třídy a nakonec blok `postinit` potomka. (Zdroj [1], str. 40)

Abychom zjistili, zda byla konkrétní proměnná inicializována, můžeme využít funkce `isInitialized()`, která vrací `true` pro inicializovanou proměnnou. (Zdroj [1], str. 62)

Kód 9 *Příklad deklarace třídy a vytvoření instance třídy v JavaFX Scriptu*

```
/* Trida.fx */
public class Trida {
    public-init var cislo: Integer;  //volny pristup jen pri instanciaci
    public-init var doba: Duration;
    public-read var nazev: String;
    var k: Number;
    init {
        nazev = "moje trida";
    }
}
...
/* JinySoubor.fx */
...
/* vytvoreni objektu z tridy Trida, inicializace promennych */
var objekt: Trida = Trida {
    cislo: 1984
    doba: 3m
}
println(objekt.nazev);  // vypise "moje trida"
objekt.nazev = "abcd";  // !!! toto nelze !!!
println(objekt.cislo);  // !!! toto nelze !!!
objekt.k = 15.7;        // !!! toto nelze !!!
```

Na konci kódu 9 je ukázka nepovolených přiřazení hodnot do proměnných `objekt.nazev` a `objekt.k` a také čtení, zvenku „neviditelné“, proměnné `objekt.cislo`.

Jazyk JavaFX Script rozlišuje několik identifikátorů přístupových práv k proměnným tříd. Přehled jednotlivých identifikátorů přístupu ukazuje tabulka 2.

Identifikátor	Popis
`public-init`	Umožňuje zápis do proměnné pouze při inicializaci.
`public-read`	Proměnná je pouze pro čtení. Zápis lze provést jen metodami dané třídy.
`public`	Přístup k proměnné není omezen.
`protected`	Proměnné jsou dostupné pouze v dané třídě a odvozených třídách.
`package`	Určuje viditelnost proměnné v rámci jednoho balíku.
bez identifikátoru	Přístup k proměnné je možný pouze v dané třídě.

Tabulka 2: Přehled datových typů jazyka JavaFX Script, (Zdroj [27])

2.2.6.1 Dědičnost

Pro deklarování vztahu potomek – rodič se používá standardní klíčové slovo `extends`. Třídy jazyka JavaFX Script mohou být potomky pouze jedné třídy Java nebo JavaFX. V případě použití třídy Java je nutné, aby konstruktor neměl žádný parametr. Omezení dědění jedné třídy lze obejít faktem, že třídy v jazyce JavaFX Script mohou být potomky několika tzv. `mixin` tříd a nebo mohou implementovat několik Java rozhraní. (Zdroj [1], str. 36)

Kód 10 *Příklad použití mixin třídy z aplikace SpaceGame, která je rozebírána v této práci*

```
/* Actor.fx */
/* Actor je rodicovska trida pro nektere tridy v projektu SpaceGame */
public mixin class Actor {
    public-read var posX: Number;
    public-read var posY: Number;
    public abstract function run() : Void;
    ...
```

```
    ...
    public function getX() : Number {return posX;}
    ...
}
/* SpaceShip.fx */
/* Trida SpaceShip je potomkem abstraktni tridy CustomNode z JavaFX API */
/* a zaroven potomkem vytvorene mixin tridy Actor */
public class SpaceShip extends CustomNode, Actor {
    /* je treba implementovat metodu run() z tridy Actor */
    public override function run(): Void {
        ...
    }
    /* implementace metody run() z tridy CustomNode */
    /* neni to nutne pro bezchybny preklad, ale chceme to */
    public override function create(): Group {
        ...
    }
}
```

2.2.7 Princip tvorby uživatelského rozhraní

Deklarativní přístup jazyka JavaFX při vytváření instancí tříd přináší výhodu v tom, že lze ze struktury zdrojového kódu, která je takovými instancemi tříd tvořena, snadněji vyčíst propojenost objektů a přeneseně tedy i vyčíst chování.

Každá aplikace JavaFX by měla vytvářet instanci třídy `Stage`. Třída `Stage` dále obsahuje proměnnou `scene`, která je třídy `Scene`. Ta představuje samotný obsah okna. Pro názornost si uvedeme ukázkový kód, který vytvoří aplikaci s prázdným oknem (viz kód 11).

Kód 11 *Příklad jednoduché aplikace JavaFX s prázdným oknem*

```
Stage {
    title: "Titulek okna"
    scene: Scene {
        width: 640  height: 480
        content: []
    }
}
```

Obsah okna pak vytvářejí objekty, které jsou umísťovány do sekvence `content` třídy `Scene`. Do této sekvence můžeme vkládat všechny objekty jejichž třídy jsou potomky standardní JavaFX třídy `Node`, resp. `CustomNode` .

Nyní je třeba ještě ukázat příklad, jak do obsahu okna vložit nějaký objekt. Kód 12 ukazuje vložení objektu tlačítka do objektu centrovacího kontejneru třídy `Stack`.

Kód 12 *Stromová struktura vznikající z jednotlivých objektů GUI*

```
Stage {
    title: "Titulek okna"
    scene: Scene {
        width: 640
        height: 480
        content: [
            Stack {
                content: [
                    Button {
                       text: "Klikni"
                    }
                ]
            }
        ]
    }
}
```

Při pohledu na kód 12 si nyní snadno všimneme, že nám takovýmto vkládáním prvků vzniká stromová struktura. A přesně to je hlavním principem návrhu grafické stránky aplikací JavaFX.

2.2.7.1 Použití CSS stylů

Pro definování vzhledu jednotlivých grafických prvků aplikací JavaFX je možné využít kaskádových stylů. Základní princip i syntaxe jsou stejné jako u CSS stylů používaných při tvorbě webových stránek. Existují zde však i určité odlišnosti.

Jednou z nich je například to, že nelze pomocí CSS stylů definovat samotnou velikost grafického prvku, tu je třeba definovat v kódu aplikace. (Zdroj [19])

Dalším rozdílem jsou i názvy jednotlivých vlastností prvků. Například pro definování velikosti písma určitého prvku se používá vlastnost `-fx-font-size`, zatímco v běžných webových kaskádových stylech má stejná vlastnost název `font-size` (Zdroj [7]). Tímto způsobem byly názvy odlišeny až s příchodem verze 1.3 (Zdroj [15]).

Pro připojení souboru se styly se název souboru uvede při inicializaci třídy `Scene` do proměnné `stylesheets`. Kompletní řešení propojení CSS stylů z konkrétním grafickým objektem je ukázáno v kódu 13. Kompletní přehled CSS stylů v JavaFX uvádí zdroj [23].

Kód 13 *Příklad použití CSS stylů v JavaFX Scriptu*

```
/* Main.fx */
import javafx.scene.*;
import javafx.stage.*;
import javafx.scene.control.*;

Stage {
    scene: Scene {
        stylesheets: "styly.css";
        content: Button {text: "Text tlacitka"}
    }
}

/* styly.css */
.button {
    -fx-text-fill: white;
}

.button:hover {
    -fx-text-fill: red;
}
```

2.2.7.2 Cílový spouštěcí profil aplikace

Jak již bylo zmíněno, aplikace JavaFX, které využívají API společné pro více rozdílných cílových zařízení (platforem), mohou být teoreticky spouštěny právě na všech těchto zařízeních bez změny kódu aplikace. Občas je tedy nutné v programu zjistit, ve kterém spouštěcím profilu aplikace vlastně běží. Získání takové informace může být motivováno například potřebou přizpůsobit část GUI mobilnímu zařízení.

Právě tuto informaci lze zjistit z obsahu proměnné `__PROFILE__`, která obsahuje řetězec identifikující spouštěcí profil. Popis jednotlivých možných hodnot proměnné `__PROFILE__` je ukázán v tabulce 3. Tyto hodnoty jsem zjistil pomocí jednoduchého experimentu, při kterém jsem zjišťoval obsah proměnné `__PROFILE__` během spouštění testovací aplikace na všech spouštěcích variantách v prostředí NetBeans.

`__PROFILE__`	**Popis**
`"desktop"`	Aplikace běží na klasickém desktopu.
`"browser"`	Aplikace běží v okně webového prohlížeče.
`"mobile"`	Aplikace běží na mobilním zařízení.
`"tv"`	Aplikace běží na televizním zařízení

Tabulka 3: Možné hodnoty proměnné __PROFILE__

3 Aplikace

Pomocí principů a postupů uvedených v předchozí kapitole jsem pro tuto práci vytvořil čtyři aplikace, které by měly čtenáři přiblížit konkrétní postupy na praktické ukázce. Následující část práce se tedy věnuje popisu principů fungování jednotlivých aplikací a je prokládána souvisejícími částmi zdrojových kódů. Doporučuji čtení textu kombinovat se zkoušením jednotlivých aplikací.

3.1 Countdown

3.1.1 Návrh tématu a činnosti aplikace

K vytvoření aplikace Countdown mě inspirovala aplikace pro odpočet času na mém mobilním telefonu, který je vybaven operačním systémem Android (Zdroj [4]). Takovou aplikaci však lze považovat za užitečnou i na dalších platformách a zařízeních. Aplikace Countdown by tedy měla být napsána tak, aby ji bylo možné bez jediné změny kódu spustit na emulátoru jak pro mobilní zařízení, tak i pro televizní zařízení.

Jako cíl jsem tedy stanovil vytvoření aplikace, která bude uživateli umožňovat nastavit koncový čas a datum, do kterého se bude čas odpočítávat. Při samotném odpočítávání budou zobrazeny zbývající dny, hodiny, minuty, vteřiny i desetiny vteřiny.

Jedním z hlavních přínosů aplikace Countdown pro tuto práci by mělo být názorné osvětlení principu umísťování jednotlivých ovládacích prvků pomocí polohovacích kontejnerů a použití kaskádových stylů pro definování vzhledu jednotlivých grafických prvků.

3.1.2 Návrh uživatelského rozhraní

Uživatelské rozhraní aplikace Countdown by mělo umožňovat zadávat čas a datum pouhým výběrem z připravených hodnot místo zdlouhavého psaní do jednotlivých políček. Takový přístup bude zároveň vhodný, jestliže chceme mít aplikaci snadno přenositelnou a tedy i snadno ovladatelnou na dalších zařízeních, kde je třeba psaní jednotlivých číslic komplikované. Při spuštění aplikace by se měl navíc výběr cílového času a datumu nastavit na hodnoty aktuálního času. Tím bude ovládání ještě více usnadněno a odpočet času půjde nastavit třeba i změnou jediné části času či datumu. Po výběru cílového času a data pak nastavení zmizí a objeví se běžící odpočet. Ten bude možné kdykoliv přerušit stisknutím tlačítka „Stop“. Dále by mělo být možné z pohledu běžícího času stisknout tlačítko „Settings“, které uživatele dovede zpět do nastavení nového odpočtu času.

3.1.3 Implementace

Celou aplikaci Countdown jsem rozdělil do sedmi tříd. Toto rozdělení ukazuje tabulka 4.

Třída/soubor	Popis
`Background`	Třída představující barevné pozadí aplikace. Zajišťuje změnu pozadí při přechodu z části nastavení do režimu zobrazení displeje s běžícím odpočítáváním času.
`Display`	Představuje okno displeje časovače. Třída je potomkem třídy `Screen`.
`Screen`	Rodičovská třída pro třídy `Display` a `Settings`.
`Settings`	Představuje okno nastavení časovače. Třída je potomkem třídy `Screen`.
`Timer`	Třída představuje časovač, implementuje metodu pro výpočet rozdílu časů.
`TimerData`	Představuje jednotlivé hodnoty rozdílu časů.
`View`	Hlavní třída, která řídí animování zobrazení nastavení nebo displeje.

Tabulka 4: Přehled tříd aplikace Countdown a jejich popis

3.1.3.1 Uživatelské rozhraní

Použití kaskádových stylů

V aplikaci Countdown jsem se rozhodl vyzkoušet možnost definování vzhledu objektů ve scéně pomocí kaskádových stylů. Vytvořil jsem čtyři CSS soubory, každý pro jiný spouštěcí profil:

- `styleSheet-browser.css` – Pro běh aplikace v prohlížeči
- `styleSheet-desktop.css` – Pro běh standardní desktopové aplikace
- `styleSheet-mobile.css` – Pro běh v emulátoru mobilních zařízení
- `styleSheet-tv.css` – Pro běh aplikace v emulátoru TV.

V souboru `Main.fx` jsem pomocí proměnné `__PROFILE__` definoval jméno CSS souboru, jak ukazuje kód 14. Pro úplnost je třeba ještě uvést, že proměnná `__DIR__` v kódu 14 představuje umístění adresáře aplikace a běžně se pro takové účely používá.

Kód 14 *Definování zdroje pro CSS styly podle aktuálního spouštěcího profilu*

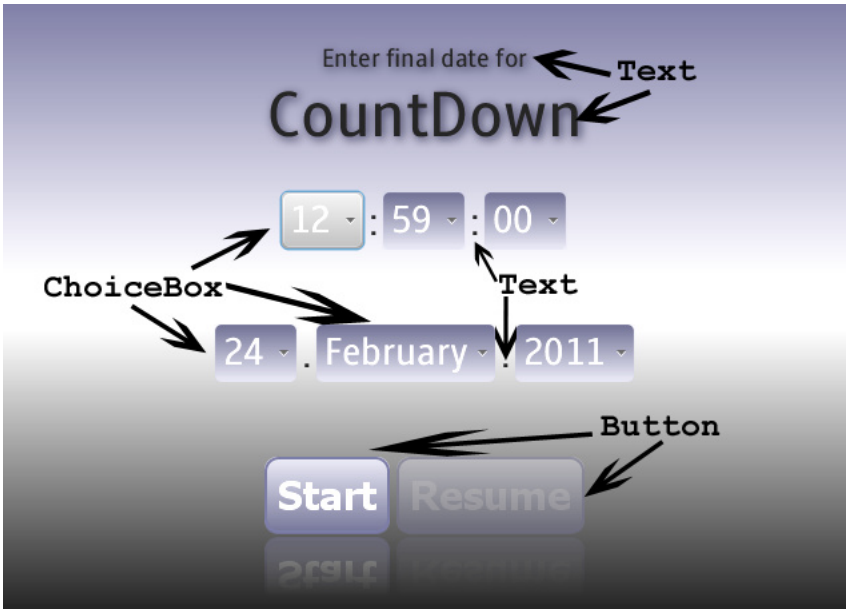
```
/* Main.fx */
...
Stage {
    title: "Countdown"
    width: 640
    height: 480
    scene: Scene {
        stylesheets: "{__DIR__}styleSheet-{__PROFILE__}.css"
        content: [
            View{}
        ]
    }
}
...
```

Samotné CSS styly jsou v aplikaci použity například pro definování barvy a velikosti písma jednotlivých prvků. Dále pak pro definování barevných gradientů na pozadí prvků. Vzhledem k již uvedeným omezeným možnostem kaskádových stylů JavaFX 1.3 oproti webovým CSS stylům, je jejich použití spíše zajímavostí než nutností.

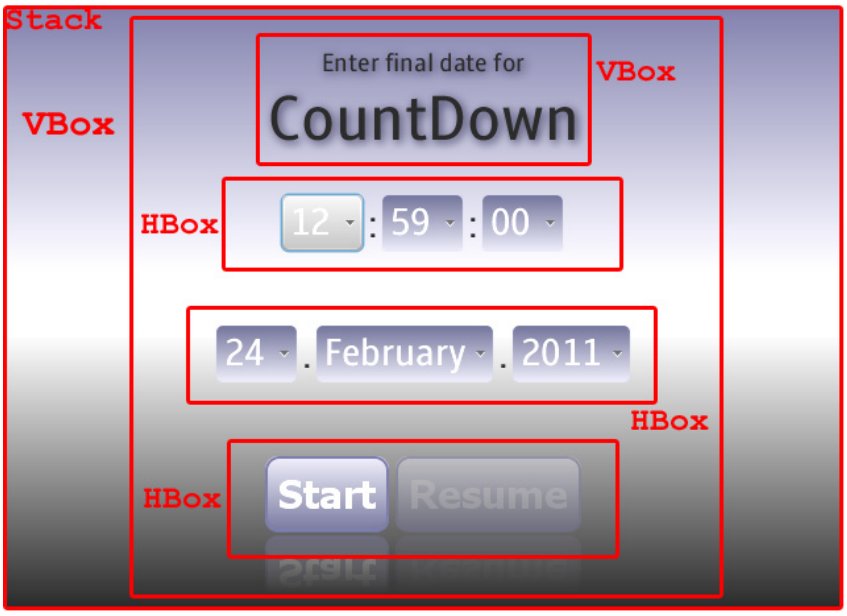
Vraťme se ale ke konkrétnímu řešení vizuální stránky aplikace Countdown. Grafické uživatelské rozhraní aplikace jsem rozdělil na dvě části. Konkrétně se jedná o nastavení časovače a displej časovače. Tyto části jsou odděleny animovaným přechodem.

Nastavení časovače

Prostředí nastavení časovače je vytvořeno ve třídě `Settings`. Tato třída je potomkem standardní třídy `CustomNode` a nově vytvořené třídy `Screen`. Vzhled nastavení časovače je deklarován v metodě `create()`, která skládá jednotlivé prvky do sekvence. Jedná se o strukturu kontejnerů pro pozicování, do kterých jsou vloženy objekty tříd `Text`, `ChoiceBox` a `Button` jak je ukázáno na obrázku 3.1. Přehled použitých kontejnerů je zobrazen na obrázku 3.2.



Obrázek 3.1: Použití objektů tříd ChoiceBox, Button a Text pro aplikaci Countdown

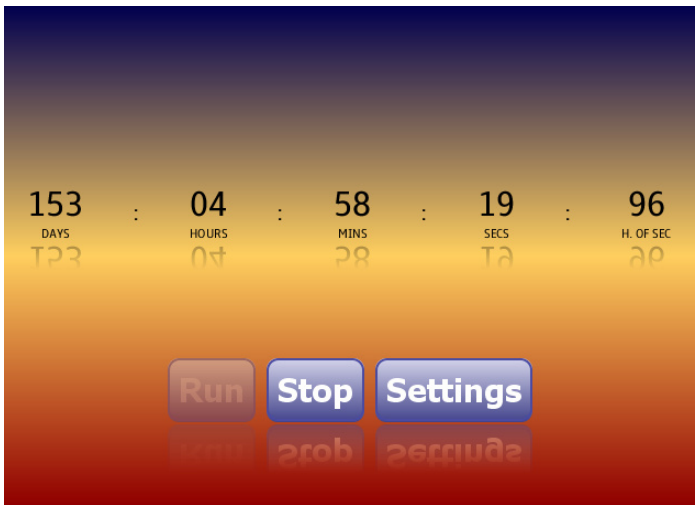


Obrázek 3.2: Umísťování objektů pomocí kontejnerů Stack, VBox a HBox ve třídě Settings

Z obrázku 3.2 vyplývá, že kontejner `Stack` byl použit pro vycentrování svého obsahu (vnější `VBox`). Kontejner `VBox` jsem použil pro vkládání prvků do vertikální posloupnosti a kontejner `HBox` pro vkládání prvků do horizontální posloupnosti.

Displej časovače

Pro sledování samotného běžícího času bylo třeba vytvořit třídu, která bude toto umožňovat. Třída má název Display a vytváří grafickou reprezentaci běžícího času a ovládacích tlačítek jak je ukázáno na obrázku 3.3.



Obrázek 3.3: Displej odpočítávání času

Pro umístění jednotlivých prvků byla opět použita kombinace tříd `Stack`, `HBox` a `VBox`. Textový obsah každého objektu s číselnými údaji je svázán pomocí jazykové konstrukce `bind` s příslušnou proměnnou, která je definována ve třídě `Display`.

Kód 15 *Svázání obsahu textového pole s proměnnou strDays ve třídě Display*

```
/* Display.fx */
var strDays : String = "00";
public override function create() : Node {
    Text {
        content: bind strDays
        styleClass: "countdownNumber"
        effect: bind reflectionFX
}   }
```

Běh odpočítávání je realizován objektem třídy `Timeline`, který je definován uvnitř třídy `Display`. Ten zajišťuje běh nekonečné smyčky o zadané frekvenci. Při každém průchodu je volána metoda `updateDisplayValues()`, která provede aktualizaci údajů na displeji, viz kód 16.

Kód 16 *Definování objektu timerLoop pro běh nekonečné smyčky a část updateDisplayValues()*

```
/* Display.fx */
...
def timerLoop : Timeline = Timeline {
    repeatCount: Timeline.INDEFINITE
    keyFrames: [
        KeyFrame{
            time: 50ms
            action: function() {
                updateDisplayValues();
            }
        }
    ]
}
...
function updateDisplayValues() : Void {
    var dt: Date = Date{};
    timerData = timer.getTimeDifference(dt);
    strDays = getFineStringValue(timerData.getDays(),0);
    strHours = getFineStringValue(timerData.getHours(),1);
    ...
```

3.1.3.2 Výběr pohledu – z nastavení do časovače a zpět

Ve třídě `View` jsem definoval metodu `changeView()`, která přepíná pohled z nastavování odpočtu času do běhu odpočtu času a zpět. V této metodě jsou spouštěny příslušné animované přechody mezi těmito stavy. Navíc je zde řízena změna pozadí.

Metoda `changeView()` je volána ze tříd `Display` a `Settings` jako reakce na stisk příslušného tlačítka buďto při nastavení údajů pro odpočet nebo v režimu běhu odpočtu času. Tělo metody `changeView()` je ukázáno v kódu 17.

Kód 17 *Metoda changeView() třídy View*

```
/* View.fx */
...
public function changeView(state: Integer) : Void {
    if (state == 0) {
        showSettings.playFromStart();
        background.playFadeToSettingsBackground();
    } else {
        hideSettings.playFromStart();
        background.playFadeToDisplayBackground();
    }
}
...
```

Dále jsou ve třídě `View` definovány objekty třídy `SequentialTransition`, které představují posloupnost animovaných přechodů mezi nastavením a displejem. Kód 18 ukazuje definování přechodů pro „odjezd“ nastavení odpočtu času (`hideSettings`) a plynulé objevení displeje běžícího odpočtu času. V kódu 18 se poprvé setkáváme s třídou Interpolator, pomocí které se definuje dynamika animačního cyklu. Zjednodušeně lze tvrdit, že třída `Interpolator` nabízí několik konstant pro volbu dynamiky průběhu dané animace. Bližší informace lze získat v dokumentaci samotné třídy `Interpolator` v JavaFX API (Zdroj [21]).

Kód 18 *Přechod hideSettings se skládá z posloupnosti objektů tříd Translate a Fade Transition*

```
/* View.fx */
...
def transHideSettings = TranslateTransition {
    duration: 600ms
    node: bind screenSettings
    fromX: 0 toX: 0
    fromY: 0 toY: bind -scene.height*2
    repeatCount:1 autoReverse: false
    interpolator: Interpolator.EASEIN
}
...
def transShowDisplay = FadeTransition {
```

```
    ...
    duration: 1s
    node: bind screenDisplay
    fromValue: 0.0 toValue: 1.0
    repeatCount:1 autoReverse: false
}
...
def hideSettings = SequentialTransition {
    content: [transHideSettings,  transShowDisplay]
}
...
```

3.1.3.3 Výpočet aktuálního zbývajícího času

Pro výpočet rozdílu cílového a aktuálního času byla vytvořena třída `Timer`, která obsahuje metodu `getTimeDifference()`. Tato metoda je volána ze třídy `Display` pro pravidelnou aktualizaci údajů na displeji.

Kód 19 *Část těla metody getTimeDifference() třídy Timer*

```
/* Timer.fx */
...
var nTotalDiff : Long = finalDate.getTime() - earlierDate.getTime();

data.setDays(Math.floor(nTotalDiff/1000/60/60/24));
nTotalDiff -= data.getDays()*1000*60*60*24;
data.setHours(Math.floor(nTotalDiff/1000/60/60));
nTotalDiff -= data.getHours()*1000*60*60;
data.setMinutes(Math.floor(nTotalDiff/1000/60));
nTotalDiff -= data.getMinutes()*1000*60;
data.setSeconds(Math.floor(nTotalDiff/1000));
nTotalDiff -= data.getSeconds()*1000;
data.setHOfSec(Math.floor(nTotalDiff/10));

return data;
...
```

3.1.4 Přenositelnost aplikace Countdown

Jak jsem zmínil již v návrhu této aplikace, k vytvoření programu Countdown jsem byl inspirován podobnou aplikací z mobilního prostředí. Tuto variantu jsem tedy vyzkoušel a spustil aplikaci Countdown na emulátoru mobilního telefonu, který je součástí vývojových nástrojů JavaFX v prostředí NetBeans (Zdroj [20]). Spuštění je zaznamenáno na obrázku 3.4. Pro úplnost bych ještě dodal, že jsem provedl test i na emulátoru televizního zařízení, kde aplikace Countdown běžela také bezchybně.

Obrázek 3.4: Aplikace Countdown na emulátoru mobilních zařízení

3.1.5 Závěr k aplikaci Countdown

Countdown je ukázkou typu aplikace, jaké jsou běžně součástí standardní aplikační výbavy na mobilních zařízeních (Zdroj [4]). Díky JavaFX lze vytvořit takovou aplikaci, která bude, bez jediné změny v kódu, přenositelná i na další zařízení. Při tvorbě aplikace jsem navíc využil možnost částečně ovlivnit vzhled grafických prvků pomocí kaskádových stylů. Ty byly využity při definování vzhledu pozadí a také barvy a velikosti textů v ovládacích prvcích (viz obrázek 3.5). Schopnosti CSS stylů v JavaFX jsou však zatím značně omezené (viz kapitola 2.2.7).

Obrázek 3.5: Aplikace Countdown, nastavení odpočtu

3.2 Currency Converter

3.2.1 Návrh tématu a činnosti aplikace

Standardní API JavaFX nabízí třídu pro parsování souborů ve formátech XML a JSON. Tato třída se nazývá `PullParser`. Zároveň je k dispozici i třída `HttpRequest`, která zajišťuje provádění http požadavků. Právě na součinnosti těchto dvou tříd jsem se rozhodl postavit další JavaFX aplikaci. (Zdroj [21])

Na Internetu jsem hledal volně přístupné XML soubory, které by poskytovaly nějaké aktuální informace, které bych v aplikaci mohl využít. Nakonec jsem se rozhodl použít XML soubor s aktuálními kurzy měn, který je umístěn na adrese: http://rh-weby.cz/kurzy-men/kurzy.xml a jeho obsah je pravidelně aktualizován. Jako cíl jsem tedy stanovil vytvoření aplikace pro převod měn podle aktuálních kurzů.

3.2.2 Návrh uživatelského rozhraní

Při startu aplikace se zobrazí informace o připojování či nahrávání dat. Poté se zobrazí formulář s prvky pro výběr vstupní a výstupní měny. Formulář dále musí obsahovat vstupní textové pole pro zadávání částky určené pro přepočet a také statické výstupní místo pro text, kde bude zobrazen výsledek přepočtu. Nesmí chybět ani tlačítko, kterým bude přepočet vyvolán.

3.2.3 Implementace

Kód aplikace Currency Converter jsem rozčlenil do pěti souborů (viz tabulka 5). Hlavní `Stage` aplikace je jako vždy obsažena v souboru `Main.fx`. Do obsahu její scény je vložena instance třídy `View`.

Třída/soubor	Popis
`Config`	Soubor `Config.fx` obsahuje pouze definice veřejných proměnných určujících výslednou šířku a výšku okna aplikace.
`Converter`	Soubor `Converter.fx` obsahuje definici veřejné funkce `convert()`, která vrací výsledek převodu dané částky z jedné měny na druhou.
`Currency`	Třída reprezentující měnu a její kurz ke koruně.
`Data`	Třída představuje tabulku dat. Zajišťující provedení http požadavku, parsování XML souboru.
`View`	Třída, která definuje vzhled uživatelského rozhraní a reaguje na uživatelský vstup.

Tabulka 5: Popis jednotlivých tříd aplikace Currency Converter

3.2.3.1 Získání dat

Pro nahrávání a parsování dat jsem vytvořil třídu `Data`, která pomocí metody `load()` získá data z internetového XML souboru. V metodě `load()` je nejprve vytvořen objekt třídy `PullParser`, který je inicializován pro práci s daty ve formátu XML. Součástí této inicializace je i definování reakce na událost `PullParseru`, viz kód 20.

Kód 20 *Příprava objektu třídy PullParser*

```
/* Data.fx */
...
var parser = PullParser {
    documentType: PullParser.XML;
    input: null
    onEvent: function(event: Event) {
    if (event.type == PullParser.END_ELEMENT) {
        if (event.qname.name == "title" and event.level == 3) {
            currStr = event.text;
        }
        ...
```

Dále je třeba vytvořit objekt třídy `HttpRequest`, který provede http požadavek na přečtení obsahu internetové adresy. Při vytváření objektu je třeba implementovat reakce na události třídy `HttpRequest`, především `onInput` a `onException`. Obslužná metoda události `onInput` má jeden parametr typu `java.io.InputStream`, kterým se získává proud čtených dat. Toto místo je příkladem skutečnosti, že JavaFX může zcela normálně využít třídy Java. Proud `InputStream` se dále předává objektu `parser`. Je vhodné implementovat i metodu pro událost `onException`, tedy reakci na výjimku. Konstrukce objektu třídy `HttpRequest` je ukázána ve zjednodušené podobě v kódu 21.

Kód 21 *Příprava objektu třídy HttpRequest, zjednodušená verze oproti třídě Data*

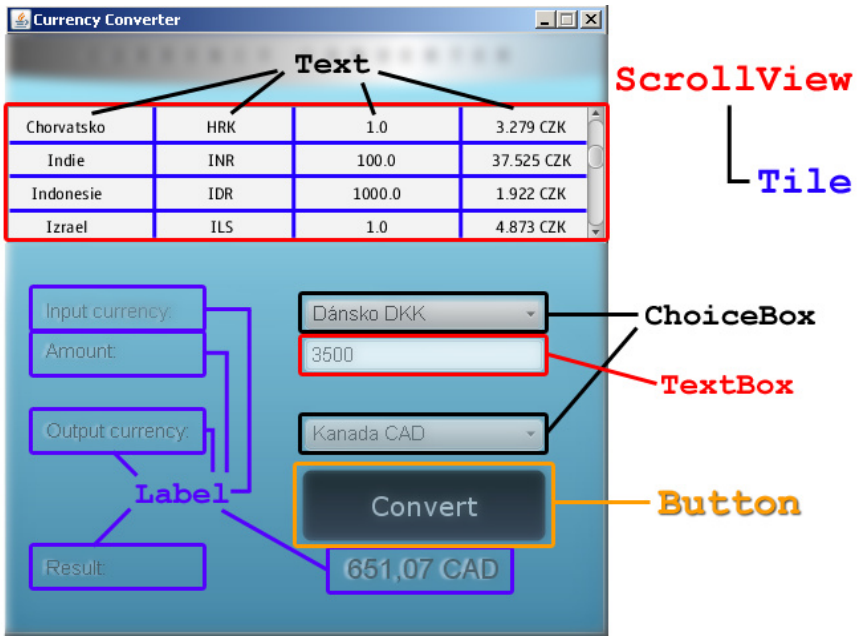
```
...
var request: HttpRequest = HttpRequest {
    location: URL_XML
    method: HttpRequest.GET
    onException: function(ex: Exception) {
        if (ex instanceof java.net.UnknownHostException) {
            println("Host was not found.");
        }
    }
    onInput: function(input: java.io.InputStream) {
        try {
            println("HttpRequest: loading data...");
            parser.input = input;
```

```
        parser.parse();
        input.close();
        println("HttpRequest: data loaded");
    }
    catch (e : Exception) {
        e.printStackTrace();
    }
...
```

Objekt `parser` během své činnosti z každého záznamu měny v XML vytvoří pomocí metody `createCurrencyFromXML()` objekt třídy `Currency`. Ten pak přidá do kolekce `dataTable`, která je součástí třídy `Data`. Kolekce `dataTable` je spojena se spouštěčem, který při každé změně kolekce volá metodu `updateChoiceBoxData()`, která naplní seznamy měn v uživatelském rozhraní.

3.2.3.2 Uživatelské rozhraní

Grafické uživatelské rozhraní aplikace Currency Converter je tvořeno objekty tříd `Text`, `ChoiceBox` a `ScrollView`. Grafický obsah objektu třídy `ScrollView` je dále zarovnán pomocí kontejneru `Tile`, viz obrázek 3.6.



Obrázek 3.6: Přehled GUI prvků v uživatelském rozhraní aplikace Currency Converter

3.2.3.3 Převod měny

Samotný převod částky ve vstupní měně na částku v měně výstupní je proveden po stisknutí tlačítka „Convert“. Ovladač této události je obsažen ve třídě `View`. Po kliknutí se přečte obsah vstupního textového pole a převede se z formátu textu na číslo. Dále jsou získány konkrétní měny, které jsou pro převod vybrány. Poté se již zavolá funkce `convert()` definovaná v souboru `Converter.fx`. Při úspěchu se výsledek uloží do proměnné `newValue`.

Kód 22 *Reakci na kliknutí tlačítka „Convert“ v aplikaci Currency Converter*

```
/* View.fx */
...
Button {
    ...
    text: "Convert"
    action: function(){
        var inputAmount : Number;
        try {
            inputAmount = java.lang.Float.parseFloat(valueInput.text);
        }
        ...
        var inputCurrIndex = currencyChoiceBox.selectedIndex;
        var outputCurrIndex = currencyChoiceBoxDest.selectedIndex;
        var curr : Currency = data.getData()[inputCurrIndex];
        var finalCurr : Currency = data.getData()[outputCurrIndex];
        var result = Converter.convert(inputAmount, curr, finalCurr);
        if (result != -1) {
            var currStr : String = finalCurr.getCurrShort();
            newValue = "{%.2f result} {currStr}";
            blurFXtimeline.playFromStart();
        ...
```

Poté se spustí efekt rozmáznutí a zaostření výsledku. Až teprve při maximální rozmáznutí textu výsledku dojde k zapsání nově vypočítané hodnoty. Efekt `GaussianBlur`, který je aplikován na objekt `valueOutput`, což je objekt třídy `Text`, který obsahuje výsledek převodu. Parametru efektu `radius` je svázán s proměnnou `blurRadius`, jejíž

hodnota je řízena objektem `blurFXtimeline`, což je objekt třídy `Timeline`. Tato konstrukce je pro názornost ukázána v kódu 23.

Kód 23 *Konstrukce efektu rozmáznutí a zaostření výsledku při stisku tlačítka „Convert“*

```
/* View.fx */
...
def blurFXtimeline = Timeline {
    repeatCount: 2  autoReverse: true
    keyFrames: [
        KeyFrame {
            time: 500ms
            values: [blurRadius => 10 tween Interpolator.LINEAR]
            action: function() : Void {valueOutput.text = newValue; }
        }
    ]
}
Group {
    content: [valueOutput]
    effect: GaussianBlur {radius: bind blurRadius}
}
...
```

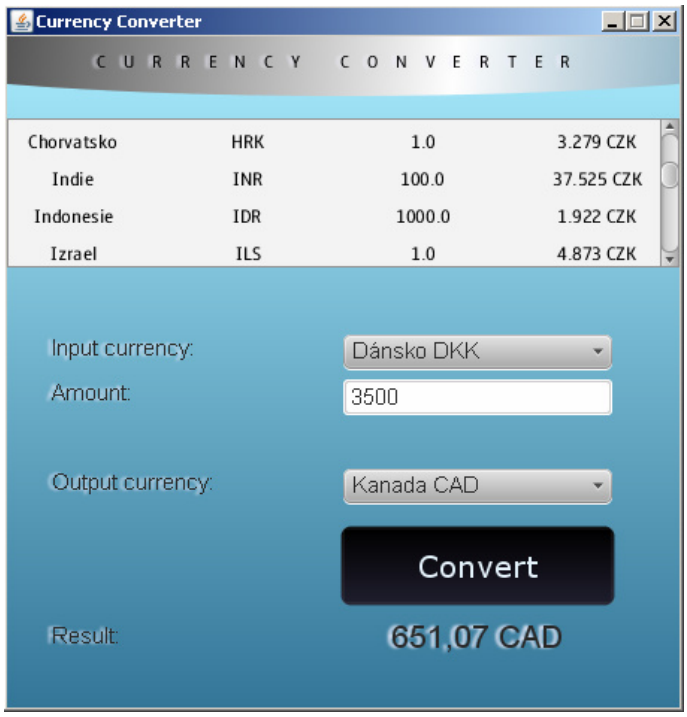
3.2.4 Přenositelnost aplikace Currency Converter

Aplikaci Currency Converter je možné spustit jako klasickou desktopovou i webovou aplikaci. Nelze ji však přeložit pro mobilní zařízení, protože pro tento cílový spouštěcí profil neexistují v API některé třídy a metody. Například třída `String` postrádá metodu `replaceAll()` pro práci s regulárními výrazy, které jsou použity při rozdělování jednotlivých dat z tagů v XML souboru. Dále nejsou, při kompilaci aplikace pro mobilní profil, k dispozici třídy `java.io.FileInputStream` a `java.io.FileNotFoundException`.

3.2.5 Závěr k aplikaci Currency Converter

Aplikace Currency Converter ukazuje použití standardních JavaFX tříd `PullParser` a `HttpRequest` pro parsování souboru ve formátu XML na dané internetové adrese. Dále

pak ukazuje konkrétní použití získaných dat. V aplikaci je využita i jednoduchost JavaFX při vytvoření uživatelského rozhraní s drobnými efekty. Currency Converter dále ukazuje, že ne vždy je snadné napsat aplikaci, která by byla přenositelná na více typů klientských zařízení.



Obrázek 3.7: Aplikace Currency Converter

3.3 Reklamní banner

3.3.1 Návrh tématu a činnosti aplikace

V dnešní době (rok 2010) je možné vidět na Internetu reklamní bannery připravené především ve formátu Adobe Flash (Zdroj [1]). JavaFX je stejně jako Flash platforma pro běh RIA aplikací (Zdroj [1]). Tato skutečnost mě motivovala k tomu, abych se pokusil vytvořit reklamní banner a vyzkoušet tak JavaFX právě při tvorbě animovaného banneru. Schopností banneru by samozřejmě mělo být přemístit uživatele prohlížečem na jiný web. Jako téma banneru jsem zvolil reklamu na Pedagogickou fakultu Jihočeské univerzity.

3.3.2 Návrh uživatelského rozhraní

Vzhledem k tomu, že cílem je vytvořit banner, nebude se tedy jednat o klasické uživatelské rozhraní, ale o animaci s možností kliknutí.

V první časti bude banner zobrazovat střed města České Budějovice s textem „Chcete studovat na jihu Čech?“. Fotografie města by se při tom měla pohybovat, aby měl pozorovatel dojem, že letí nad městem. Po několika vteřinách dojde k prolnutí do další fotografie, která by opět nějakým pohybem přispěla k dynamice banneru. Na této fotografii bude budova Pedagogické fakulty. V této části by byl také zobrazen text „Široké spektrum studijních programů Vám nabízí...“ a po chvilce text „Pedagogická fakulta Jihočeské univerzity v Českých Budějovicích“. Fotografie by se měla opět plynule ztratit. V závěru by pak bylo efektní, kdyby se z tečky nad „i“ ve slově „Pedagogická“ za letu stalo logo Jihočeské univerzity.

Pokud uživatel klikne na banner, bude přemístěn na adresu `www.pf.jcu.cz`. Předání cílové adresy samotnému banner by mělo být realizováno obdobně jako u běžných reklamních bannerů přes parametr aplikace v HTML kódu. (Zdroj [8])

3.3.3 Implementace

Aplikace Banner byla rozdělena na jednotlivé třídy podle logických částí aplikace. Každá část animace má svou třídu. Toto rozdělení ukazuje tabulka 6.

Třída/soubor	**Popis**
`AnimElement`	Třída definovaná s modifikátorem `mixin`. Obsahuje dvě abstraktní metody. Třída je společná pro všechny animované objekty v banneru.
`CBImage`	Třída představující animace obrázku středu města České Budějovice.
`ClickArea`	Třída reprezentující oblast pro kliknutí.
`Config`	Soubor `Config.fx` obsahuje definice globální proměnných.
`JCULogo`	Třída představující animace obrázku loga JČU.
`MainTimeline`	Třída reprezentující hlavní časovou osu animace banner.
`PFJUBuilding`	Třída představuje animaci obrázku budovy PF.
Tabulka pokračuje na následující stránce…	

Text1	Třída, která reprezentuje animovaný text v první časti banneru.
Text2	Třída představující animovaný text v druhé části banneru.
Text3	Třída reprezentuje animovaný text ve třetí části banneru.
WaterMark	Třída, která reprezentuje drobný text v pravé části celého banneru.

Tabulka 6: Popis jednotlivých tříd aplikace Banner

3.3.3.1 Hlavní časová osa

Hlavní osa animace banneru je představována objektem třídy Timeline, který je definován ve třídě MainTimeline.

Kód 24 *Ukázka hlavní časové osy, spouštění jednotlivých animací a změna viditelnosti prvků*

```
/* MainTimeline.fx */
def timeLine = Timeline {
    repeatCount:Timeline.INDEFINITE
    framerate: 20
    keyFrames: [
            KeyFrame {
                time: 0s
                action: function () : Void {
                    text1.playAnim();
                    ...
                }
            }
            ...
            KeyFrame {
                time: 1s
                values: [text1.opacity => 1 tween Interpolator.LINEAR, ...]
            }
            ...
```

3.3.3.2 Animace „přeletu nad městem“

Obrázek 3.8: Začátek animace banneru

Úvodní animace banneru je obstarávána třídou `CBImage`, která využívá schopností třídy `ParallelTransition` spouštět několik přechodů najednou. V animaci jsem použil třídy `ScaleTransition`, `RotateTransition` a `TranslateTransition`. Všechny tyto přechody jsou aplikovány na obrázek centra Českých Budějovic. Tím vzniká jakýsi dojem přeletu nad městem. Třída `CBImage` je potomkem třídy `CustomNode`, můžeme tedy aplikovat `ParallelTransition` na ní samotnou, jak je ukázáno v kódu 25.

Kód 25 *Použití tří animačních přechodů najednou díky třídě ParallelTransition*

```
/* CBImage.fx */
...
def transTransition = TranslateTransition {
    duration: 5s
    fromX: -40
    ...
def scaleTransition = ScaleTransition {
    duration: 5s
    ...
def rotateTransition = RotateTransition {
    ...
def totalTransition = ParallelTransition {
    content: [transTransition, scaleTransition, rotateTransition]
    node: this
}
...
...
public override function playAnim() : Void {
   totalTransition.playFromStart();
}
...
```

V této části animace banneru je navíc aplikován animační přechod třídy `ScaleTransition` na text. O animaci tohoto textu se stará třída `Text1`. Tento text se pod působením animačního přechodu `ScaleTransition` roztahuje. Navíc je na text aplikován efekt třídy `MotionBlur`. Parametr tohoto efektu je svázán s proměnnou `blurRadius`, která se mění díky časové ose `animTimeline` třídy `Timeline`. Toto ukazuje kód 26.

Kód 26 *Animace textu pomocí třídy Text1 v první časti celkové animace banneru*

```
/* Text1.fx */
...
def textNode = Text {
    content: "Chcete studovat vysokou školu na jihu Čech?"
    font: Font {size: 30}
    ...
def scaleTransition = ScaleTransition {
    duration: 5s
    node: textNode
    ...
def animTimeline = Timeline {
    repeatCount: 1
    keyFrames: [
            at    (0ms) { blurRadius => 20.0 tween Interpolator.LINEAR }
            at (1500ms) { blurRadius => 0.0 tween Interpolator.LINEAR }
            ...
```

Další části animace banneru jsou vytvořeny velmi podobně. Zmínil bych se však ještě o jednom animační přechodu, který nastane v závěru celkové animace banneru. Jedná se o letící kuličku, která letí z místa tečky nad písmenem „i“ ve slově „Pedagogická“. Kulička letí po eliptické dráze a poté se z ní stane logo Jihočeské univerzity (obrázek 3.9).

Obrázek 3.9:Vyznačená dráha letu kuličky díky přechodu PathTransition

Jedná se o použití třídy `PathTransition`, která vytváří přechod umožňující vést daný objekt po libovolné dráze (Zdroj [21]). Pro lepší představu uvádím část kódu, který je použit ve třídě JCULogo (viz kód 27).

Kód 27 *Vytvoření animovaného přechodu po křivce pro objekt silhouettePoint*

```
/* JCULogo.fx */
...
def pathTransition = PathTransition {
    duration: 2s
    path: AnimationPath.createFromPath(
        Path {
            elements: [
                MoveTo {x: SP_STARTX y: SP_STARTY},
                ArcTo {
                    x: SP_ENDX, y: SP_ENDY
                    radiusX: 50, radiusY: 9
                }
            ]
    })
    repeatCount:1 autoReverse: false
    node: silhouettePoint
}
...
```

3.3.3.3 Reakce na kliknutí na banner

Cílem každého kliknutí na banner je načtení požadované adresy do okna internetového prohlížeče. O tuto reakci se stará třída `ClickArea`, která je potomkem třídy `CustomNode` a vytváří neviditelný obdélník jako oblast pro klikání. Díky skutečnosti, že se jedná o třídu dědící vlastnosti třídy `CustomNode`, je možné implementovat obslužnou metodu události kliknutí myši `onMouseClicked` (viz kód 28).

Pro otevření adresy v prohlížeči jsem využil třídu `AppletStageExtension` resp. její metodu `showDocument()`, které je předána adresa cílové stránky. Tato adresa je získána z parametru aplikace Banner v HTML kódu stránky. Tento parametr jsem pojmenoval `clickthru` a jeho obsah je získáván metodou `getArgument()`, ze třídy `FX`. Kód 28 osvětluje celý způsob reakce na kliknutí a otevření adresy v prohlížeči.

Kód 28 *Část kódu třídy ClickArea*

```
/* ClickArea.fx */
...
public class ClickArea extends CustomNode {
    public override function create() : Node {
        Rectangle {
            x:0 y:0
            width: Config.BANNER_WIDTH
            height: Config.BANNER_HEIGHT
            opacity: 0
            onMouseClicked: function (e:MouseEvent) : Void {
                var openUrl:String = null;
                try {
                    openUrl = FX.getArgument("clickthru").toString();
                    AppletStageExtension.showDocument(openUrl);
                }
                catch (exc:Exception) {
                ...
```

3.3.3.4 Vložení aplikace do webové stránky

Pro vložení banneru do webové stránky stačí vložit HTML kód, který je uveden v kódu 29. Z kódu lze vyčíst, že je třeba specifikovat cestu k souboru s JavaFX aplikací pomocí parametru `archive`, dále rozměry pro zobrazení ve stránce, umístění hlavní třídy a název celé aplikace. Dále je specifikován parametr `clickthru`, které obsahuje cílovou adresu pro kliknutí na banner.

Kód 29 *HTML kód stránky, vložení banneru a parametr clickthru*

```
<!-- http://javafx.xoe.cz/aplikace/banner/ -->
...
<script src="http://dl.javafx.com/1.3/dtfx.js"></script>
<script>
    javafx(
        {   archive: "http://javafx.xoe.cz/banner/Banner.jar",
            width: 728,
            height: 90,
            code: "banner.Main",
            name: "Banner"
```

```
        },
        {   clickthru: "http://www.pf.jcu.cz" }
    );
</script>
...
```

3.3.4 Přenositelnost aplikace Banner

Banner je aplikace, kterou nelze přeložit a spustit pro mobilní zařízení kvůli použití třídy `AppletStageExtension`, která je k dispozici pouze pro profil desktop a browser (Zdroj [21]). Třídu `AppletStageExtension` však nelze z kódu aplikace vypustit, protože ve třídě `ClickArea` poskytuje prostřednictvím metody `showDocument()` možnost otevřít cílovou webovou stránku. Aplikace Banner je tedy spustitelná jen jako desktopová a webová aplikace.

3.3.5 Závěr k aplikaci Banner

Aplikace Banner ukazuje použitelnost JavaFX i pro tvorbu reklamních bannerů. Vzhledem k poměrně pracnému programování jednotlivých animací a přechodů by se však pro tvorbou bannerů hodil nějaký nástroj pro vizuální programování, který by zmenšil nutnost psaní kódu a sám by jej během tvorby banneru generoval.

3.4 SpaceGame

3.4.1 Návrh tématu a činnosti aplikace

Cílem je vytvořit program, který bude typickým a zároveň populárním příkladem RIA aplikace. Půjde o aplikaci, která bude představovat počítačovou hru schématicky podobnou prvním počítačovým hrám z konce sedmdesátých a začátku osmdesátých let dvacátého století. (Zdroj [28])

Děj hry by měl být vsazen do prostředí vesmíru, kde se hráč se svojí kosmickou lodí snaží ničit asteroidy, které letí ve směru proti jeho lodi. Za každý zničený asteroid je hráči připsán bod. Celkový součet získaných bodů je zobrazen během hry v pravém dolním rohu.

Kosmická loď hráče je ovládána šipkami na klávesnici, vypouštění střel je ovládáno mezerníkem. Při každém stisknutí mezerníku loď vypustí po svých stranách dvě laserové střely. Pokud střela zasáhne asteroid, zobrazí se výbuch, který znamená zničení asteroidu. Asteroid i střela tím okamžikem zmizí. Pokud je loď zasažena asteroidem, hra při výbuchu lodi končí. Hra by se měla postupně narůstajícím počtem zničených asteroidů zrychlovat, aby se hraní nestávalo postupem času monotónním. V každém okamžiku by mělo být možné hru pozastavit, ideálně klávesou „P“.

3.4.2 Návrh grafického provedení

Grafické provedení hry by mělo díky použití JavaFX samozřejmě snadno překonat podobu prvních počítačových her. Základní grafické pozadí hry bude tvořeno černým obdélníkem, který představuje barvu vesmíru. Přes tento obdélník se bude při hře přesouvat obrázek obsahující tmavě modré shluky představující vesmírné mlhoviny. Tímto by měl pozorovatel získat dojem, že loď pluje vesmírem. Vždy, když se obrázek s pozadím přesune mimo viditelnou oblast hrací plochy, se jeho vertikální souřadnice nastaví do záporných hodnot tak, aby plynule opět připlul do viditelné části hrací plochy. K získání lepšího dojmu z pohybu by měly být přidány i plovoucí hvězdy, které se pohybují ve stejném směru jako pozadí, jen trochu rychleji. Kombinací přesouvajícího se obrázku pozadí a pohybujících se hvězd lze dosáhnout základního trojrozměrného efektu. Letící asteroidy budou z grafického hlediska polygony s barevně přiměřenou výplní. Navíc by měly asteroidy vykonávat nějaký pohyb kolem své osy, aby pouze nepluly vesmírem a nepůsobily tak příliš staticky. Dalším prvkem, který se bude v aplikaci SpaceGame objevovat jsou exploze. Ty by měly mít patřičně dynamický průběh, aby obohatily celkový dojem ze hry. Měly by mít kruhový tvar a oranžovou barvu. Přesnější vzhled bude upřesněn až při vlastní realizaci aplikace.

V aplikaci by se měly objevit i zvukové efekty. Konkrétně by měly doprovázet výstřely a výbuchy. Pomineme-li fakt, že se ve vakuu nešíří zvuk, získá tím aplikace dojem regulérní počítačové hry.

3.4.3 Implementace

Celý projekt aplikace byl rozdělen na několik tříd, které v aplikaci reprezentují konkrétní grafické objekty a řídí jejich chování. Toto rozdělení je uvedeno v tabulce 7. Dále bych se chtěl v jednotlivých podkapitolách věnovat hlavním částem implementace hry.

Třída/soubor	Popis
`Actor`	Rodičovská třída s modifikátorem dědičnosti `mixin`. Deklaruje metody a proměnné společné pro všechny podtřídy a abstraktní metodu run.
`Asteroid`	Třída představující asteroid. Potomek třídy `Actor` a standardní třídy `CustomNode`.
`Background`	Třída reprezentující grafické pozadí hry. Představuje obrázek pozadí a hvězd (pole objektů třídy `Star`).
`Explosion`	Třída představující explozi. Potomek třídy `Actor` a standardní třídy `CustomNode`.
`GameArea`	Třída zastřešující všechny objekty ve hře, definuje hlavní smyčku a realizuje zadávání příkazů od uživatele.
`GameState`	Třída reprezentující stav hry. Hra se může dostat do čtyř stavů: intro, konec hry, pauza a vlastní hra.
`LaserShot`	Třída představující objekt laserové střely. Potomek třídy `Actor` a standardní třídy `CustomNode`.
`SoundManager`	Třída zajišťující nahrání a přehrávání zvuků během hry.
`SpaceShip`	Třída představující objekt vesmírné lodi. Potomek třídy `Actor` a standardní třídy `CustomNode`.
`Star`	Třída reprezentující objekt hvězdy. Potomek třídy `Actor` a standardní třídy `CustomNode`.
`Stage`	Třída představující hlavní okno aplikace. Třída je obsažena v souboru `Main.fx`.
`StatusPlace`	Třída představující stavové okénko v pravém dolním rohu. Potomek standardní třídy `CustomNode`.

Tabulka 7: Přehled tříd aplikace SpaceGame

3.4.3.1 Pohybující se objekty – třída Actor

Pro několik tříd aplikace, které reprezentují pohybující se objekty, je vytvořen rodičovská třída `Actor` s modifikátorem `mixin`. Ta obsahuje deklarace proměnných, které představují polohu daného objektu na hrací ploše. Dále jsou ve třídě `Actor` deklarovány proměnné `speed` a `used`. První znamená rychlost pohybu a druhá indikátor, zda je objekt právě ve hře používám.

Jako společná metoda pro všechny podtřídy je zde deklarovaná abstraktní metoda `run()`. Ta má díky modifikátoru `abstract` v každé z podtříd svou vlastní podobu. Vždy se ale jedná o zajištění pohybu a „života“ objektu.

Kód 30 *Třída Actor*

```
/* Actor.fx */
...
public mixin class Actor {
    protected var posX : Number;
    protected var posY : Number;
    protected var vecX : Number;
    protected var vecY : Number;
    protected var speed : Number;
    protected var used : Boolean = false;
    public abstract function run() : Void;
    public function getX() : Number {return posX;}
    public function getY() : Number {return posY;}
}
...
```

3.4.3.2 Hlavní smyčka aplikace

Aby aplikace mohla plynule fungovat a průběžně provádět nezbytnou činnost, je třeba v kódu hry použít smyčku, která by vždy po uplynutí konstantní doby volala metodu, která vše potřebné provede. V každém kroku je třeba provést například detekci kolizí mezi střelami a asteroidy a také mezi vesmírnou lodí a asteroidy, dále pak zavolat metodu `run()` u všech potomků třídy `Actor` a zajistit tak činnost všech objektů ve hře.

Kód 31 *Definování objektu hlavní smyčky*

```
/* GameArea.fx */
...
def gameLoop : Timeline = Timeline {
    repeatCount: Timeline.INDEFINITE
    keyFrames: [
        KeyFrame {
            time: 50ms
            action: function() {
```

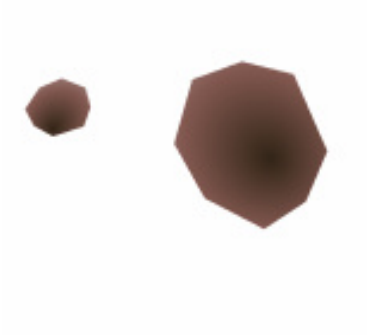
```
                gameUpdate();
            }
        }
    ]
}
...
```

V případě aplikace SpaceGame je hlavní smyčka definována ve třídě `GameArea` a zajišťována instancí standardní JavaFX třídy `Timeline` z balíčku `javafx.animation`. Ta každých 50 milisekund volá metodu `gameUpdate()` třídy `GameArea`.

3.4.3.3 Asteroidy

Grafické znázornění asteroidu

Pro grafické znázornění asteroidu se nejlépe hodilo použít mnohoúhelník s vhodnou barevnou výplní. Toto kritérium splňuje objekt třídy `Polygon`. Takové řešení nevyžaduje načtení obrázku a má i další výhodu jako je přesná detekce kolizí podle tvaru asteroidu. Mnohoúhelník znázorňující asteroid má 8 vrcholů (viz obrázek 3.10). Jako výplň mnohoúhelníku je použit kruhový barevný gradient, objekt třídy `RadialGradient`.



Obrázek 3.10: Asteroidy

Pro animaci rotace asteroidu kolem vlastní osy byla použita standardní třída z JavaFX API s názvem `RotateTransition` (Zdroj [21]). Tato třída je umístěna v balíčku `javafx.animation.transition`.

Kód 32 *Animace rotace asteroidu*

```
/* Asteroid.fx */
...
rotateTrans = RotateTransition {
    duration: bind Duration.valueOf(rotSpeedms)
    node: this
    byAngle: 360
    repeatCount: Timeline.INDEFINITE
    interpolator: SimpleInterpolator.LINEAR
    autoReverse: false
}
rotateTrans.play();
...
```

Třída typu Asteroid má deklarovanou proměnnou `rotateTrans`, která je při vzniku objektu inicializována novou instancí třídy `RotateTransition`. Proměnná `rotSpeedms` je již naplněna náhodnou hodnotou a následně svázána s parametrem `duration`. Tím je zajištěno, že rychlost rotace každého asteroidu je různá. Následně je animace spuštěna metodou `play()`.

Chování asteroidu

Pohyb asteroidů je realizován neustále se opakujícím zvětšováním hodnoty vertikální pozice v metodě `run()` třídy `Asteroid`. Pokud je tato pozice větší než velikost scény, nastaví se vertikální pozice na zápornou, horizontální pozice je nastavena náhodně a asteroid je v podstatě přemístěn nahoru, nad viditelný okraj scény. Zároveň je zvýšena hodnota proměnné `speed`, která určuje rychlost asteroidu. Každý asteroid tak ve hře létá stále dokola a při každém novém průchodu je rychlejší.

Kód 33 *Metoda run() asteroidu*

```
/* Asteroid.fx */
...
public override function run(): Void {
    posY += speed;
    if (posY > scene.height) {
        var rand: Random = Random{};
        posX = Math.abs(rand.nextInt() mod scene.width);
        ...
```

```
        posY = -60;
        speed++;
    }
}
...
```

3.4.3.4 Pozadí

Grafické znázornění pozadí

Pro lepší dojem z vesmírného prostoru bylo třeba vytvořit pozadí, které bude vyplňovat celý prostor scény. Pozadí scény je vytvořeno ve třídě `Background` a skládá se z černého obdélníku (standardní třída `Rectangle`), hvězd a obrázku, na kterém jsou náznaky vesmírných mlhovin. Třída `Background` obsahuje sekvenci objektů třídy Star. Tato sekvence je naplněna při inicializaci třídy. Součástí třídy `Background` je také proměnná `backgroundImage`, což je objekt třídy `ImageView` z API JavaFX (Zdroj [21]). Tento objekt představuje obrázek pozadí vesmíru. Vertikální pozici obrázku bylo třeba svázat s proměnnou třídy `Background posY`, aby se měnící se hodnota proměnné projevovala posouváním obrázku pozadí. Roztažení obrázku pozadí podle rozměrů scény pak zajišťuje svázání `fitWidth` s proměnnou `scene.width` a `fitHeight` s proměnnou `scene.height`.

Kód 34 *Vytvoření grafické reprezentace pozadí*

```
/* Background.fx */
...
def backgroundImage = ImageView {
    image: Image {url: "{__DIR__}background.jpg"}
    y: bind posY
    fitWidth: bind scene.width
    fitHeight: bind scene.height*2
}
...
var stars : Star[];
...
public override function create(): Node {
    Group {
        content: [
```

```
            Rectangle {
                x: 0  y: 0
                width: bind scene.width
                height: bind scene.height
                fill: Color.rgb(0, 0, 0)
            },
            backgroundImage,
            stars
        ]
    }
}
...
```

Chování pozadí

Pohyb resp. posun pozadí, tak aby docházelo k dojmu letící vesmírné lodi, je realizován v metodě `run()` třídy `Background`. Nejprve se prochází sekvence objektů hvězd a u každého je volána metoda `run()`. Poté je zvýšena vertikální pozice obrázku pozadí v proměnné `posY`. Pokud se tato pozice obrázku pozadí dostane mimo scénu, tedy nad hodnotu aktuální výšky scény, je nastavena na záporný dvojnásobek výšky scény. Tím jsem docílil plynulého opakování plutí pozadí.

Vždy, když pozadí vyjede v dolní části ze scény, tak se plynule zase objeví v horní časti. Jeden obrázek tak neustále navozuje dojem pohybu ve vesmíru.

Kód 35 *Metoda run() třídy Background*

```
/* Background.fx */
...
public override function run(): Void {
    for (star in stars) star.run();
    posY+=2;
    if (posY > scene.height) {
        posY = -scene.height*2;
    }
}
...
```

3.4.3.5 Hvězdy

Skupina pohybujících se hvězd je řešena jako sekvence objektů třídy `Star` v třídě `Background`. Samotná třída `Star` je potomek (stejně jako všechny ostatní třídy představující pohyblivé objekty) tříd `CustomNode` a `Actor`.

Grafická podoba hvězdy (třídy `Star`) je jako u každého potomka třídy `CustomNode` obstarána v metodě `create()`. Pro vytvoření objektu hvězdy jsem zvolil vertikální úsečku o délce jeden pixel. Pro vykreslení úsečky se v JavaFX používá třída `Line` (Zdroj [21]). Pozice počátečního a koncového bodu úsečky jsou svázány s proměnými `posX` a `posY`, aby se při změně těchto proměnných neustále projevovaly změny polohy úsečky. Použitím vlastnosti `stroke` jsem nastavil barvu čáry na bílou.

Kód 36 *Grafická reprezentace hvězdy pomocí úsečky*

```
/* Star.fx */
...
Line {
    startX: bind posX
    startY: bind posY
    endX: bind posX
    endY: bind posY+1
    stroke: Color.rgb(255, 255, 255)
    opacity: 0.7
    strokeWidth: 1
}
...
```

Při inicializaci třídy `Star` je nastavena náhodná pozice a rychlost hvězdy. Horizontální i vertikální pozice je nastavena náhodně v rozsahu podle aktuálně vybraného cílového profilu zařízení.

Kód 37 *Inicializace proměnných třídy Star*

```
/* Star.fx */
...
init {
  posX = rand.nextInt() mod (if (__PROFILE__=="mobile") then 240 else 640);
  posY = rand.nextInt() mod (if (__PROFILE__=="mobile") then 320 else 480);
```

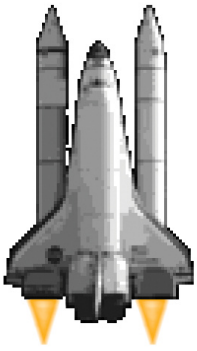
```
  speed = 1+Math.abs(rand.nextInt() mod 3);
}
...
```

Pohyb hvězdy je realizován v metodě `run()` zvyšováním hodnoty vertikální pozice hvězdy, tedy zvyšováním hodnoty proměnné `posY`. Jakmile je pozice mimo viditelnou scénu, je proměnné `posY` přiřazena nula a hvězda se tak ocitá opět na horním okraji scény.

3.4.3.6 Objekt vesmírné lodi

Grafické znázornění vesmírné lodi

Vesmírnou loď představuje v aplikaci třída `SpaceShip`. Graficky je loď znázorněna obrázkem `spaceShipImage`, který je definován jako objekt třídy `ImageView`. Dále jsem k obrázku ještě přidal dva objekty třídy `Polygon`, které mají znázorňovat plameny šlehající z tryskových motorů (viz obrázek 3.11). Tyto mnohoúhelníky mají tři vrcholy přičemž poloha spodního vrcholu se neustále mění a budí tak dojem aktivní trysky. Vesmírná loď díky tomu nepůsobí tak staticky. Pozice obrázku `spaceShipImage` i pozice polygonů trysek je svázána s proměnnými `posX` a `posY`, které představují pozici vesmírné lodi ve scéně.



Obrázek 3.11: Vesmírná loď, grafická reprezentace třídy SpaceShip

Na výplň polygonů tvořících plameny z trysek je použit kruhový gradient barev třídy `RadialGradient`. Pro ještě lepší dojem z plamenů je na polygony aplikován efekt třídy `MotionBlur`, který je uložen ve třídě `SpaceShip` v proměnné `fireBlur`.

Třída `MotionBlur` vytváří efekt, který rozmázne daný grafický prvek v zadaném směru (Zdroj [21]). Pro vytvoření pohyb plamenů z trysek lodi jsem použil třídu `Timeline`. Ta pomohla vytvořit časovou osu pro animaci plamenů, která řídí kolísání hodnoty v proměnné `jetFireSize`.

Kód 38 *Definování časové osy animace pro pohyb plamenů v tryskách vesmírné lodi*

```
/* SpaceShip.fx */
...
def jetFireAnim = Timeline {
    repeatCount: Timeline.INDEFINITE
    autoReverse: true
    keyFrames: [
        at (0ms) { jetFireSize =>  0.0 tween Interpolator.LINEAR }
        at (300ms) { jetFireSize => 10.0 tween Interpolator.LINEAR }
    ]
}
...
```

Chování efektu `MotionBlur` jsem také využil při vytvoření dojmu rozmáznutí lodi při pohybu aplikováním na obrázek `spaceShipImage`. Toto funguje tak, že jsou neustále upravovány proměnné vyjadřující úhel a sílu efektu podle směru a rychlosti pohybu.
Tyto proměnné jsou vázány v deklaraci efektu v `effectMotionBlur`. Výpočet úhlu a poloměru (síly) pro efekt `MotionBlur` je prováděn v metodě `calcMotionBlur()` pomocí okamžitého směru vesmírné lodi.

Kód 39 *Definování efektu směrového rozmáznutí při pohybu lodi a výpočet směru a poloměru*

```
/* SpaceShip.fx */
...
def effectMotionBlur = MotionBlur {
    radius: bind mbRadius
    angle: bind mbAngle
};
...
```

```
function calcMotionBlur(): Void {
    mbAngle = Math.atan(vecY/vecX)/Math.PI*180.0;
    if (Math.abs(vecX) > Math.abs(vecY)) {
        mbRadius = Math.abs(vecX);
    } else {
            mbRadius = Math.abs(vecY);
    }
}
...
```

Chování vesmírné lodi

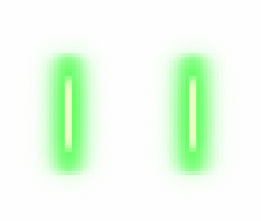
Ve třídě `SpaceShip` je definována metoda `run()`, která je společná pro všechny potomky třídy `Actor`. V této metodě je zajišťován „život“ objektu, tedy plynulost pohybu lodi, čítač destrukce lodi a volání již zmíněné metody `calcMotionBlur()`.

Třída `SpaceShip` dále obsahuje metody s prefixem „`move`“, které nastavují požadovaný směr pohybu lodi a spouští vlastní pohyb. Tyto metody jsou volány při detekci stisku příslušné klávesy ze třídy `GameArea`. Dále v třídě `SpaceShip` existují metody s prefixem „`stopMove`“, které nulují povel pro pohyb v dané ose. Tyto metody jsou naopak volány po uvolnění klávesy. Výsledkem je pak plynulé brždění předchozího pohyb až do klidu.

3.4.3.7 Střely

Grafické znázornění střely

Z grafického hlediska střelu tvoří dva obdélníky (třída `Rectangle`) přičemž jeden obdélník je větší. Obdélník o větší velikosti je zde proto, aby vytvářel díky poloviční neprůhlednosti (opacity) jakési světlo okolo menšího obdélníku. U těchto obdélníků bylo třeba zaoblit rohy, aby střela měla hladký tvar (viz obrázek 3.12). Toho lze docílit inicializací proměnných `arcWidth` a `arcHeight` třídy `Rectangle` (Zdroj [21]). Dále je na větší obdélník aplikován efekt `MotionBlur`, který ještě více posílí dojem ze světla okolo samotné střely. Viditelnost obou obdélníků je svázána s proměnnou `used`. Ta je zděděna ze třídy `Actor`. Dále je samozřejmě svázána i pozice obou obdélníků s proměnnou `posX` a `posY`.



Obrázek 3.12: Dvě letící střely

Chování střely

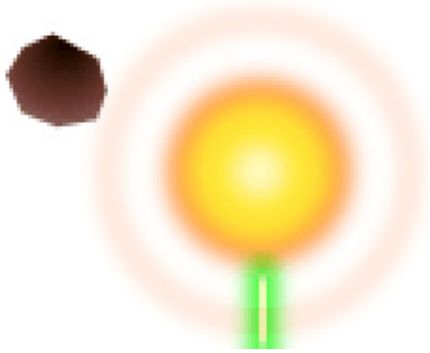
Sekvence objektů třídy `LaserShot` je obsažena v třídě `GameArea`. Samotná třída `LaserShot` je potomkem tříd `CustomNode` a `Actor`. Střela je „nastartována“ z požadované pozice metodou `start()`. Metoda `run()`, která je volána pro každou střelu ze třídy `GameArea`, zajišťuje pohyb střely zmenšováním proměnné `posY`. To se však děje pouze, pokud je střela aktivní, neboli pokud má proměnná `used` hodnotu `true`. Detekce kolizí je řešena ve třídě `GameArea`.

3.4.3.8 Exploze

Grafické znázornění exploze

Exploze je z grafického hlediska tvořena třemi kruhy. V JavaFX se o zobrazení kruhu postará třída `Circle`. První kruh je bez výplně a jde tedy pouze o jakýsi prstenec s největším poloměrem. Další kruhy již mají barevnou výplň (viz obrázek 3.13). Pozice všech kruhů je svázána s proměnnými `posX` a `posY`. Velikost poloměrů všech kruhů je svázána s násobky proměnné `phase`. Na skupinu všech tří kruhů je aplikován efekt `GaussianBlur` (rozmáznutí).

Efekt rozmáznutí (`GaussianBlur` i `MotionBlur`) je poměrně náročný na výkon zařízení (počítače), proto by jeho použití mělo být dobře zváženo. Vlastním pozorováním jsem zjistil, že čím větší je míra rozmáznutí objektu, tím se výpočetní čas prodlužuje. Tento jev lze pozorovat i za běhu aplikace SpaceGame, kdy na scéně běží několik explozí najednou. Je velmi pravděpodobné, že novější verze JavaFX přinesou další zlepšení výkonu aplikací tak, jak tomu bylo i při přechodu z verze 1.2 na verzi 1.3 (Zdroj [15]).



Obrázek 3.13: Exploze

Kód 40 *Vytvoření grafické podoby exploze*

```
/* Explosion.fx */
...
public override function create(): Node {
    Group {
        effect: GaussianBlur {radius: 10}
        content: [
            Circle {
                centerX: bind posX
                centerY: bind posY
                radius: bind phase*20
                fill: null
                visible: bind used;
                ...
```

Chování exploze

Třída `Explosion` představující explozi je taktéž potomek tříd `CustomNode` a `Actor`. Sekvence objektů explozí je obsažena ve třídě `GameArea`. Samotná exploze se „startuje“ na požadované pozici metodou `start()`. Třída `Explosion` dále definuje metodu `run()`, kde je implementován průběh exploze. Poté, co proměnná `phase` dosáhne hodnoty `10`, je exploze odstavena přiřazením hodnoty `false` proměnné `used`. Kód 40 ukazuje svázání důležitých parametrů exploze pomocí konstrukce `bind` a v kódu 41 je zapsána metoda `run()`.

Kód 41 *Metoda run() třídy Explosion*

```
/* Explosion.fx */
...
public override function run(): Void {
    if (used==true)
    {
        phase++;
        if (phase >= 10) used = false;
    }
}
...
```

3.4.3.9 Ukazatel bodů

Grafické znázornění ukazatele bodů

Ukazatel bodů představuje třída `StatusPlace` (viz obrázek 3.14). Vzhled ukazatele bodů je vytvořen obdélníkem se zaoblenými rohy. Dále textem „DESTROYED ASTEROIDS“, který je vytvořen pomocí třídy `Text`. A nakonec hlavním textem, který ukazuje počet zničených asteroidů. Jedná se opět o použití třídy `Text`. Pro vycentrování textu s číslem, byla použita třída `Stack`. Ta má takovou vlastnost, že veškeré objekty, které jsou vloženy do její proměnné `content`, vycentruje podle své velikosti. Tím je zajištěno, že každé číslo bude zarovnáno na střed. Velikost ohraničujícího obdélníku i použitého písma je definována podle profilu cílového zařízení. Pro mobilní zařízení je tedy celková velikost ukazatele bodů menší.

Obrázek 3.14: Ukazatel bodů

Kód 42 *Definování proměnných pro ukazatel bodů na základě profilu cílového zařízení*

```
/* StatusPlace.fx */
...
def textNode = Text {
    font: Font {
        name: "Arial Bold"
        size: if (__PROFILE__=="mobile") then 20 else 45
    }
    content: bind "{destroyedAsteroidsValue}"
    effect: GaussianBlur {radius: 2}
    fill: Color.rgb(0,240,0,0.9)
}
def destroyedTitleFont = Font {
    size: if (__PROFILE__=="mobile") then 7.5 else 11.5
}
def statPlaceWidth = if (__PROFILE__=="mobile") then 100 else 120;
def statPlaceHeight = if (__PROFILE__=="mobile") then 48 else 60;
...
```

Pro ukazatel bodů jsem v poslední fázi vývoje hry připravil reakci na zvýšení počtu zničených asteroidů. Ta spočívá v tom, že číslo zapulzuje, neboli se roztáhne a hned zase smrští do původní velikosti. Při vytváření tohoto efektu jsem využil konstrukci spouštěče jazyka JavaFX Script, který zde hlídá počet zničených asteroidů, tedy proměnnou `destroyedAsteroidsValue`. Při každé změně se pak spustí změna měřítka textu. Tato změna je definována jako objekt třídy `ScaleTransition` a aplikována na objekt `textNode`, tedy objekt s textem hodnoty zničených asteroidů.

Kód 43 *Definování přechodu a spouštěče pro reakci na zvýšení počtu zničených asteroidů*

```
/* StatusPlace.fx */
...
def scaleTransition = ScaleTransition {
    duration: 300ms
    node: textNode
    fromX: 1.0 fromY: 1.0
    toX: 1.2 toY: 1.2
    repeatCount:2 autoReverse: true
```

```
    interpolator: Interpolator.EASEIN
}
public var destroyedAsteroidsValue : Integer = 0 on replace {
    scaleTransition.playFromStart();
}
...
```

3.4.3.10 Oznamovací zpráva

Grafické znázornění oznamovací zprávy

Oznamovací zpráva je z pohledu grafiky text s ohraničujícím obdélníkem (viz obrázek 3.15). Vzhled je deklarován ve třídě `GameState`. Pro vycentrování obdélníku i textu bylo vhodné použít třídu `Stack` stejně jako u centrování textu v ukazateli bodů.

Třída `GameState` v sobě drží informaci o tom, v jakém stavu se hra právě nachází. Jsou definovány 4 stavy: `INTRO`, `GAMEOVER`, `PAUSED`, `GAME`. Pro objevení a mizení oznamovací zprávy jsem použil přechody JavaFX třídy `FadeTransition`, které jsou aplikovány na kompletní rámeček se zprávou. Díky nim je zobrazení i mizení zprávy pěkně plynulé.

Obrázek 3.15: Oznamovací zpráva (jedna ze tří možných)

Poloprůhlednost podkladu oznamovací zprávy je řešena standardně nastavením proměnné `opacity`, kterou je vybavena každá třída, která je potomkem třídy `CustomNode`.

Chování oznamovací zprávy

Nastavení nového stavu hry je prováděno metodou `setState()`. Pokud je nový zadávaný stav vlastní hraní hry (`newState==GAME`), je spuštěn přechod pro plynulé zmizení oznamovací zprávy. Jestliže je však nový stav jiný, spustí se přechod pro plynulé objevení zprávy. Tento princip osvětluje kód 44.

Kód 44 *Úryvek ze zdrojového kódu třídy GameState ukazující souvislosti se změnou stavu*

```
/* GameState.fx */
...
public function setState(newState : Integer) : Void {
    state = newState;
    if (state != GAME)
        fadeIn.playFromStart();
    else
        fadeOut.playFromStart();
}
...
def msgStack = Stack {
    width: bind scene.width
    height: bind scene.height
    content: [
        Rectangle {
            width: bind scene.width/1.5
            height: bind scene.height/2.0
...
...
def fadeIn = FadeTransition {
    node: msgStack
    fromValue: 0.0 toValue: 1.0
    ...
```

3.4.3.11 Detekce kolizí

Při kontaktu vesmírné lodi s asteroidem či kontaktu asteroidu se střelou by mělo dojít k příslušné reakci. Proto jsem ve třídě `GameArea` vytvořil metodu `checkCollisions()`, která se právě o reakce na kolize stará. Tato metoda je volána při každém průchodu hlavní smyčkou hry.

Kód 45 *Část metody checkCollisions() třídy GameArea*

```
/* GameArea.fx */
...
public function checkCollisions() : Void {
    for (shot in laserShots) {
        if (shot.isUsed()==true) {
            for (asteroid in asteroids) {
                if (asteroid.intersects(shot.boundsInParent)) {
                    asteroid.erase();
                    shot.erase();
                    destroyedAsteroids++;
                    startExplosion(shot.getX(),shot.getY());
                    sndManager.playSound(sndManager.SND_BOOM);
                }
                ...
```

Pro variantu kolize asteroidu se střelou je použit cyklus typu `for`, který prochází sekvenci střel. Ty, které jsou zrovna aktivní, jsou dále porovnány s každým asteroidem ze sekvence všech asteroidů. Samotné porovnávání je realizováno metodou `intersects()`, kterou třída `Asteroid` dědí ze třídy `CustomNode`, která je součástí JavaFX API (Zdroj [21]). Tato metoda zjišťuje, zda hranice objektu protíná jiný zadaný objekt. Metoda má jeden argument typu `Rectangle2D`. Ten je získán z objektu střely proměnnou `boundsInParent()`. Při zjištění překrytí asteroidu a střely vrací metoda `intersects()` hodnotu `true`. Reakcí na kolizi je zavolání metody `erase()` právě testovaného objektu asteroidu a také kolidující střely. Dále je zvýšena hodnota celkového počtu zničených asteroidů, spuštěna exploze na daném místě a přehrán zvuk exploze.

Dalším případem kolize je střet vesmírné lodi a asteroidu. Pokud tedy již neprobíhá destrukce lodi, kterou by potvrzovala kladná hodnota proměnné `crashCounter`, projde se sekvence asteroidů. Každý objekt ze sekvence je testován metodou `intersects()` podobně jako u testování kolize asteroidů se střelou. V tomto případě je však argumentem metody `intersects()` proměnná `boundsInParent()` objektu `spaceShip`. Pokud tedy hranice konkrétního asteroidu kolidují s hranicemi vesmírné lodi, jsou učiněny příslušné kroky pro odpovídající reakci, které jsou vypsány v kódu 46.

Kód 46 *Pokračování metody checkCollisions() třídy GameArea*

```
/* GameArea.fx */
...
for (asteroid in asteroids) {
    if (asteroid.intersects(spaceShip.boundsInParent)) {
        crashCounter = 3;

        spaceShip.destroy();
        var expX = ((spaceShip.getX() + 32) + asteroid.getX()) / 2;
        var expY = ((spaceShip.getY() + 53) + asteroid.getY()) / 2;
        startExplosion(expX, expY);
        sndManager.playSound(sndManager.SND_BOOM);
    }
}
...
```

3.4.3.12 Přehrávání zvuků

Schopnost aplikace přehrávat zvuky je v případě počítačové hry zásadní. Vzhledem k silně proklamované orientaci platformy JavaFX na snadnou práci s médii, není řešení příliš obtížné (Zdroj [16]). Pro zajištění přehrávání zvukových efektů jsem vytvořil samostatnou třídu `SoundManager`. Uvnitř třídy jsou definovány dvě proměnné jako zdroje zvukových efektů pomocí třídy Media. Zároveň třída `SoundManager` obsahuje sekvenci objektů třídy `MediaPlayer` s názvem `players`. Třída `MediaPlayer` je určena pro přehrávání médií (Zdroj [21]). Při inicializaci třídy `SoundManager` jsou objekty sekvence `players` inicializovány.

Kód 47 *Definice zdrojů se zvukem a inicializace přehrávačů zvuků*

```
/* SoundManager.fx */
...
def soundLaser = if (__PROFILE__!="mobile") then Media {
    source: http://javafx.xoe.cz/spacegame/laser.mp3
} else null;
...
def samples = [soundLaser, soundBoom];
var players : MediaPlayer[];
...
```

```
init {
    if (__PROFILE__!="mobile") {
        for (i in [0..MAX_SOUNDS]) {
            players[i] = MediaPlayer{};
            players[i].media = samples[i];
        }
    }
}
...
```

Samotné přehrávání zvuků jsem vyřešil metodou `playSound()`, která má jeden parametr. Tím je index zvukového efektu ze sekvence `samples`. Stejně jako u inicializace třídy, je i před přehráním požadovaného zvuku podmínka zakazující přehrání v profilu mobilního zařízení. Před přehráním požadovaného zvuku je příslušný objekt třídy `MediaPlayer` v sekvenci zastaven metodou `stop()`. Poté je teprve zavolána metoda `play()`. Zvolil jsem toto řešení proto, aby se vždy nastavila počáteční pozice pro přehrávání zvuku.

3.4.3.13 Ovládání hry

Hra je ovládána běžnými klávesami klávesnice. Konkrétně se jedná o šipky pro pohyb lodi, mezerník pro střílení, klávesu Enter a klávesu „P“. Reakce na vstup uživatele je implementována ve třídě `GameArea`. Vzhledem k tomu, že třída `GameArea` je potomek třídy `CustomNode`, umožňuje to využití zděděné vlastnosti `onKeyPressed`, do které lze přiřadit vlastní funkci, která bude na stisk klávesy reagovat.

Kód 48 *Implementace ovladače události stisknutí klávesy*

```
/* GameArea.fx */
...
public override var onKeyPressed = function ( e: KeyEvent ) : Void {
    if (gameState.getState()==gameState.GAME) {
        if (e.code == KeyCode.VK_P) {
            pauseAllAnimations();
            gameState.setState(gameState.PAUSED);
        }
        ...
```

```
        if ( e.code == KeyCode.VK_LEFT)
            spaceShip.moveLeft();
        if ( e.code == KeyCode.VK_UP )
            spaceShip.moveForward();
        ...
```

Obdobně je ve třídě `GameArea` využita i vlastnost `onKeyReleased`, které je přiřazena funkce reagující na uvolnění klávesy.

3.4.4 Komplikace při vývoji aplikace

Vzhledem k tomu, že v případě aplikace SpaceGame se jedná o poměrně větší program, rozhodl jsem se popsat některé zádrhely, které mě během vývoje hry potkaly. Jedním z těchto zádrhelů byl fakt, že emulátor mobilních zařízení pro platformu JavaFX, který je součástí prostředí NetBeans, vůbec nebyl schopen zobrazit obrázek ve formátu Gif se zapnutou transparentností. To se týkalo zobrazení vesmírné lodi. Bylo třeba použít jiný grafický formát, který podporuje transparentnost. Vybral jsem tedy grafický formát PNG ve verzi pro osm bitů. (Zdroj [28])

Dalším překvapivým problémem byla neschopnost emulátoru mobilních zařízení v prostředí NetBeans nahrát mp3 soubory se zvuky. Řešení tohoto problému se mi nepodařilo nalézt. Problém pravděpodobně souvisí s novým omezením platformy JavaFX ve verzi 1.3, kdy již sestavovací program nevkládá soubory médií s programem do stejného `jar` archivu (Zdroj [15]). Byl jsem tedy nucen na několik místech ve zdrojovém kódu třídy `SoundManager` vložit omezující podmínku pro nahrání a přehrání zvuků. Toto řešení ukazuje zdrojový kód 49.

Kód 49 *Nahrávání a přehrávání zvuku pouze na „nemobilním“ zařízení*

```
/* SoundManager.fx */

if (__PROFILE__!="mobile") {
    ...
    ...
}
```

3.4.5 Přenositelnost aplikace SpaceGame

Aplikace SpaceGame byla úspěšně vyzkoušena v emulátorech pro mobilní i televizní zařízení, které jsou součástí vývojového prostředí NetBeans (viz obrázek 3.16). Z toho plyne, že teoreticky je možné hru spustit nejen jako desktopovou či webovou aplikaci, ale nic by tedy nemělo bránit tomu, aby fungovala například i na mobilním telefonu či televizi. Jedinou podmínkou je pouze přítomnost běhového prostředí JavaFX ve verzi 1.3 na těchto zařízeních.

Obrázek 3.16: Hra SpaceGame v emulátoru mobilních zařízení

3.4.6 Závěr k aplikaci SpaceGame

Vytvořením aplikace SpaceGame jsem chtěl poukázat na skutečnost, že platforma JavaFX může být bez problému použita pro vývoj webových her. Díky jednoduchosti tvorby animací a používání základních efektů, lze vytvářet graficky bohaté hry. Čtenář této práce může z popsané konstrukce hry snadno získat inspiraci pro rozšíření hry SpaceGame či vytvoření své vlastní. Záběr z aplikace je ukázán na obrázku 3.17.

Obrázek 3.17: JavaFX aplikace SpaceGame

4 Závěr

JavaFX je mladá technologie, kterou ještě pravděpodobně čeká dlouhá cesta, nežli bude více rozšířena a využívána. Nejsilnější zbraní JavaFX je bezpochyby snadné programování grafické stránky aplikací díky výborně navrženému jazyku JavaFX Script. Jeho zvládnutí nepředstavovalo během vytváření této práce větší problém. Rozsáhlé API JavaFX a jeho členění na jednotlivé běhové profily se občas při vyvíjení aplikací stávalo mírně nepřehledné. Na druhou stranu je API JavaFX bohatě vybaveno třídami pro snadnou tvorbu animací a efektů, které lze při vývoji aplikací snadno využívat (Zdroj [21]). Další velkou výhodou je i možnost využívat standardní třídy Java (Zdroj [16]).

Vývojové prostředí NetBeans poskytovalo při vývoji aplikací dostatečný komfort a výrazně ulehčovalo práci například při vkládání jednotlivých tříd do zdrojového kódu a to včetně deklarace potřebných importů. Zklamáním pro mě byl jedině nástroj JavaFX Composer, který pracuje na principu WYSIWYG a neumožňuje manipulaci s již vloženými neviditelnými layout kontejnery typu `VBox`, `HBox`, `Stack` apod. Pro tvorbu aplikací jsem tedy použil standardní editor zdrojového kódu prostředí NetBeans.

Věřím, že aplikace, které jsem v rámci této práce vytvořil, čtenáři dobře poslouží pro počáteční seznámení se s platformou JavaFX a vzbudí zájem o tvorbu RIA aplikací pomocí zajímavého programovacího jazyka JavaFX Script.

Literatura

[1] Adobe Inc. *Adobe* [online]. 2010 [cit. 2011-02-27]. *Flash Player Statistics*. Dostupné z WWW: <http://www.adobe.com/products/player_census/flashplayer/>.

[2] Adobe Inc. *Adobe* [online]. 2009, 07-14-2009 [cit. 2011-01-27]. Rich Internet Applications. Dostupné z WWW: <http://www.adobe.com/resources/business/rich_internet_apps/>.

[3] CLARKE, Jim; CONNORS, Jim; BRUNO, Eric. *JavaFX : Developing Rich Internet Applications*. California, U.S.A. : Addison–Wesley, Pearson Education, 2009. 359 s.

[4] Demerjee Technologies. *Demerjee.com* [online]. 2010 [cit. 2011-01-04]. Countdown timer. Dostupné z WWW: <http://www.demerjee.com/software/android>.

[5] DOEDERLEIN, Osvaldo. *Java.net* [online]. 2010-05-21 [cit. 2011-03-01]. JavaFX's Game of Life. Dostupné z WWW: <http://weblogs.java.net/blog/opinali/archive/2010/05/21/javafxs-game-life>.

[6] GEORGE, Gregory. *The Atari Times* [online]. 2001-09-07 [cit. 2011-01-06]. All About Asteroids. Dostupné z WWW: <http://www.ataritimes.com/article.php?showarticle=174>.

[7] JANOVSKÝ, Dušan. *Jak psát web* [online]. 2001 [cit. 2011-01-18]. Font-size. Dostupné z WWW: <http://www.jakpsatweb.cz/css/font-size.html>.

[8] JavaFX. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2009, last modified on 2011 [cit. 2011-04-28]. Dostupné z WWW: <http://en.wikipedia.org/wiki/JavaFX>.

[9] KALDA, Martin. *Design portál* [online]. 2007-07-04 [cit. 2011-02-16]. Jak vytvořit funkční banner pro reklamní systémy. Dostupné z WWW: <http://www.designportal.cz/navody-tutorialy/jak-vytvorit-banner.html>.

[10] OLIVER, Christopher. *Chris Oliver's Weblog* [online]. 2011 [cit. 2011-02-16]. Dostupné z WWW: <http://blogs.sun.com/chrisoliver/>.

[11] Oracle Corporation. *JavaFX Technology At a Glance* [online]. 2009 [cit. 2010-11-04]. Dostupný z WWW: <http://www.oracle.com/technetwork/java/javafx/overview/index.html>.

[12] Oracle Corporation. *JavaFX Technologies At a Glance* [online]. 2010 [cit. 2010-11-04]. Dostupný z WWW: <http://java.sun.com/javafx/technologies/>.

[13] Oracle Corporation. *JavaFX Documentation Library* [online]. 2010 [cit. 2010-11-04]. Dostupný z WWW: <http://download.oracle.com/javafx/>.

[14] Oracle Corporation. *NetBeans IDE 6.9 Features* [online]. 2010 [cit. 2010-11-04]. Dostupný z WWW: <http://netbeans.org/features/javafx/index.html>.

[15] Oracle Corporation. Oracle Software Downloads [online]. 2010 [cit. 2011-12-28]. *JavaFX 1.3 Migration Guide.* Dostupné z WWW: <http://download.oracle.com/javafx/1.3/tutorials/porting-guide-javafx1-3.html>.

[16] Oracle Corporation. *JavaFX* [online]. 2010 [cit. 2011-01-18]. JavaFX Overview. Dostupné z WWW: <http://javafx.com/about/overview/index.jsp>.

[17] Oracle Corporation. *JavaFX* [online]. 2010 [cit. 2011-01-23]. JavaFX Windows Download. Dostupné z WWW: <http://javafx.com/downloads/windows.jsp>.

[18] Oracle Corporation. JavaFX [online]. 2010 [cit. 2011-04-03]. JavaFX Graphics Files (FXZ, FXD). Dostupné z WWW: <http://javafx.com/docs/integrate-graphic-assets/Fileformats.jsp>.

[19] Oracle Corporation. *JavaFX Documentation Library* [online]. 2010 [cit. 2011-03-05]. Applying CSS to UI Controls. Dostupné z WWW: <http://download.oracle.com/javafx/1.3/tutorials/UIControls/theming.html>.

[20] Oracle Corporation. *NetBeans* [online]. 2010 [cit. 2011-02-12]. NetBeans IDE - JavaFX Development. Dostupné z WWW: <http://netbeans.org/features/javafx/>.

[21] Oracle Corporation. *Oracle Software Downloads* [online]. 2010 [cit. 2010-12-03]. JavaFX 1.3.1 API. Dostupné z WWW: <http://download.oracle.com/docs/cd/E17802_01/javafx/javafx/1.3/docs/api/>.

[22] Oracle Corporation. *Oracle Software Downloads* [online]. 2010 [cit. 2010-12-19]. Learn Java Data Types In JavaFX Language. Dostupné z WWW: <http://download.oracle.com/javafx/1.3/tutorials/core/dataTypes/>.

[23] Oracle Corporation. *Oracle Software Downloads* [online]. 2010 [cit. 2010-12-04]. JavaFX CSS Reference Guide. Dostupné z WWW: <http://download.oracle.com/docs/cd/E17802_01/javafx/javafx/1.3/docs/api/javafx.scene/doc-files/cssref.html>.

[24] Oracle Corporation. *Oracle Software Downloads* [online]. 2010 [cit. 2011-02-15]. Understand Data Binding in JavaFX Language. Dostupné z WWW: <http://download.oracle.com/javafx/1.3/tutorials/core/dataBinding/>.

[25] Oracle Corporation. *JavaFX* [online]. 2010 [cit. 2010-12-02]. How to and General Questions About JavaFX. Dostupné z WWW: <http://javafx.com/faq/>.

[26] Oracle Corporation. *Oracle Software Downloads* [online]. 2010 [cit. 2011-04-01]. Create Sequences in JavaFX Language. Dostupné z WWW: <http://download.oracle.com/javafx/1.3/tutorials/core/sequences/>.

[27] Oracle Corporation. *Oracle Software Downloads* [online]. 2010 [cit. 2011-04-05]. Define Access Modifiers in JavaFX Language. Dostupné z WWW: <http://download.oracle.com/javafx/1.3/tutorials/core/modifiers/>.

[28] Planet JFX [online]. 2010 [cit. 2010-11-04]. Dostupný z WWW: http://jfx.wikia.com/wiki/Main_Page

[29] ROELOFS, Greg. *Libpng.org Portable Network Graphics* [online]. 1996, 2009 [cit. 2011-03-08]. Intro to PNG Features. Dostupné z WWW: <http://www.libpng.org/pub/png/pngintro.html>.

[30] SLAVNIĆ, Saša. *JavaFX RIA Cookbook* [online]. 2010-04-29 [cit. 2011-02-01]. Introduction to JavaFX. Dostupné z WWW: <http://javafx.retrocode.com/2010/05/technology-diagram.html>.

[31] Sun Microsystems. *Oracle Technology Network for Java Developers* [online]. 2000 [cit. 2011-03-13]. Arrays. Dostupné z WWW: <http://java.sun.com/docs/books/jls/second_edition/html/arrays.doc.html>.

[32] Write once, run anywhere. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2004, last modified on 2010 [cit. 2011-04-28]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Write_once,_run_anywhere>.

Příloha

CD nosič

Součástí bakalářské práce je CD obsahující:

- Elektronickou podobu textu ve formátu PDF
- Projektové složky se zdrojovými kódy aplikací *SpaceGame*, *Countdown*, *Currency Converter* a *Banner* pro vývojové prostředí NetBeans