UNIVERZITA PALACKÉHO V OLOMOUCI

PEDAGOGICKÁ FAKULTA

Katedra technické a informační výchovy

Bakalářská práce

Jindřich Kovalčík

Vývoj vzdělávací hry ve Flashi

Vedoucí práce: Mgr. Jan Kubrický, Ph.D.

Olomouc 2015

Prohlašuji, že jsem diplomovou práci zpracoval samostatně pod vedením Mgr. Jana Kubrického, Ph.D. a uvedl všechny použité literární a odborné zdroje a dodržoval zásady vědecké etiky.

V Olomouci dne ……...

Děkuji Mgr. Janu Kubrickému, Ph.D. za pomoc a cenné rady, které mi poskytl při zpracování bakalářské práce.

Obsah

Úvod

O dnešní době se často hovoří jako o době chytrých telefonů a tabletů. Je zřejmé, že dopad těchto zařízení na osobní životy lidí je obrovský. Lidé si za jejich pomoci často plánují celé dny, udržují zdravý životní styl, monitorují své sportovní aktivity, čtou knihy a v neposlední řadě se baví formou různých her. Je naprosto běžné, že žáci již na prvním stupni základní školy vlastní a používají takováto zařízení, avšak v dnešní době je již nepoužívají jen pro osobní potěšení, ale také se záměrem vzdělávání se. S vývojem doby se totiž také inovuje školství, a to mimo jiné v přístupu ke vzdělávání. Ideálním příkladem takové inovace je například projekt „Tablety do škol“, který přímo podporuje implementaci ICT do výuky. To, že se děti již v útlém věku dostávají do kontaktu s chytrými zařízeními, je nevyhnutelné a dle mého názoru se jedná o logický vývoj. Právě proto je důležité, že můžeme, skrze projekty jako „Tablety do škol“, používání chytrých zařízení dětmi řídit a ukázat tak dětem, že takováto zařízení mají obrovský potenciál, který se neomezuje pouze na používání sociálních sítí, chatování a pořizování fotek. Aby bylo však vzdělávání na chytrých zařízeních možné, je na ně potřeba stále dodávat kvalitní vzdělávací obsah v nejrůznějších možných interpretacích.

Hlavním cílem bakalářské práce je vývoj vzdělávací hry. Cílem dílčím je zajistit, aby bylo možné tuto vzdělávací hru rozšířit na nejpoužívanější platformy, tedy jako aplikaci pro mobilní zařízení běžící na systémech Android a iOS, desktopovou aplikaci pro Windows a aplikaci spustitelnou na webu.

V první části práce jsem se zaměřil na rozbor a objasnění použitého vývojového prostředí a technologií použitých při vývoji aplikace. Druhá část je poté přímo zaměřena na vývoji aplikace, od návrhu hry, přes objasnění částí kódů, až po export do spustitelné aplikace.

1 Vývojové prostředí

Před samotnou tvorbou hry, je velice důležité zvolit si vhodné prostředí, ve kterém budeme hru vyvíjet. Tuto volbu ovlivňuje z největší části zvolené cílové zařízení. To znamená, na čem všem si bude moci jedinec naši hru zahrát. Dalo by se říci, že neexistuje pouze jedna správná odpověď. Já jsem pro námi vyvíjenou vzdělávací hru vybral platformu Flash.

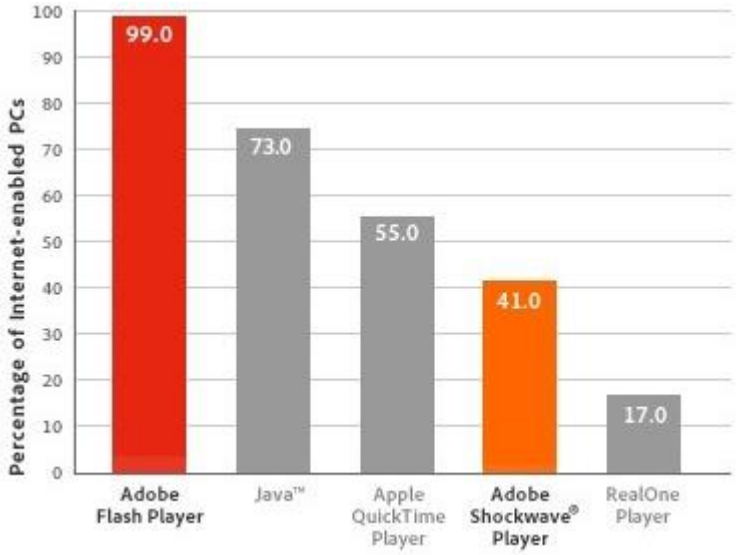
1.1 Adobe Flash

Ideální volbou pro publikování aplikace, za účelem oslovení co možná největší části cílové skupiny uživatelů, je právě platforma Flash. Ta se používá zejména za účelem nasazení bohatého obsahu na web, ať už se jedná o audiovizuální efekty, kompletní vysoce interaktivní webové prezentace, nebo také stále populárnější RIA aplikace (*Rich Internet Application*). Takový obsah se v prohlížeči zobrazuje pomocí příslušného zásuvného modulu, který je dostupný ke stažení a lze ho použít na téměř všech, zaručeně na v dnešní době nejpoužívanějších, webových prohlížečích. Právě tento fakt dělá z Adobe Flash vynikající nástroj pro oslovení jednotlivců z naší cílové skupiny napříč různými platformami. Své tvrzení o vhodnosti použití Adobe Flash pro naši vzdělávací hru, podkládám grafem na obrázku 1, který ukazuje penetraci běhového prostředí Adobe Flash Player v porovnání s konkurencí, na všech počítačích připojených k internetu.

Jedna výjimka zde však existuje, a to v podobě operačního systému iOS, který nepodporuje Adobe Flash ve webovém prohlížeči. To se může jevit jako zásadní problém, jelikož procento populace používající zařízení od firmy Apple je vysoké a stále narůstá. Tento problém je však elegantně řešitelný, a to v podobě tzv. *standalone* aplikace, což znamená samostatnou offline aplikaci, spustitelnou pomocí běhového prostředí Adobe AIR (*Adobe Integrated Runtime*). Vývojem AIR mobilních a desktopových aplikací se však budu konkrétně zabývat až v další kapitole.

Jelikož svět Adobe Flash není v žádném případě malý, je vhodné zmínit a popsat základní části Flash platformy, které ji drží konzistentní a se kterými budu během vývoje naší vzdělávací hry v úzkém spojení a vývoj jako takový by bez nich nebyl možný.

Graf



Obrázek 1: Penetrace Adobe Flash počítači připojenými k internetu[1]

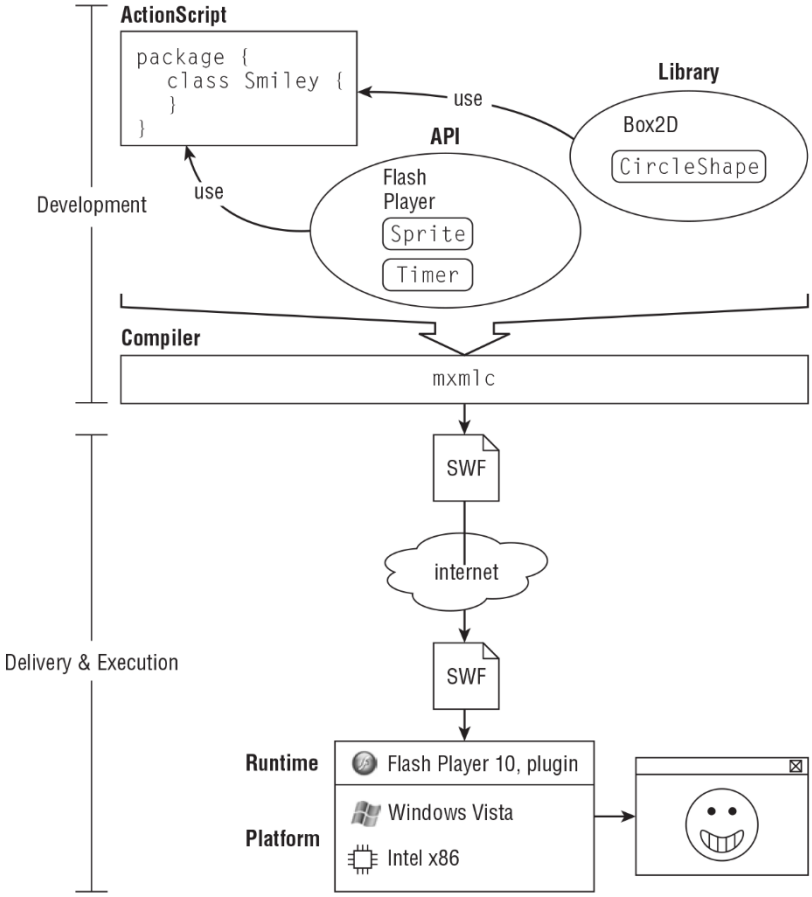
1.1.1 Základní přehled platformy Flash

Je důležité mít povědomí o tom, jak spolu všechny části Flash platformy spolupracují a komunikují. Následující vývojový diagram, který popisuje příklad vývoje ve Flashi se záměrem publikace na internetu, zobrazuje všechny důležité komponenty Flash platformy a v základu ukazuje, jak spolu tyto komponenty spolupracují. Je rozdělen na dvě základní části:

- *Development* (Vývoj)
- *Delivery & Execution* (Publikace a Provedení programu)

[1] Zdroj: http://www.adobe.com/mena_en/products/flashplatformruntimes/statistics.html

Diagram



Obrázek 2: Přehled Flash platformy (Braunstein 2010, s. 4)

Logický průchod diagramem je takový, že programátor píše kód v odpovídajícím programovacím jazyce (*ActionScript 3.0*), kdy využívá funkce, procedury, třídy apod. z API (*Application Programming Interface*) příslušného běhového prostředí, v případě našeho diagramu Adobe Flash Playeru, popřípadě z dalších externích knihoven (*Library*). Hotový kód je poté zpracován příslušným kompilátorem, jehož výstupem je soubor ve formátu SWF. Takový to soubor je pak nahrán na web, tedy doručen koncovému uživateli a na jeho/její platformě spuštěn v příslušném běhovém prostředí, v našem případě ve Flash Playeru, v podobě zásuvného modulu v internetovém prohlížeči. V dalších podkapitolách se budu zabývat jednotlivými komponenty a částmi platformy Flash.

1.1.2 ActionScript 3.0

Flash sám o sobě je v podstatě pouze program pro tvorbu animací, které můžeme pomocí formátu SWF publikovat na internetu. Avšak nejvíce se Flash proslavil díky interaktivnímu obsahu a zejména díky Flash hrám. A právě ActionScript 3.0 je jazykem, který slouží pro programování interaktivního obsahu a je nezbytný, chceme-li vytvořit vysoce dynamický, responzivní a přizpůsobitelný obsah. Jedná se o velice dobře organizovaný, vyspělý, *strongly typed* programovací jazyk, který sdílí hodně ze své syntaxe a metodologie s jinými objektově orientovanými jazyky (Braunstein 2010).

ActionScript 3.0 má podobné některé rysy s JavaScriptem či s Javou, protože jejich jádro je postaveno na ECMAScript specifikacích jazyka. ECMAScript 4 definuje několik pravidel jak psát v ActionScript 3, zahrnující gramatiku a syntaxi, také několik vestavěných datových typů a dovoluje tak vývojářům využívat časté datové typy jako *array*, *Boolean*, *number* či *string* (Anderson 2012).

Historie Flash a ActionScript

Je zřejmé, že společnost Adobe neustále pracuje na různých vylepšeních a nových verzích Flash a ActionScript. Rozebírat všechny dostupné verze a jednotlivé změny jim příslušné, by bylo zbytečné a není to pro naše téma nijak relevantní. Avšak napříč tomu všemu bych rád pro naše potřeby shrnul některé důležité milníky v historii vývoje Flash a Actionscriptu. Jedná se informace popisující evoluci Flash a ActionScriptu k dnešní podobě. Dle mého názoru je tento základ vývoje dobré zmínit, jelikož by bez něho tento projekt nebyl možný.

V prvních třech verzích Flash pro něj nebyl k dispozici žádný programovací jazyk a interaktivita znamenala pouze pár jednoduchých *drag-and-drop* možností z panelu akcí. Flash 4 byl první verzí, která podporovala psaní velice jednoduchého skriptovacího jazyka, který byl neformálně nazýván ActionScript. Až s příchodem Flash 5 se ActionScript vyvinul ještě více a stal se tak oficiálním skriptovacím jazykem. S každou další verzí se možnosti ActionScriptu rozšiřovaly a nabízely interaktivní ovládání animací, textu, zvuků, dat a dalších. V roce 2003 byl představen ActionScript 2.0, jehož možnosti se daly již srovnávat s jazyky jako Java či C#, avšak ze stránky výkonu stále docela výrazně zaostával, a to hlavně z důvodu toho, že byl postaven na skromných

základech předchozí verze. Skromných proto, že Flash, nebyl původně navržen pro tvorbu výkonných aplikací a her, ale právě pro tyto účely ho začala většina vývojářů používat. Na základě toho bylo tedy zřejmé, že další verze toho skriptovacího jazyku, bude muset být navržena od základů. V roce 2006 byl tedy představen ActionScript 3.0, který nabízel výraznou novou funkcionalitu a také opravdu významný nárůst výkonu. Flash CS3 byl první verzí se zahrnutým ActionScriptem 3.0, Flash CS4 k jazyku přidal novou funkcionalitu, jako nové 3D možnosti, nové ovládací prvky animace a nové třídy (*classes*) pro práci s běhovým prostředím Adobe AIR (Florio 2010).

1.1.3 Rozhraní pro programování aplikací – API

Programovací jazyk sám o sobě určuje klíčová slova, syntaxi a gramatiku, ale je to právě rozhraní pro programování aplikací, dále jen API, které udělá většinu práce. Každé běhové prostředí, na které zaměříme náš vývoj, má svoje vlastní API (Braunstein 2010). V našem případě tedy:

- Flash Player runtime -> Flash Player API
- AIR runtime -> AIR API

Pokud při vývoji pracujeme například s datovými typy, pracujeme tak s něčím, co je už zabudované v programovacím jazyku jako výchozí balíček.

Příklad kódu

```
private static const odsazeniPoleX:uint = 135;
var karty:Array = new Array();
```

Chceme-li však pracovat třeba se zobrazením a ještě přidat efekt stínu k nějakému objektu, musíme už sáhnout do balíčku příslušného API, čehož dosáhneme jeho importováním, a to nám dovolí s danou instancí dále pracovat.

Příklad kódu

```
import flash.display.*;
import flash.filters.DropShadowFilter;

var stin:DropShadowFilter = new DropShadowFilter();
hraciKarta.filters = [stin];
```

Je to tedy právě běhové prostředí (*runtime*), které poskytuje ve Flashi většinu zajímavých věcí jako grafika, animace, zvuk apod. Nic z tohoto není zabudované v programovacím jazyku, ale je nám to přístupné přes běhovému prostředí odpovídající API (Braunstein 2010).

1.1.4 Nástroje pro vývoj a kompilátory

V dnešní době jsou na trhu zejména tři nejpoužívanější nástroje pro vývoj Flash aplikací, a to:

- Adobe Flash Builder
- Adobe Flash Professional
- FlashDevelop.

Já jsem pro vývoj naší vzdělávací hry zvolil IDE Adobe Flash Proffesional. Na projekty zaměřené pouze na programování, bez potřeby designerského prostředí, by byl vhodnější Adobe Flash Builder či FlashDevelop. Jelikož však při vývoji pracuji trochu i s grafikou a časovou osou, je k tomuto účelu vhodnější právě tato volba.

Kompilátor

Abychom z planého textu, kterým v podstatě zdrojový kód je, dostali spustitelný soubor, v našem případě např. SWF, musíme jej zkompilovat. K tomu právě slouží kompilátor. Zjednodušeně řečeno, kompilátor přeloží zdrojový kód do jednoduššího jazyka, který může běhové prostředí vykonávat přímo. Hlavní kompilátory potřebné k vývoji jsou převážně integrovány ve vývojových prostředích, což platí i u námi zvoleném prostředí Adobe Flash Professional, kde je integrováno těchto pět hlavním kompilátorů:

- mxmlc – pro kompilování MXML a ActionScript 3.0
- asc – pro kompilování ActionScript 3.0
- fcsh – shell pro opakované kompilování
- adl – pro kompilování a zabalení AIR aplikace (Braunstein 2010).

1.1.5 SWF formát

Jak jsem již zmínil několikrát výše, soubor ve formátu SWF dostaneme po zkompilování ActionScript 3.0 kódu. Jedná se o efektivní, zkomprimovaný binární soubor, který může obsahovat grafiku, animace, zvuk apod. Taktéž samozřejmě obsahuje zkompilovaný ActionScript, což je pro nás velice důležité. Výsledný SWF soubor je samozřejmě spustitelný pomocí běhové prostředí Flash Player ve webovém prohlížeči. Vývoje tzv. *standalone* aplikace se obsáhleji dotkneme až v druhé kapitole, ale rád bych zde zmínil jeden fakt spojený se SWF formátem. Když kompilujeme AIR aplikaci, z *adl* kompilátoru nám vyjde souboru formátu air, ale je důležité si uvědomit, jelikož AIR runtime je nadmnožinou Flash Playeru, tak i soubor formátu air obsahuje ve svém balíčku soubor SWF pro svůj spustitelný ActionScript kód (Braunstein 2010).

1.1.6 Běhové prostředí - Runtime

Běhové prostředí (*runtime),* je prostředí, ve kterém se vykonává příslušný program. *Runtime* poskytuje všechny služby, nezbytné k provedení úkonů, které API slíbilo, že budou dostupné. Ve Flash Player API můžeme např. vytvořit instanci `Camera` pro přístupu k webkameře a je na běhovém prostředí Flash Player, splnit komplexní úkol nalezení připojeného hardware a zajištění video streamu. Stejně tak můžeme programově kreslit nějakou grafiku, ale je opět na běhovém prostředí tuto grafiku renderovat a spolu s operačním systémem ji zobrazit na obrazovce (Braunstein 2010).

Jak jsem již několikrát zmiňoval výše, v našem případě budeme požívat dvě běhová prostředí.

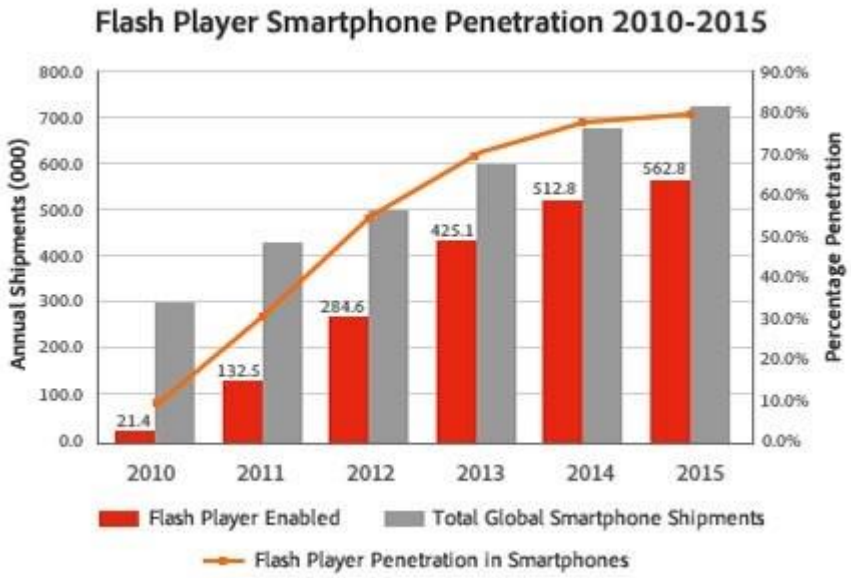
- Adobe Flash Player runtime – pro publikování obsahu napříč webem
- Adobe AIR runtime – pro publikování obsahu jako *standalone* mobilní a desktopové aplikace

2 Samostatná mobilní a desktopová aplikace

Pro úspěšné rozšíření vyvíjené aplikace mezi co největší část naší cílové skupiny, je velice důležité ji doručovat napříč nejrozšířenějšími platformami, a to ne pouze za pomoci webové aplikace, ale také jako samostatnou (*standalone*) offline aplikaci.

To usuzuji i z hlediska praktického, kdy ne pokaždé bude jednotlivec, osloven naší hrou, mít přístup k internetu. Pokud bude mít hru na svém zařízení nainstalovanou jako aplikaci, nebude mu nic bránit v tom, aby si ji z dlouhého času zahrál při cestě autobusem a zároveň se tak něčemu přiučil. Také například pro učitele se může naše hra stát dobrým pomocníkem, bude-li nainstalována jako program na třídním PC s OS Windows 8.1, a tak třeba díky interaktivní tabuli bude moci žáky pomocí hry aktivně zapojit do hodiny. Je tedy zřejmé, že výhody *standalone* aplikací jsou veliké a tak považuji za důležité publikovat naši hru i tímto způsobem. K dosažení tohoto cíle poslouží více než dobře Adobe AIR. A právě na objasnění práce s touto technologií s cílem publikování aplikací pro Android, iOS a Windows se v následujících podkapitolách zaměřím.

Graf



Obrázek 3: Penetrace Adobe Flash mobilními zařízeními[2]

[2] Zdroj: http://www.adobe.com/mena_en/products/flashplatformruntimes/statistics.displayTab2.html

2.1 Adobe AIR

Adobe Integrated Runtime, AIR, je cross-platformní běhové prostředí, které vývojářům umožňuje vytvořit a publikovat aplikace mimo omezení internetového prohlížeče, napříč zařízeními a platformami pro (Anderson 2012):

- Desktopové operační systémy
- Android zařízení, zahrnující smarthphony a tablety
- BlackBerry PlayBook
- iOS zařízení, zahrnující iPhony a iPady
- podporované televizní zařízení

AIR pro chytré mobilní telefony a tablety umožňuje vytvořit aplikace, které můžou být nasazeny stejně jako nativní aplikace napříč mobilními platformami. Na zařízeních s operačním systémem Google Android, které nemají nainstalované Adobe AIR, ale podporují ho, bude uživatel vyzván ke stažení a instalaci běhového prostředí s prvním spuštěním AIR aplikace. Tomuto problému lze však pohotově předejít tím způsobem, že při publikaci naší aplikace v IDE Adobe Flash Professional, přibalíme k aplikaci rovnou i běhové prostředí AIR, které se při prvním spuštění, je-li třeba, samo nainstaluje. Tímto ušetříme uživatele dodatečného stahování a instalování dalšího software, což by mohlo k vést k frustraci a nepoužití aplikace vůbec. Na zařízeních od firmy Apple s operačním systémem iOS, jako je například iPad nebo iPhone, nemůže být AIR nainstalován jako samostatné běhové prostředí, a tak je na tento systém AIR aplikace nasazena jako soběstačný balíček (Anderson 2012).

Obrovskou výhodou pro vývojáře je fakt přenositelnosti kódu. V našem případě kdy vyvíjíme aplikaci jak pro běhové prostředí Flash Player tak i AIR, můžeme pracovat se znalostí toho, že AIR je nadmnožinou Flash Playeru, tudíž aplikace naprogramovaná pro Flash Player se pouhou změnou nastavení cílového běhové prostředí může změnit v soběstačnou AIR aplikaci pro mobilní zařízení.

V následujících podkapitolách se zaměříme na součást vývoje AIR aplikace mimo úkon vlastního programování aplikace. Při vývoji aplikací pro Android či iOS je totiž ještě velice důležitý soubor deskriptoru AIR aplikace.

2.2 Soubor deskriptoru AIR aplikace

Soubor deskriptoru AIR aplikace obsahuje důležité parametry, které jsou příslušným operačním systémem používány pro identifikaci, instalování a spuštění AIR aplikace. S každým projektem, který je zaměřen na AIR mobilní prostředí se automaticky vytvoří šablona souboru deskriptoru. Například novému projektu s názvem pexeso bude vytvořen soubor deskriptoru `pexeso-app.xml` (Anderson 2012).

Soubor deskriptoru AIR aplikace je v podstatě XML soubor skládající se z mnoha prvků, které je potřeba specifikovat pro sestavění mobilní aplikace. Některé parametry v souboru deskriptoru jsou povinné a některé pouze volitelné (Anderson 2012).

V následující tabulce jsou vypsány všechny základní prvky souboru deskriptoru AIR aplikace pro mobilní zařízení.

Tabulka 1: Prvky souboru deskriptoru AIR aplikace (Anderson 2012)

PRVEK	VYUŽITÍ
<application>	Nastaví deklaraci jmenného prostoru AIR. Vyžadováno pro sestavění AIR aplikace.
<id>	Unikátní identifikátor aplikace.
<filename>	Jméno pro Android Package soubor (APK, .apk soubor).
<name>	Nastaví jméno aplikace zobrazené na zařízení.
<versionNumber>	Číslo verze aplikace.
<versionLabel>	Používáno k zobrazení popisku v instalačním dialogu aplikace.
<initialWindow>	Obsahuje vlastnosti pro výchozí vzhled aplikace.
<content>	Nastavení cesty k hlavnímu obsahu .swf souboru aplikace.
<visible>	K nastavení viditelnosti obsahu.
<fullScreen>	Definuje, jestli by aplikace měla použít celou obrazovku zařízení.
<aspectRation>	Specifikuje, jestli je aplikace v horizontální či vertikálním módu zobrazení.
<autoOrients>	Nastavuje, jestli se obsah aplikace automaticky přeorientuje se zařízením.
<supportedProfiles>	Definuje podporovaný profil, který nejlépe sedí typu AIR aplikace.
<icon>	Specifikuje obrázky ikon, sloužící pro spuštění aplikace.

2.2.1 Příklad kódování souboru deskriptoru

Z uvedeného v předchozí kapitole víme, že máme vytvořený soubor deskriptoru `pexeso-app.xml` a díky tabulce 1 již také známe význam jednotlivých prvků tohoto

popisného souboru. Nyní probereme kompletní kódování deskriptoru a objasníme nějaké důležitější části.

V první části se jedná především o hlavičku souboru složenou ze základních atributů, které jsou povinné pro správné nastavení popisného souboru.

Příklad kódu

```
<?xml version=”1.0” encoding=”utf-8” standalone=”no”?>
<application xmlns=”http://ns.adobe.com/air/application/3.4”>
<id>jakykoliv.jedinecny.identifikator</id>
<filename>pexeso</filename>
<name>Hudební Pexeso</name>
<versionNumber>0.9.0</versionNumber>
<versionLabel>0.9.0 (BETA)</versionLabel>
<supportedProfiles>mobileDevice</supportedProfiles>
```

Prvku `<supportedProfiles>` můžou být dodány tři hodnoty:

- `desktop`: AIR aplikace pro dekstop
- `extendedDesktop`: AIR aplikace s podporou nativních procesů API na desktopu
- `mobileDevice`: AIR aplikace pro mobilní zařízení (Anderson 2012).

Příklad kódu

```
<initialWindow>
  <content>notovepexeso.swf</content>
  <visible>true</visible>
  <fullScreen>true</fullScreen>
  <aspectRatio>landscape</aspectRatio>
  <autoOrients>false</autoOrients>
</initialWindow>
```

Ve výše uvedené části kódu popisujeme výchozí vzhled okna. Takto nastavená aplikace poběží v módu plné obrazovky, v horizontálním natočení a bez automatického otáčení podle zařízení. Následující kód odkazuje na obrázky, které budou použity jako ikony.

Příklad kódu

```
<icon>
<image36x36> pexeso36x36.png</image36x36>
```

```
<image48x48> pexeso48x48.png</image48x48>
</icon>
```

Nastavení oprávnění pro Android

Bezpečnostní model operačního systému Android požaduje po každé aplikaci, aby měla zvláštní oprávnění pro použití takových vlastností zařízení, které můžou mít dopad na bezpečnost či soukromí. Zde je seznam jednotlivých oprávnění:

- `android.permission.ACCESS_FINE_LOCATION` – povoluje aplikaci přístup k GPS datům
- `android.permission.CAMERA` – povoluje aplikaci přístup ke kameře
- android.permission.INTERNET – povoluje aplikaci provádět síťové požadavky
- `android.permission.READ_PHONE_STATS` – povoluje aplikaci ztlumit audio, když se objeví příchozí hovor
- `android.permission.RECOR_AUDIO` – povoluje aplikaci přístup k mikrofonu
- `android.permission.WAKE_LOCK` – brání zařízení vstoupit do režimu spánku, zatímco běží aplikace
- `android.permission.DISABLE_KEYGUARD` – zastaví zařízení v zamčení se, zatímco běží aplikace
- `android.permission.WRITE_EXTERNAL_STORAGE` – povoluje aplikaci zapisovat do externí paměťové karty zařízení (Anderson 2012).

Kódové nastavení oprávnění pak může vypadat následovně.

Příklad kódu

```
<android>
   <manifestAdditions>
        <manifest>
              <data>
                    <![CDATA[
<uses-permission android:name=”android.permission.CAMERA/>
<uses-permission android:name=”android.permission.INTERNET/>
                    ]]>
              </data>
        </manifest>
```

```
    </manifestAdditions>
</android>
```

Nastavení možností pro iOS

Stejným způsobem jako v předchozí podkapitole, je potřeba definovat aplikací požadované vlastnosti také u systému iOS. Následující seznam obsahuje nějaké nejběžněji používané vlastnosti a hodnoty k nim náležející. Tedy:

- `UIApplicationExitOnSuspend` - když má tento řetězec hodnotu `<true/>`, aplikace se pouze nepozastaví, ale ukončí se kompletně
- `UIDeviceFamily` – pole hodnot, definující na jakém typu iOS zařízení by aplikace měla běžet. Hodnota 1 specifikuje iPhone a iPod, hodnota 2 zase iPad
- `UIPrerenderedIcon` – pokud je tento řetězec nastaven na hodnotu YES, odstraní se z ikony aplikace výchozí efekt lesku
- `UIRequiredDeviceCapabilities` – pole řetězců, jmenující možnosti zařízení, které jsou požadovány k tomu, aby byla aplikace nainstalována. Možné hodnoty jsou:
    - `accelerometer`
    - `autofocuscamera`
    - `camera-fash`
    - `gps`
    - `location-services`
    - `microphone`
    - `sms`
    - `still-camera`
    - `telophony`
    - `video-camera`
    - `wifi`
- `UIRequiresPersistentWifi` – pokud je tento řetězec nastaven na hodnotu YES, aplikací je vyžadováno Wifi spojení a zařízení udržuje toto spojení aktivní po dobu běhu aplikace
- `UIStatusBarStyle` – specifikuje styl stavového řádku při startu aplikace
    - `UIStatusBarStyleBlackOpaque` – neprůhledný černý
    - `UIStatusBarStyleBlackTranslucent` – průhledný černý
    - `UIStatusBarStyleDefault` – výchozí šedý

Zapsána v kódu pak může tato část vypadat například následovně.

Příklad kódu

```
<iPhone>
      <InfoAdditions>
            <![CDATA[
                  <key>UIDeviceFamily</key>
                  <array> <string>1</string> </array>

                  <key>UIStatusBarStyle</key>
                  <string>UIStatusBarStyleBlackTranslucent</string>

                  <key>UIPrerenderedIcon</key>
                  <string>YES</string>
            ]]>
      </InfoAdditions>
<requestedDisplayResolution>high</requestedDisplayResolution>//vysoke
rozliseni
</iPhone>
```

3 Návrh hry

Před samotným programováním a vývojem hry, je nezbytné položit si tyto otázky:

- o jaký typ hry se bude jednat:
  - arkáda, logická, akční, apod.
- jaký bude její vzdělávací obsah
- kolik hráčů ji bude hrát
- jaká bude cílová skupina

Také je velice důležité si předem navrhnout kompletní logický návrh struktury hry, podle kterého bude probíhat samotné programování. Všemi těmito aspekty se právě v této kapitole budu zabývat.

3.1 Typ hry

Při rozhodování se o typu hry, jsem se především řídil tím, do jaké míry na ni bude možné aplikovat nějaký vzdělávací obsah. Dále bylo pro hru také důležitým aspektem to, že hra bude přístupná na několika různých zařízeních a její ovládání by tedy mělo být velice intuitivní, abych se dopředu vyvaroval jakýmkoliv problémům s kompatibilitou ovládání.

Když jsem tedy začal přemýšlet nad nějakým elementárním typem hry, při níž člověk, aniž by o tom věděl, například trénuje paměť, poznávací schopnosti apod., téměř okamžitě mne napadla logická karetní hra Pexeso. Myšlenka, že by tato hra mohla být vhodná pro modifikaci jako vzdělávací hra na mobilní zařízení, se mi utvrdila v okamžiku, kdy jsem ji hrál se svým tříletým synovcem. Toto konkrétní pexeso mělo na místě obrázků různé automobily, pod kterými byl i jejich název. Vždy, když s ním někdo tuto hru hrál, tak při každém odkrytí karty synovci řekl i příslušný název automobilu. Přibližně po měsíci občasného hraní, mi byl synovec schopen automaticky říkat názvy většiny automobilů, které v průběhu hry odkryl, a to se nejednalo pouze o jednoduché, jednoslovné názvy českého původu.

3.1.1 Herní logika

Pro správné naprogramování hry je velice důležité znát dobře herní algoritmus, což je v reálu ve své podstatě ekvivalentem k herní logice. Rozebráním herní logiky krok po kroku jsem si tedy o něm utvořil základní představu.

Hra je v základu určená pro dva hráče a má velice jednoduchou strukturu. Na hracím stole leží sudý počet karet. Každá karta má k sobě jednu v páru, která má totožný obsah, například obrázek. Všechny karty jsou promíchány a leží obrázkem dolů, takže hráči nevidí jejich obsah. Hráč „A“ si v jednom tahu vybere dvě karty, které otočí obrázkem nahoru. Jsou-li karty shodného obsahu, odstraní se z hracího pole a hráč může pokračovat dalším tahem. Jsou-li však karty nestejné, zanechávají se na stejném místě a opět se otáčí obrázkem dolů. V tom okamžiku přichází na řadu hráč „B“ a odhaluje další karty stejným způsobem. Hra končí v okamžiku vyčištění herního pole a vyhrává ten hráč, který má více správných odhalení.

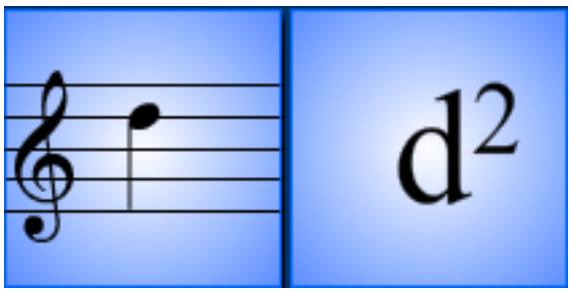
Tuto herní logiku jsem se rozhodl upravit pouze pro jednoho hráče, což bude pro naše účely vhodnější, protože nechceme omezovat hráče možností hry pouze s druhým člověkem. Přidáme tedy jednoduchý systém bodového hodnocení, kdy za odhalení shodných karet budou body připočteny a při nesprávné volbě body odečteny. Cílem hry je tedy vyčištění herního pole a dosažení co možná nejvyššího skóre.

3.2 Přidání vzdělávacího obsahu

Dalším krokem v návrhu hry je zvolení vzdělávacího obsahu. V první řadě jsem vybral obor, ve kterém by hra měla vzdělávat. Jelikož mým druhým oborem na Pedagogické fakultě je Hudební kultura, tak právě toto zaměření jsem cítil jako správnou volbu. Z oboru hudební kultury mne napadalo hned několik možných problematik, které by byly vhodné pro interpretaci v podobě pexesa, jako například hudební nástroje, názvosloví, noty apod. Jelikož bych chtěl touto vzdělávací hrou oslovit hlavně jako cílovou skupinu žáky hudební nauky na základních uměleckých školách, rozhodl jsem se tedy pro podporu elementárních znalostí notové linky a houslového klíče.

Vzdělávací obsah do pexesa jsem přidal snad v možná v nejjednoznačnější formě, kdy jeden symbol je interpretován různým vyzobrazením. Vezměme si tedy jeden totožný pár. Na jedné kartě bude zobrazena nota v notové lince a na kartě druhé bude tatáž nota

napsána svým jménem, včetně příslušné oktávy. Takto budou muset jedinci, hrající tuto hru, nejen slepě hledat noty se stejnou pozicí v notové lince, ale v první řadě budou muset vědět na jakém místě v lince notu hledat, či jaký je název konkrétní noty.



Obrázek 4: Příklad totožného páru karet

3.3 Algoritmus hry

Pojmem algoritmus rozumíme nějaký postup, či sled úkonů, který vede k vyřešení daného problému. Právě tento sled budu v této kapitole popisovat. Na základě předchozí podkapitoly, v níž byla krok po kroku objasněna základní herní logika, zde zpracuji teoretický princip chování programu od spuštění až po jeho konec.

Ještě před popisem samotného algoritmu, shrneme dohromady základní přehled pravidel a rozhodnutím, které jsem pro hru učinil:

- hra bude právě pro jednoho hráče
- na hracím poli bude umístěno třicet karet, tedy patnáct párů karet se shodným obsahem
- vybírání karet bude probíhat kliknutím myši či dotykem na obrazovku
- hráčův postup hrou bude hodnocen bodovým systémem
- hra bude mít tři základní okna:
    - *intro* – okno s názvem hry a s tlačítkem pro start
    - *playgame* – okno ve kterém bude probíhat proces samotného hraní pexesa
    - *outro* – okno, následující po úspěšném vyčištění hracího pole, s gratulací a s dosaženým skóre
- cílovou skupinou jsou žáci přibližně ve věku 1. stupně základní školy.

Nyní už tedy znám všechny potřebné informace k navrhnutí teoretického chování programu. To popíši dalším seznamem, který představuje hierarchický sled úkonů, dle kterého lze naprogramovat základní chování programu. Tedy:

- spuštění hry -> načtení úvodního *intro* okna
- po stlačení tlačítka „ODEJÍT“ uzavření aplikace
- po stlačení tlačítka „HRÁT“ přechod na okno *playgame* se samotnou hrou
- spuštění okna *playgame* -> rozestavení karet na hrací pole obsahem dolů -> čekání na interakci s hráčem
- zvolení karty hráčem
- jedná-li se o první kartu výběru, zobrazíme hráči její obsah a čekáme na výběr druhé karty
- zvolení karty hráčem
- jedná-li se o druhou kartu výběru, zobrazíme hráči její obsah
- porovnáme obsah karet
- ponecháme karty na dvě sekundy otočené, aby měl hráč nějakou šanci zapamatovat si pozici karet
- **jsou-li karty rozdílné**, otáčíme je opět obsahem dolů, proběhne strhnutí skóre a čekáme na další interakci s hráčem
- **jsou-li karty shodné**, odstraníme je z hracího pole, proběhne přičtení skóre a čekáme na další interakci s hráčem
- poté co je hrací pole vyčištěno přechod na okno *outro*
- výpis skóre a gratulace, po pěti sekundové prodlevě přechod na úvodní obrazovku *intro*

Dále je také důležité, uvědomit si fakt, že žáci budou zkoušet hru různými způsoby obejít a najít nějakou skulinku ve funkčnosti. Jelikož hráč je při výběru karty v interakci se hrou pouze pomocí myši, popřípadě dotyku, potřebujeme ošetřit všechny možné stavy, které zde můžou nastat. Naštěstí jich není mnoho a možné způsoby podvedení mne napadají pouze dva, a to:

- hráč při výběru zvolí již vybranou kartu
- při časové prodlevě, kdy hráči zobrazujeme obsah karet, se snaží odkrýt další karty

Nyní už máme v návrhu všechny hlavní a důležité informace o typu a chování hry. Můžeme tedy přejít k další části vývoje vzdělávací hry a to k jejímu samotnému vytváření.

4 Vývoj hry Notové pexeso

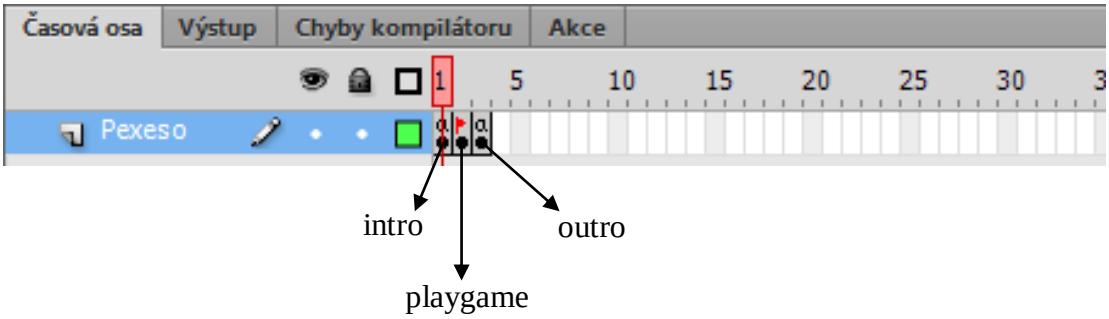
V této kapitole projdeme všemi důležitými aspekty samotného vývoje hry. Ten se neskládá pouze z programování. Nedílnou součástí je taky práce s designérskou a animátorskou částí programu Adobe Flash Professional.

4.1 Základní okna

Jak jsem již uvedl výše, hra se v úplném základu sestává ze tří oken, chcete-li stavů a to:

- *intro*
- *playgame*
- *outro.*

Tohoto dosáhneme nejjednodušeji tak, že samotnou hlavní vrstvu scény s názvem Pexeso, rozdělíme na tři samostatné snímky. Každému tomuto snímku poté přiřadíme jeho jedinečný název, v našem případě to bude právě *intro*, *playgame*, *outro*, což nám umožní se mezi nimi programově pohybovat.



Obrázek 5: Rozdělení vrstvy Pexeso

V následujících podkapitolách se z blízka zaměříme na každý snímek (okno) zvlášť a rozebereme jeho návrh i funkčnost.

4.2 Intro

Tento úvodní snímek má za úkol ve své podstatě pouze přivítat hráče a pomocí tlačítka „HRÁT“ mu umožnit spustit hru, nebo v případě standalone aplikace hru opustit pomocí tlačítka „ODEJÍT“. Vzhled úvodního okna můžeme vidět na obrázku 6.

Obrázek 6: Vzhled úvodního snímku

4.2.1 Funkčnost úvodního snímku

O to, aby vše v úvodním snímku fungovalo tak jak má, se stará script, vázaný přímo na tento konkrétní snímek. Kód je v celku jednoduchý a jako celek to funguje následovně. Aby bylo možné tlačítka ovládat, musíme první vytvořit instance konkrétních objektů. V našem případě máme objekt „tlacitkoHrat“ a jeho instanci `playB` a také samozřejmě objekt „tlacitkoOdejit“ a jeho instanci `leaveB`. Nyní už můžeme k tlačítkům programově přistupovat. V první části skriptu musíme importovat balíček událostí spouštěných myší, a také balíček funkcí nativní aplikace. Dále ještě vytváříme proměnou pro skóre, které budeme zobrazovat na konci hry.

Příklad kódu

```
import flash.events.MouseEvent;
import flash.desktop.NativeApplication;

var score:int;
```

V další části již přidáváme posluchače událostí na jednotlivá tlačítka. Ti se podle kódu chovají tak, že je-li na tlačítko kliknuto, zavolá se tlačítku příslušná funkce, která ošetří tuto událost.

Ač se tak nejeví, také příkaz `stop()` je velice důležitý. Ve flashi se scéna chová jako film, který běží určitou snímkovou frekvencí. Abychom zabránili neustálému přeskakování scény mezi prvním, druhým a třetím, je důležité tímto příkazem „film“ zastavit právě na tomto úvodním snímku.

Příklad kódu

```
playB.addEventListener(MouseEvent.CLICK,startGame);
leaveB.addEventListener(MouseEvent.CLICK,leaveGame);
stop();
```

Následující funkce ošetřuje stisknutí tlačítka „HRÁT“. Příkazem `gotoAndStop` přejde a zastaví se na snímku s jedinečným jménem *playgame*, ve kterémuž poté probíhá samotná hra.

Příklad kódu

```
function startGame(event:MouseEvent)
{
  gotoAndStop("playgame");
}
```

Funkce `leaveGame()` ukončí po zmáčknutí tlačítka „ODEJÍT“ aplikaci.

```
function leaveGame(event:MouseEvent)
{
  NativeApplication.nativeApplication.exit();
}
```

Úspěšně jsme ošetřili funkčnost úvodního snímku a vysvětlili funkci jeho skriptu. V další podkapitole se budu zabývat tím, co vše se bude dít po zmáčknutí tlačítka „HRÁT“.

4.3 Playgame

Dostáváme se ke snímku *playgame*, který je v herní struktuře nejdůležitější, protože obsahuje samotnou hru pexeso. Tento snímek obsahuje pouze jeden objekt, a to objekt pexesoGame typu Filmový klip. Tento objekt poté samozřejmě pracuje s dalšími objekty, jako například s hracími kartami, textovými poli apod., ale to vše je již řešeno programově. Objekt pexesoGame je přímo svázán se souborem pexesoGame.as, který obsahuje již rozsáhlejší skript ovládající celou herní logiku.

4.3.1 Objekt herní karty

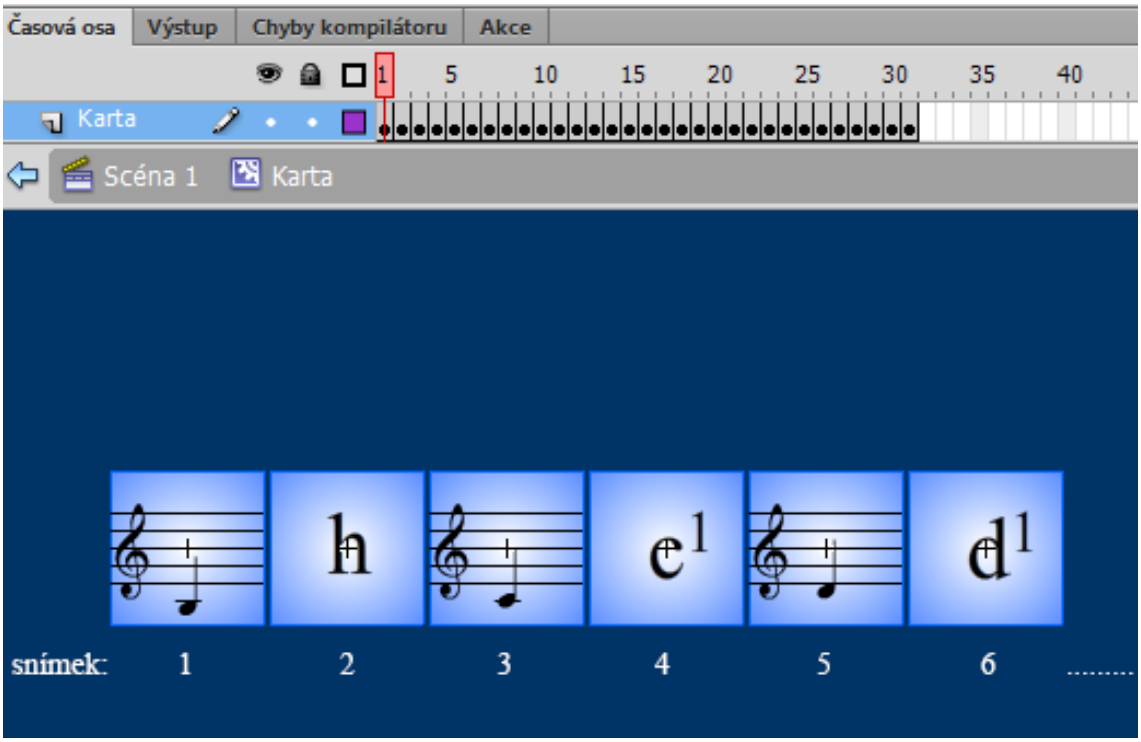
Abychom mohli vůbec začít rozmisťovat karty do hracího pole, musíme mít samozřejmě nějaké vytvořené.

Z logického hlediska existují v podstatě dvě možnosti jak hrací karty vytvořit. První možností by bylo vytvoření několika, v našem případě třiceti jedna, samostatných objektů typu filmový klip. Toto řešení by bylo ovšem velice neúsporné a poměrně pracné z hlediska možných dalších úprav. Druhá možnost, o poznání více elegantní, spočívá opět v práci s nástrojem pro tvorbu animace, tedy s časovou osou. Ta nám tak umožňuje vytvořit pouze jeden objekt s názvem Karta, typu Filmový klip, kterému vytvoříme třicet jedna snímku. Každý tento snímek poté obsahuje grafiku pro konkrétní hrací kartu. Ke konkrétním snímkům přistupujeme pomocí příkazu:

Příklad kódu

```
instanceObjektuKarta.gotoAndStop(3);,
```

kdy v rámci instance tohoto objektu přejdeme na snímek označený číslem v závorce a zastavíme se na něm. Obrázek 7 ukazuje strukturu objektu Karta.



Obrázek 7: Ukázka objektu Karta

V návrhu hry jsem výše uvedl, že hra bude mít třicet karet a nyní tvrdím, že budeme mít třicet jedna snímků. To z toho důvodu, že nesmíme zapomenout také na rub karty. To je na stranu karty, která bude zobrazena, pokud je karta otočena obsahem dolů. Tuto kartu jsem tedy umístil na třicátý první snímek. Jak vypadá, můžeme vidět na následujícím obrázku.

Obrázek 8: Rub hrací karty

Nyní, když je již hotový objekt hrací karty, nepotřebujeme už vytvářet žádné další vizuální herní prvky. Můžeme nyní už konečně přistoupit k programování hlavního herního objektu.

4.3.2 Programování hlavního herního objektu

V první řadě musíme importovat všechny balíčky z API, ze kterých budeme potřebovat konkrétní funkce, třídy apod. Pojďme si nyní shrnout, na základě návrhu hry, co budeme potřebovat. Tedy:

- budeme potřebovat zobrazovat objekty na scéně
- reagovat na události způsobené hráčovou interakcí pomocí myši, či dotyku
- potřebujeme také textové pole pro zobrazování aktuálního skóre
- dále budeme také pracovat s časovačem
- a aby karty nevypadaly moc fádně, bude jim přidán efekt vrhání stínu a záře

Importujeme tedy příslušné balíčky přímo v jazyce ActionScript 3.0, což vypadá následovně.

Příklad kódu

```
import flash.display.*;
import flash.events.Event;
import flash.events.MouseEvent;
import flash.events.TimerEvent;
import flash.utils.Timer;
import flash.filters.GlowFilter;
import flash.filters.DropShadowFilter;
import flash.text.TextField;
```

Další fází v úvodu skriptu je deklarování konstant. Opět z návrhu hry je zřejmé, jaké tyto konstanty budou. Názvy těchto konstant jsem volil tak, aby z jejich názvu byla zřejmá jejich funkce. U některých konstant pouze odůvodním zvolené hodnoty.

Příklad kódu

```
private static const pocetSloupcu:uint = 6;
```

Následující konstanta „mezera“ slouží k sedmi pixelové mezeře mezi kartami. Hodnota 77 je to proto, že k požadované mezeře je potřeba přičíst také délku strany karty, což je 70 obrázkových bodů. Objekt karty se totiž umisťuje na scénu pomocí souřadnic, které udávají její levý vrchní roh. A tak pokud chceme první kartu například na x-ové souřadnici 100 (karta.x = 100) a druhou kartu mít vedle ní se sedmi pixelovou mezerou, bude její x-ová souřadnice 177 (karta2.x = 177).

Příklad kódu

```
private static const mezera:uint = 77;
private static const pocetKaret:uint = 30;
private static const odsazeniPoleX:uint = 135;
private static const odsazeniPoleY:uint = 100;
```

Konstanty „shoda“ a „neshoda“ slouží pro realizaci bodového systému. Konstanta „shoda“ je znatelně vyšší, a to z toho důvodu, že při hře pexeso je pravděpodobnější, hlavně z počátku, nesprávná volba.

Příklad kódu

```
private static const shoda:int = 100;
private static const neshoda:int = -10;
```

V další části kódu deklarujeme všechny proměnné, které v průběhu používáme. Jejich uplatnění vysvětlím později, až přijde na práci s nimi.

Příklad kódu

```
/* PROMENNE */
var karty:Array = new Array();
var pauza:Timer = new Timer(2000,1);
var zbytek:int = 30;
var prvniVyber:Karta;
var druhyVyber:Karta;
var score:int = 0;

/* FILTRY */
var glow:GlowFilter = new GlowFilter();
var stin:DropShadowFilter = new DropShadowFilter();

/* TEXTOVe POLE */
var skoreText:TextField = new TextField();
/* FORMAT PRO TEXTOVE POLE */
var format1:TextFormat = new TextFormat();
```

Pole karet

Se snímky objektu typu Filmový klip, v našem konkrétním případě objektu Karta, není možné nijak fyzicky manipulovat, přesněji ve smyslu přesouvání obsahu jednoho snímku na druhý. To nám značně komplikuje fakt, že při rozestavení karet na hracím poli, je potřebujeme mít náhodně rozloženy a při tom mít zachovánu nějakou jejich

identifikační hodnotu, pro jejich následné porovnávání. Tento problém je řešitelný právě skrze použití datového typu pole (*array*). Vytvořením pole o třiceti jedna prvcích si v podstatě vytvoříme kopii snímků objektu Karta, se kterou můžeme na abstraktní úrovni manipulovat a poté skrze prvky pole přistupovat ke konkrétním snímkům, za pomoci indexů. O vytvoření pole jsme se postarali již výše v sekci deklarace proměnných. Máme tedy pole „karty“, které následujícím cyklem naplníme konkrétními hodnotami.

Příklad kódu

```
for (var i:uint=0; i<pocetKaret; i++)
{
  karty.push(i);
}
```

Použijeme-li příkaz `trace(karty)`, v debugovací konzoli uvidíme obsah celého pole, což je velice užitečné pro kontrolu správného naplnění. Výstup vypadá následovně.

```
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26
,27,28,29
```

Máme tedy pole o třiceti prvcích. Dalším krokem, jak bylo uvedeno výše, je toto pole promíchat.

Fisher-Yates algoritmus

Existuje mnoho algoritmů pro míchání polí. Jeden z nich se nazývá Fisher-Yates. Právě tato moderní varianta míchacího algoritmu je pro naše použití na zamíchání určitého pole karet nejvhodnější (Feronato 2012). Princip tohoto algoritmu si nejlépe vysvětíme na jednodušším příkladu. Řekněme, že máme pole o pěti prvcích.

	A	B	C	D	E
Index:	0	1	2	3	4

Průchod algoritmem je následovný:

- nacházíme se na prvku s indexem 4 -> vygenerujeme náhodné číslo mezi 0 a 4 -> je vygenerováno číslo 2 -> hodnotu prvku s indexem 2 uložíme do pomocné proměnné temp -> hodnotu prvku s indexem 4 přeneseme na prvek s indexem 2 -> do prvku s indexem 4 vložíme hodnotu uloženou v proměnné temp

	A	B	E	D	C
Index:	0	1	2	3	4

- nacházíme se prvku s indexem 3 -> vygenerujeme náhodné číslo mezi 0 a 3 -> je vygenerováno číslo 0 -> hodnotu prvku s indexem 0 uložíme do pomocné proměnné temp -> hodnotu prvku s indexem 3 přeneseme na prvek s indexem 0 -> do prvku s indexem 3 vložíme hodnotu uloženou v proměnné temp

	D	B	C	A	E
Index:	0	1	2	3	4

- nacházíme se na prvku s indexem 2 -> vygenerujeme náhodné číslo mezi 0 a 2 -> je vygenerováno číslo 1 -> hodnotu prvku s indexem 1 uložíme do pomocné proměnné temp -> hodnotu prvku s indexem 2 přeneseme na prvek s indexem 1 -> do prvku s indexem 2 vložíme hodnotu uloženou v proměnné temp

	D	C	B	A	E
Index:	0	1	2	3	4

- nacházíme se na prvku s indexem 1 -> vygenerujeme náhodné číslo mezi 0 a 1 -> je vygenerováno číslo 1 -> hodnota prvku s indexem 1 zůstává na své pozici

Zápis této rutiny do kódu se realizuje pomocí cyklu `for`. Aplikujeme-li výše uvedený algoritmus na náš případ, bude kód vypadat následovně.

Příklad kódu

```
/* PROMICHANI POLE KARET POMOCI FISHER-YATES ALGORITMU */
    for (i=karty.length-1; i>0; i--)
    {
      var nahodnyIndex,temp:uint;
      nahodnyIndex = Math.floor(Math.random() * i);
      temp = karty[nahodnyIndex];
      karty[nahodnyIndex] = karty[i];
      karty[i] = temp;
    }
```

Použijeme-li opět příkaz `trace(karty)`, můžeme zkontrolovat, zdali se nám obsah pole správně promíchal.

```
28,0,7,18,5,29,8,19,11,20,3,1,6,4,25,17,12,21,16,13,2,23,27,14,15,26,1
0,24,9,22
```

Pole je úspěšně promíchané a v dalším kroku se již dostaneme k umisťování herních objektů na scénu.

Jako první zajistíme zobrazení textového pole s aktuálním skórem. To provedeme zavoláním funkce `nastavSkoreText()`, která má na starosti formátu textu pro jeho dobrou viditelnost.

Příklad kódu

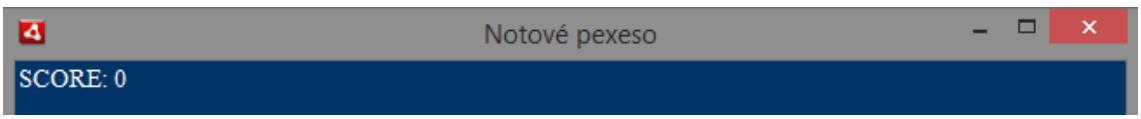
```
function nastavSkoreText(){
  format1.size = 15;
  skoreText.defaultTextFormat = format1;
  skoreText.textColor = 0xFFFFFF;
  addChild(skoreText);
  zobrazSkore();
}
```

Dále tato funkce uvnitř sama sebe volá další funkci `zobrazSkore()`, která má na starosti aktualizování zobrazeného textu dle proměnné `score`.

Příklad kódu

```
function zobrazSkore(){
   skoreText.text = "SCORE: "+ String(score);
}
```

Funkce `String()`převádí v případě uvedeném výše datový typ integer na textový řetězec. Umístění textového pole s průběžným skórem jsem zvolil v levém horním rohu, aby hráče nijak nerušilo, ale v případě potřeby mu bylo k dispozici. Jak to ve skutečnosti na scéně vypadá, můžeme vidět na obrázku 9.



Obrázek 9: Zobrazení skóre na scéně

Následuje přidání všech třiceti karet na scénu, čímž dosáhneme vytvoření kompletního hracího pole. Řešení je realizovatelné opět vhodným použitím cyklu `for`. V každém jednom průchodu cyklem se vytvoří právě jedna instance objektu Karta s názvem hraciKarta a zároveň jí budou přidány všechny jednotlivé vlastnosti. Jelikož je tento úkon komplexnější, ukážeme si nejprve celý cyklus, a poté se zaměříme na jeho jednotlivé části.

Příklad kódu

```
for (i=0; i<pocetKaret; i++)
{
   var hraciKarta:Karta = new Karta();
   hraciKarta.x = odsazeniPoleX+(mezera)*(i%pocetSloupcu);
   hraciKarta.y = odsazeniPoleY+(mezera)*(Math.floor(i/pocetSloupcu));
   hraciKarta.obsah = karty[i];
   hraciKarta.gotoAndStop(pocetKaret+1);
   hraciKarta.filters = [stin];
   hraciKarta.buttonMode = true;
   hraciKarta.addEventListener(MouseEvent.MOUSE_OVER,glowEffect);
   hraciKarta.addEventListener(MouseEvent.MOUSE_OUT,glowEffectOff);
   hraciKarta.addEventListener(MouseEvent.CLICK,klik)
   addChild(hraciKarta);
}
```

Jak nyní vypadá hrací pole, můžeme vidět na následujícím obrázku.

Obrázek 10: Hrací pole osazené všemi kartami

Následující řádky kódu umisťují jednotlivé karty na scénu, a to v posloupnosti, že se zprvu naplní první řádek, poté druhý atd. V prvních šesti průchodech, se díky níže uvedenému matematickému řešení mění pouze x-ová souřadnice karty. Jakmile proměnná „i“ dosáhne hodnoty šest, nebo obecně hodnoty shodné s konstantou „pocetSloupcu“, navýší se i hodnota y-ové souřadnice o hodnotu proměnné „mezera“ a včetně tohoto se po dalších šest průchodů opět nemění. Měnit se bude pouze souřadnice x-ová, která bude kopírovat stále stejných šest hodnot x-ové souřadnice. Takto rozmisťování probíhá až do konce cyklu, kdy je herní pole zaplněno.

Příklad kódu

```
hraciKarta.x = odsazeniPoleX+(mezera)*(i%pocetSloupcu);
hraciKarta.y = odsazeniPoleY+(mezera)*(Math.floor(i/pocetSloupcu));
```

(Feronato 2012)

Nyní se zastavíme u dalšího příkazu, který se dá považovat za klíčový pro propojení našeho pole „karty“ se snímky herního objektu `hraciKarta`.

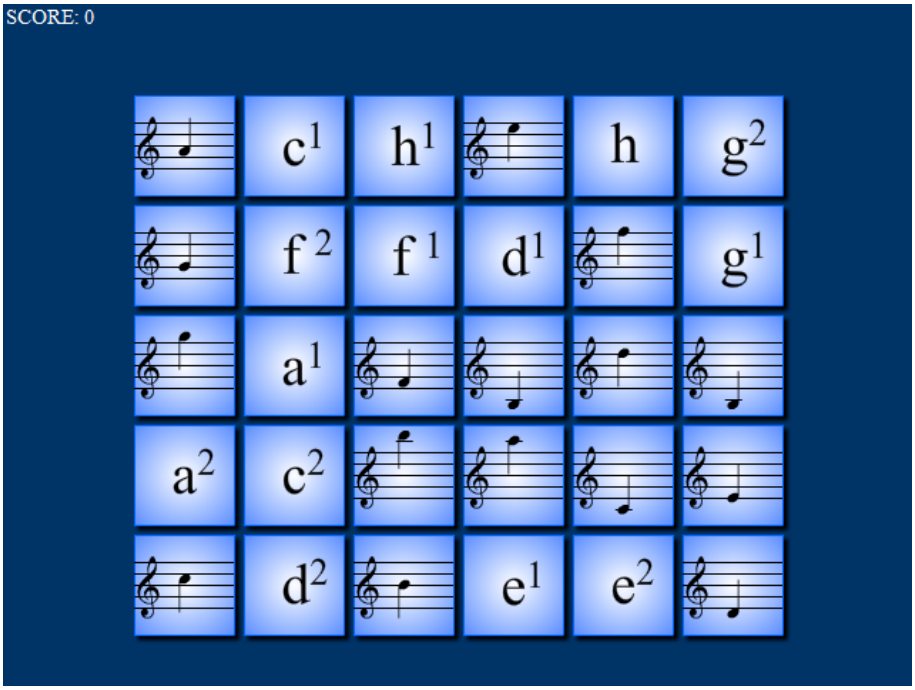
Příklad kódu

```
hraciKarta.obsah = karty[i];
```

Při práci s objektem hraciKarta tkví obrovská výhoda v tom, že této konkrétní kartě, v obecném kontextu všem kartám, můžeme takto jednoduše přiřadit jakoukoliv vlastnost. V našem případě jsem vytvořil vlastnost „obsah“, která bude obsahovat informaci o tom, který snímek má být zobrazen po odhalení karty. Jak je vidět, tuto informaci tam vkládáme ve formě i-tého prvku pole „karty“. Jelikož se nacházíme uvnitř cyklu `for`, bude se hodnota proměnné „i“ s každým průchodem zvyšovat o jednu, takže na každou jednu kartu, připadne právě jeden prvek s pole „karty“. Mějme však na mysli fakt, že pole máme již promíchané.

```
28,0,7,18,5,29,8,19,11,20,3,1,6,4,25,17,12,21,16,13,2,23,27,14,15,26,
10,24,9,22
```

Tím jsme dosáhli toho, že obsah karet přidaných na scénu bude náhodný a rozvržení karet nebude nikdy stejné, což je hlavní podstatou hry pexeso. Pokud bychom karty neotočily obsahem dolů, vypadalo by rozmístěné hrací pole následovně.



Obrázek 11: Ukázka náhodného rozmístění karet

Příklad kódu

```
hraciKarta.gotoAndStop(pocetKaret+1);
hraciKarta.filters = [stin];
hraciKarta.buttonMode = true;
```

První příkaz části kódu, uvedeného výše, otočí každou kartu obsahem dolů. Programově to znamená, že zobrazíme rub karty, který se nachází na třicátém prvním snímku objektu. Právě proto `gotoAndStop(pocetKaret+1)`. V dalším kroku přidáme první vizuální efekt. Každé kartě dodáme efekt vrženého stínu, což dodá hracímu poli trochu 3D efektu. Dále chceme, aby se při najetí myši na kartu, změnil kurzor myši z šipky na ruku. To vyšle hráči zprávu, že na kartu je možné kliknout.

V další části kódu přiřadíme ke každé kartě zrovna tři posluchače událostí, kdy každý z nich čeká na určitý úkon provedený myší, či v analogii, na dotyk.

Příklad kódu

```
hraciKarta.addEventListener(MouseEvent.MOUSE_OVER,glowEffect);
hraciKarta.addEventListener(MouseEvent.MOUSE_OUT,glowEffectOff);
hraciKarta.addEventListener(MouseEvent.CLICK,klik);
```

První dva řádky zajišťují další vizuální efekt. Při najetí kurzorem, či prstem, na kartu chceme, aby hráč viděl, že se něco děje, a že hra na něj reaguje. Toho docílíme přidáním záře kolem karty. Karta bude zářit pouze v okamžiku, kdy se nad ní objevuje kurzor. Při najetí kurzorem na kartu zavoláme funkci `glowEffect`, která kartu „rozsvítí“ a když kurzor z karty sjede, voláme funkci `glowEffectOff`, která se postará o následné „zhasnutí“ karty.

Příklad kódu

```
function glowEffect(event:MouseEvent){
      var aktualniKarta:Karta = (event.currentTarget as Karta);
      glow.color = 0xFFCC00;
      glow.quality = 3;
      aktualniKarta.filters = [stin,glow];
}

function glowEffectOff(event:MouseEvent){
      var aktualniKarta:Karta = (event.currentTarget as Karta);
      aktualniKarta.filters = [stin];
}
```

Efekt záře kolem karty poté vypadá následovně.

Obrázek 12: Ukázka efektu záře

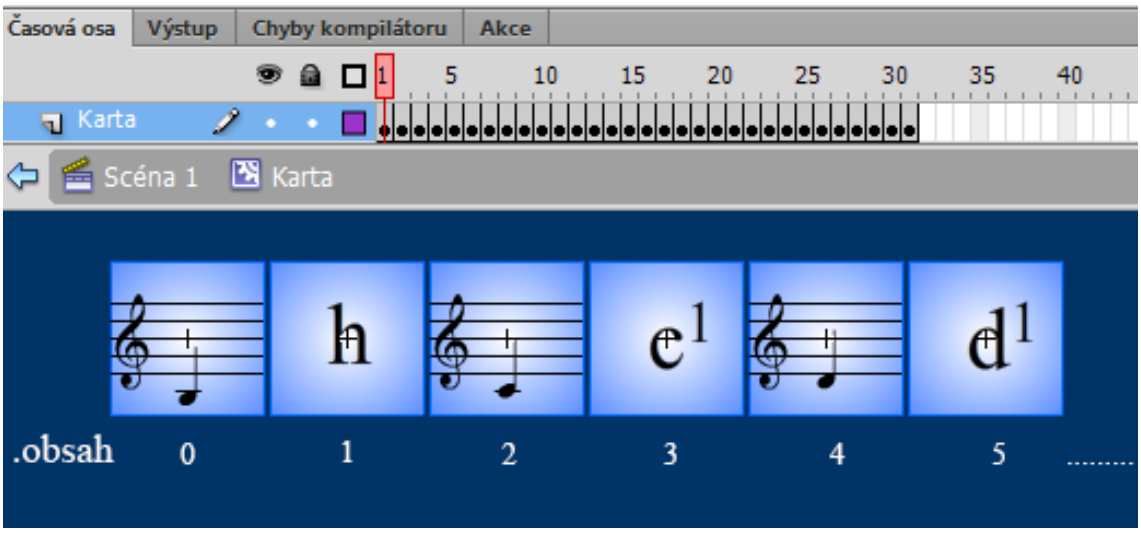
Poslední posluchač události `addEventListener(MouseEvent.CLICK,klik)` ošetřuje již samotný výběr karty.

Je-li na kartu kliknuto, zavolá se funkce „klik“, která má na starost komplexní úkol vyhodnocení. Zdali se jedná teprve o první výběr, nebo už o výběr druhý, nám pomáhají monitorovat proměnné „prvniVyber“ a „druhyVyber“, jak je uvedené v následující části kódu.

Příklad kódu

```
function klik(event:MouseEvent){
	var aktualniVyber:Karta = (event.currentTarget as Karta);
	if (prvniVyber == null) {
		prvniVyber = aktualniVyber;
		prvniVyber.gotoAndStop(prvniVyber.obsah+1);
		prvniVyber.filters = [];
	} else if (prvniVyber == aktualniVyber) {
		trace("druha karta musi byt jina");
	}else if (druhyVyber == null){
		druhyVyber = aktualniVyber;
		druhyVyber.gotoAndStop(druhyVyber.obsah+1);
		druhyVyber.filters = [];
```

V okamžiku kdy hráč vybere druhou kartu, přichází na řadu vyhodnocení, zdali jsou karty shodné. To nám umožňuje právě vlastnost „obsah“, která má informace o vybraném typu karty. Jaký je přesný vztah mezi vlastností obsah a hracími kartami, vidíme na následujícím obrázku.



Obrázek 13: Vlastnost "obsah" a hrací karty

Vybere-li hráč například hned v prvním tahu karty zobrazující notu „h“ (první a druhá karta) a jak je uvedeno v následující části kódu, vydělí u každé karty její „obsah“ dvěma a výsledek zaokrouhlí dolů, výsledek bude shodný a program tak pozná, že hráč vybral dvě shodné karty. Pokud se tedy karty shodují, navýší se skóre, zaznamenáme, že ke kompletnímu vyčištění pole zbývá ještě dvacet osm karet a po uplynutí dvou sekundové pauzy, voláme funkci odstranit. Pokud však karty shodné nebudou, bude sníženo skóre a po uplynutí pauzy, zavolána funkce obnovit.

Příklad kódu

```
if(Math.floor(prvniVyber.obsah/2) == Math.floor(druhyVyber.obsah/2)){
      score += shoda;
      zobrazSkore();
      zbytek -= 2;
      pauza.start();
      pauza.addEventListener(TimerEvent.TIMER_COMPLETE,odstranit);
} else {
      score += neshoda;
      zobrazSkore();
      pauza.start();
      pauza.addEventListener(TimerEvent.TIMER_COMPLETE,obnovit);
}
}
}
```

Hned na začátku funkce „odstranit“, která má za úkol odstranění správně zvolených karet z hracího pole, zjišťujeme, zdali je hrací pole vyčištěno. Pokud tomu tak je, předáme scéně informaci o dosaženém skóre a posouváme se na snímek závěrečný snímek *outro*. V další části následujícího kódu už pak probíhá pouze důsledné odstranění karet ze scény a reset pomocných proměnných.

Příklad kódu

```
function odstranit(e:TimerEvent){
if (zbytek == 0){
MovieClip(root).score = score;
MovieClip(root).gotoAndStop("outro");
}
pauza.removeEventListener(TimerEvent.TIMER_COMPLETE,odstranit);
prvniVyber.removeEventListener(MouseEvent.CLICK,klik);
prvniVyber.removeEventListener(MouseEvent.MOUSE_OVER,glowEffect);
prvniVyber.removeEventListener(MouseEvent.MOUSE_OUT,glowEffectOff);
druhyVyber.removeEventListener(MouseEvent.CLICK,klik);
druhyVyber.removeEventListener(MouseEvent.MOUSE_OVER,glowEffect);
druhyVyber.removeEventListener(MouseEvent.MOUSE_OUT,glowEffectOff);
removeChild(prvniVyber);
removeChild(druhyVyber);
prvniVyber = null;
druhyVyber = null;
}
```

Funkce „obnovit“ je volána v případě, že byly vybrány karty neshodné. Má tedy pouze za úkol obrátit karty opět obsahem dolů a resetovat pomocné proměnné.

Příklad kódu

```
function obnovit(e:TimerEvent){
      pauza.removeEventListener(TimerEvent.TIMER_COMPLETE,obnovit);
      prvniVyber.gotoAndStop(pocetKaret+1);
      druhyVyber.gotoAndStop(pocetKaret+1);
      prvniVyber = null;
      druhyVyber = null;
}
```

4.4 Outro

Tento snímek má hlavně informativní charakter, kdy hráči pogratuluje k vyčištění hracího pole a zobrazí ještě jednou dosažené skóre. Po uplynutí pěti sekundové prodlevy, přejde hra na úvodní snímek *intro*, kde se může hráč sám rozhodnout, zdali bude hrát ještě jednou, nebo hru opustí.

Obrázek 14: Vzhled snímku *outro*

4.4.1 Funkčnost snímku outro

V první části kódu se staráme zejména o zobrazení výsledného skóre, takže o pozici textového pole a dobře čitelný formát textu.

Příklad kódu

```
import flash.text.TextFormat;

var finalScore:TextField = new TextField();
var format1:TextFormat = new TextFormat;
```

```
var pauza:Timer = new Timer(5000,1);

format1.size = 30;
finalScore.defaultTextFormat = format1;
finalScore.x = 300;
finalScore.y = 260;
finalScore.text = String(score);
finalScore.textColor = 0xFFFFFF;
finalScore.defaultTextFormat = new TextFormat();

stop();
addChild(finalScore);
```

V následující části kódu spustíme pětisekundovou prodlevu, po jejíž uplynutí se zavolá funkce „uvod“. Ta ze scény odstraní zobrazené skóre a zajistí přechod na úvodní snímek *intro*.

Příklad kódu

```
pauza.start();
pauza.addEventListener(TimerEvent.TIMER_COMPLETE,uvod);

function uvod(event:TimerEvent){
  removeChild(finalScore);
  gotoAndStop("intro");
}
```

Po přechodu na úvodní snímek *„intro“* jsou uživateli nabídnuty naprosto totožné možnosti volby jako na začátku hry. Tedy zahrát si ještě jednou, či opustit aplikaci.

4.5 Export aplikace

Již dokončená aplikace je nyní složena ze dvou souboru, ke kterým přistupujeme přes vývojové prostředí Adobe Flash Professional. Jsou to následující soubory:

- pexesoGame.as – tento soubor obsahuje skript, který popisuje funkčnost herního objektu *pexesoGame*, umístěném na snímku *playgame*
- pexeso.fla – tento soubor obsahuje veškerou grafickou stránku hry a také drobné skripty řídící funkci jednotlivých snímku *intro, playgame* a *outro*

Aby bylo možné aplikaci publikovat, je nutno ji exportovat do formátů, které přísluší jednotlivým zařízením. Exportovat budeme tedy dohromady čtyřikrát, a to pro:

- Flash Player 11.4 – soubor formátu *.swf pro publikování na webu

- AIR 3.4 for Android – balíček formátu *.apk pro publikování na zařízeních Android
- AIR 3.4 for Desktop – instalační balíček *.exe, který po spuštění nainstaluje aplikaci do PC
- AIR 3.4 for iOS – soubor formátu *.ipa, publikovatelný na iOS zařízení skrze App Store

Pro potřeby této bakalářské práce, nebudu bohužel provádět export pro zařízení iOS. Politika společnosti Apple je taková, že aplikace můžou být instalovány dvěma různými způsoby. Buďto skrze Apple Store, nebo pomocí počítače přes iTunes a datový kabel. Aplikace lze však do požadovaného formátu exportovat pouze s vlastnictvím *mobileprovision* souboru, který můžu získat pouze registrací ve vývojářském účtu společnosti Apple. Za takovýto účet se však platí měsíční poplatky. Fakt, že export pro iOS zařízení nakonec nebude uskutečněn, tedy není zapříčiněn samotným vývojem, kdy aplikace je plně funkční a kdykoliv schopná tohoto exportu, ale hlavně ekonomickým faktorem.

4.6 Publikace

I když po exportu máme již soubory v požadovaných formátech pro určitá zařízení, stále nám zbývá je na tato zařízení dopravit a nainstalovat. V této podkapitole se tedy zaměřím na stručný popis implementace aplikace do zvolené zařízení.

4.6.1 Publikace aplikace na web

Pro publikaci na web je určen soubor formátu SWF. Je samozřejmé, že možností, jak na webové stránky vložit flash aplikaci v tomto formátu je hned několik. Ať už za pomocí JavaScriptu, či cestou jednodušší za pomoci základních html tagů. Následující kódy prezentují nejjednodušší možnosti vložení flash animace na web, za pomoci tagů *object* a *embed*,

Příklad kódu

```
<object>
<embed src="http://sportklubniva.cz/images/notove_pexeso.swf"
width="640" height="480">
</object>
```

nebo další jednoduchou možností pomocí tagu *iframe*.

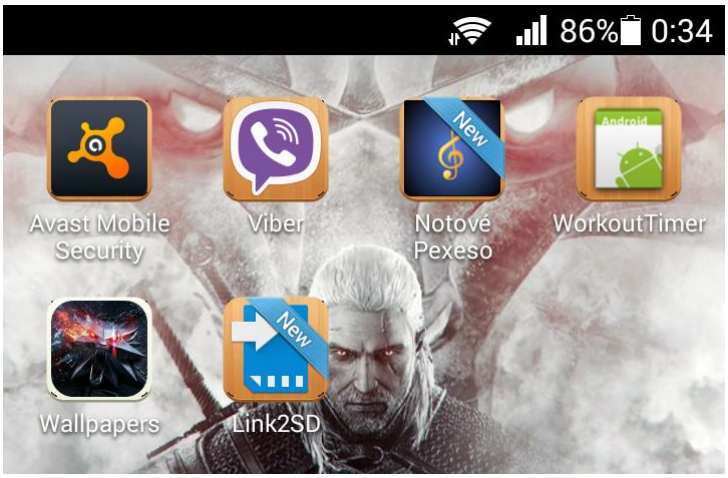
Příklad kódu

```
<iframe src="http://sportklubniva.cz/images/notove_pexeso.swf"
width="640" height="480"></iframe>
```

Je zřejmé, že pro implementaci této aplikace do více komplexnější webové prezentace, by bylo zapotřebí výše uvedené kódy poněkud vylepšit, ale to již není předmětem této práce.

4.6.2 Publikace aplikace na zařízení Android

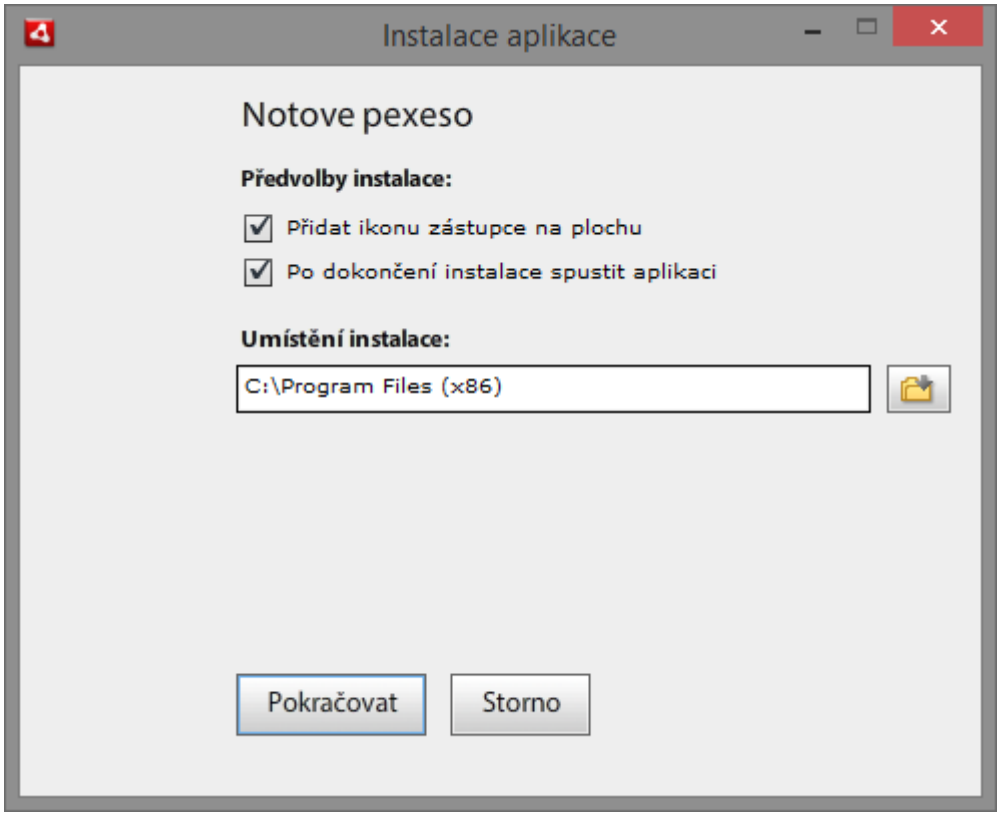
Jeden z prvních úkonů, který je potřeba provést při instalaci aplikace z jiného zdroje než z marketu Google Play, je povolit konkrétnímu zařízení Android instalaci takovýchto aplikací. To je možno provést přes nastavení zařízení v záložce „Zabezpečení“ a zatrhnutí položky „Neznámé zdroje“. Poté už stačí přes datový kabel nahrát balíček .apk na uložiště zařízení, či si ho popřípadě stáhnout z internetu přímo do zařízení. Dále stačí aplikaci už jen nainstalovat. Po instalaci se zobrazí ikona a název aplikace na jedné z ploch zařízení, jak je vidět na obrázku níže.



Obrázek 15: Snímek obrazovky Android zařízení

4.6.3 Publikace aplikace na systému Windows

Publikace na desktop vybavený systémem Windows je zdaleka nejjednodušší. Jelikož jsme provedli export do .exe instalačního souboru, probíhá instalace aplikace tak jak je většina uživatelů zvyklá. Dvojklikem na, v našem případě, soubor NotovePexeso.exe, se spustí instalace aplikace. Instalační průvodce je na obrázku níže.



Obrázek 16: Průvodce instalací

Uživatel má, jako u instalace každého druhého programu, možnosti výběru kam chce aplikaci nainstalovat, zdali chce přidat ikonu na plochu a jestli se má aplikace po ukončení instalace otevřít, či ne. Po dokončení instalace je aplikace připravena k použití. Opětovným spuštěním instalačního .exe souboru můžeme aplikaci jednoduše odinstalovat.

Závěr

Hlavním cílem této bakalářské práce, bylo vyvinout vzdělávací hru. Cílem dílčím bylo zajistit to, aby cílová skupina nebyla omezena pouze na uživatele systému Windows, či zařízení Android, ale aby aplikace byla přístupná všem a na různých platformách.

Teoretická část práce je zaměřena především na nástroje, technologie a procesy, se kterými bylo nutné se seznámit za účelem úspěšného vývoje vzdělávací aplikace. Důležitou roli v teoretické části hrají také kapitoly věnované problematice vývoje aplikací pro více zařízení, a to zejména proto, že tento fakt udával směr vývoje od úplného začátku.

V části praktické je popsán vývoj již konkrétní vzdělávací aplikace, a to od úplného začátku návrhu až po její implementaci na daná zařízení. Z větší části se jedná o logický návrh zvolené hry, který byl následně programově realizován.

Cílů bylo dosaženo hrou Notové pexeso. Jedná se o hru postavenou na principu pexesa s nasazeným vzdělávacím obsahem z oboru Hudební kultury. Hra je publikovatelná na webu, desktopových zařízeních, tabletech a mobilních zařízeních Android. Částečným zaostáním za dílčím cílem je fakt, že ve fázi realizaci, napříč tomu, že bylo vše funkční a nachystané, nedošlo k exportu do instalačního balíčku na zařízení iOS, a to z důvodu spíše ekonomického.

Aplikace samozřejmě není ve své největší formě a má veliký potenciál pro další rozvoj, kdy pomocí jiných elementárních her a různých zábavně vzdělávacích činností, je možné obsáhnout mnohem více z problematiky Hudební kultury.

Referenční seznam

ANDERSON, J.G. *Beginning Flash development for mobile devices*. Hoboken, N.J: Wiley, 2011. ISBN 978-047-0948-156.

BRAUNSTEIN, Roger. *ActionScript 3.0 bible*. 2nd ed. Indianapolis: Wiley, 2010, liii, 953 s. ISBN 978-0-470-52523-4.

FERONATO, Emanuele. *Programujeme hry ve Flashi*. 1. vyd. Brno: Computer Press, 2012, 262 s. ISBN 978-80-251-3697-3.

FLORIO, Chris. *Actionscript 3.0 for Adobe Flash Professional CS5 classroom in a book: the official training workbook from Adobe Systems*. Berkeley: Adobe Press, 2010, viii, 389 s. Classroom in a book. ISBN 978-0-321-70447-4.

ROSENZWEIG, Gary. *ActionScript 3.0 game programming university*. 2nd ed. Indianapolis, Ind.: Que, 2011, xxiii, 567 p. ISBN 07-897-4732-4.

SHUPE, Rich a Zevan ROSSER. *Learning ActionScript 3.0: a beginner's guide*. Sebastopol: O'Reilly, 2008, xviii, 363 s. ISBN 978-0-596-52787-7.

Statistics. *Adobe* [online]. [cit. 2015-06-23]. Dostupné z: http://www.adobe.com/mena_en/products/flashplatformruntimes/statistics.html