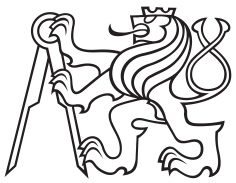
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Přehrávací systém a prototypovací nástroj pro adaptivní hudbu

Tomáš Mazač

2014
Vedoucí práce: Ing. Adam Sporka, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačové grafiky a interakce

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Tomáš Mazač**

Studijní program: Softwarové technologie a management
Obor: Web a multimedia

Název tématu: **Přehrávací systém a prototypovací nástroj pro adaptivní hudbu**

Pokyny pro vypracování:

Seznamte se s problematikou adaptivní hudby v počítačových hrách. Seznamte se s existujícími knihovnami pro jejich implementaci. Navrhněte vlastní model adaptivní hudby pro počítačové hry. Realizujte vlastní přehrávací systém, tj. engine schopný reagovat na vývoj situace ve hře dle předpisu, a jednoduchý nástroj, kterým bude možné ladit tento předpis. Zajistěte, aby bylo možné tento systém integrovat s projekty v Unity Framework. Realizovaný software podrobte příslušným testům. Rozsah práce konzultujte s vedoucím práce.

Seznam odborné literatury:

James A Larson: VoiceXML -- Introduction to Developing Speech Applications, Prentice Hall, 2003
Rich Snoman: Dance Music Manual (Second Edition), Elsevier, 2010, 978-0-2405-2107-7

Vedoucí: Ing. Adam Sporka, Ph.D.

Platnost zadání: do konce letního semestru 2014/2015

prof. Ing. Jiří Žára, CSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 19. 2. 2014

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 21. 5. 2014

.....

Abstrakt / Abstract

Adaptivní hudba je koncept využitý v řadě multimediálních aplikací – nejčastěji v počítačových hrách, kdy se přehrávaná hudba dynamicky mění podle děje ve hře. Cílem této bakalářské práce je navrhnout vlastní model adaptivní hudby a následně tento model implementovat jako přehrávací systém integrovatelný do prostředí Unity. K tomuto přehrávacímu systému je vytvořen prototypovací nástroj s grafickým uživatelským rozhraním, který umožňuje ladit popisy scén ve formátu XML.

Klíčová slova: nelineární, adaptivní, dynamická, interaktivní hudba; přehrávací systém; C++; prototypovací nástroj; Java; Unity.

Adaptive music is a concept used in many multimedia applications—mostly in computer games when playing music is dynamically changed according to what happens in a game. The aim of this bachelor thesis is to design an own adaptive music model and then implement this model as the playing system integrable into Unity evinronment. In addition to this playing system, the prototyping tool with graphical user interface has been created, which allows debugging the scene descriptions in XML format.

Keywords: non-linear, adaptive, dynamic, interactive music; playing system; C++; prototyping tool; Java; Unity.

Title translation: Playback Engine and Prototyping Tool for Adaptive Music

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

1.1 Cíle práce

Tato bakalářská práce si klade za cíl seznámit se s problematikou adaptivní hudby v počítačových hrách (pojmy jako „adaptivní hudba“ a další, které jsou v souvislosti s tímto tématem často uváděny, jsou blíže definovány v sekci 1.2), čímž rozumíme scénický hudební poklad, který se mění na základě toho, co se právě v ději hry odehrává. Také si uvedeme existující knihovny pro implementaci adaptivní hudby.

Pokud tvůrci počítačové hry využívají knihovnu, která umožňuje přehrávat adaptivní hudbu, nemusí ve vlastní logice hry a obsluze uživatelských příkazů řešit navíc ještě rutiny k ovládání hudby, ale pouze využívají několika málo funkcí skrze zpřístupněné API, jímž spouští scény a nastavují hodnotu proměnných, díky kterým se hudba dynamicky mění. Je zapotřebí navrhnout vlastní model adaptivní hudby pro počítačové hry.

Zmíněný návrh modelu bude následně implementován – vznikne engine schopný reagovat na vývoj situace ve hře dle předpisu. Při tvorbě předpisů bude praktické využít služeb prototypovacího nástroje, ve kterém bude možné je ladit. Tento prototypovací nástroj je nutné navrhnout a realizovat.

Jelikož se předpokládá nasazení tohoto přehrávacího systému v reálném projektu vyvíjeném v prostředí Unity, je nutné zajistit, aby ho bylo možné integrovat s projekty v tomto frameworku.

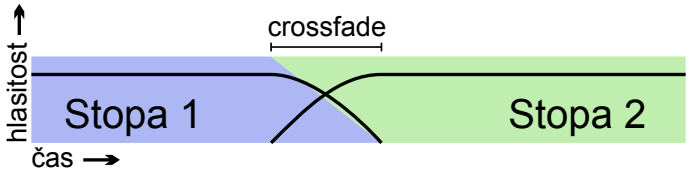
Veškerý realizovaný software bude podroben příslušným testům.

1.2 Základní pojmy

V oblasti zájmu této práce se používá spousta pojmů, které jsou mnohdy zavádějící, nebo přímo nepřesné. Pojďme si je proto jeden po druhém představit a popsat jejich skutečný význam.

1.2.1 Nelineární hudba

Nelineární hudba je označení hudby složené z více samostatných zvukových stop, která využívá techniky jako překrývání hudebních vzorků, postupné snižování nebo zvyšování úrovně zvukového signálu (fade in, fade out, crossfade), náhodné generování zvuků nebo vkládání předem připravených segmentů, aplikování různých filtrů nebo VST apod.



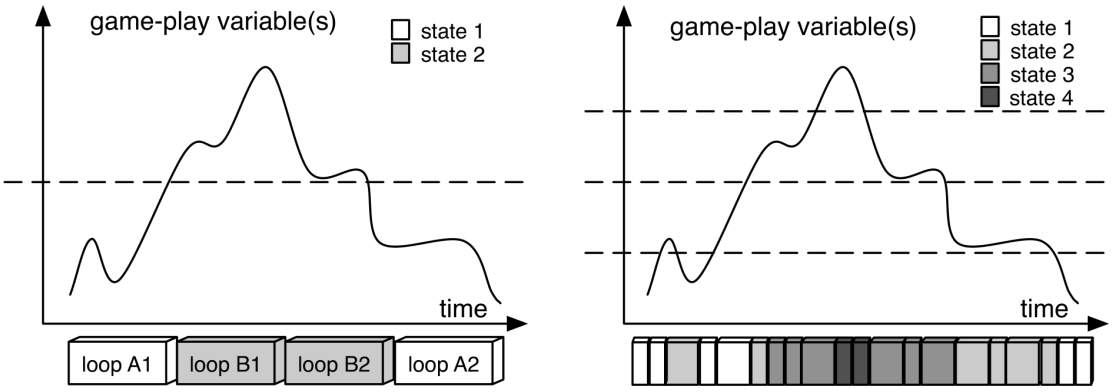
Obrázek 1.1. „Crossfade“ efekt

1.2.2 Adaptivní hudba

Adaptivní hudba je – podle [1] – koncept nelineární přehrávání hudby využitý v celé řadě počítačových her (poprvé v roce 1981 ve hře *Frogger* od společnosti Konami[1])), kdy se díky specifickým událostem mění přehrávaná scénická hudba na pozadí, která významným způsobem (ale přesto intuitivně a nenápadně) dovytváří dramatickou akci, hráčovy prožitky a emoce během hry – uživatel je tak více zapojen do děje. Jako příklad můžeme uvést třeba rozdílnou scénickou hudbu v noci a ve dne, jiný hudební motiv v klidu a při boji, rozdílné nálady v různých místnostech apod. Existují dva základní přístupy, jak takovou hudbu v reálném čase komponovat: *horizontální resekvencování* a *vertikální reorchestrace*.

1.2.3 Horizontální resekvencování

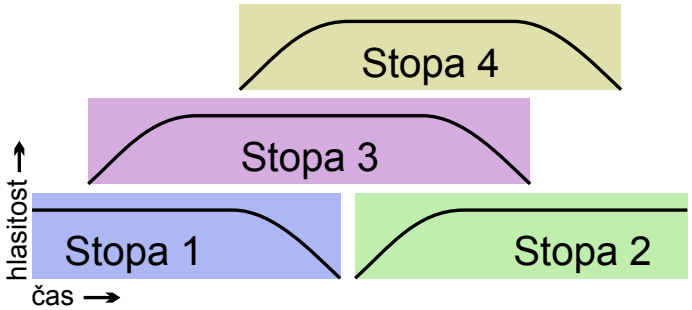
Horizontální resekvencování je metoda, která mění posloupnost přehrávání předem připravených hudebních vzorků vzhledem k hráčově rozhodnutí v příběhu nebo prostředí, ve kterém se právě nachází. Tento princip byl použit v počítačové hře *Monkey Island 2: LeChuck's Revenge*.



Obrázek 1.2. Princip horizontálního resekvencování. Převzato z [2]

1.2.4 Vertikální reorchestrace

Vertikální reorchestrace je poměrně jednoduchá technika adaptivní hudby, kdy je ve smyčce přehráváno několik zvukových vzorků o stejné délce a stejné hodnotě BPM zároveň. Hlasitost jednotlivých stop může být změněna na základě hráčova rozhodnutí, potažmo dějem ve hře. Mezním případem takového nastavování úrovně hlasitosti může být i stav, kdy jeden vzorek hraje maximální možnou hlasitostí a zbylé jsou zcela utlumeny.



Obrázek 1.3. Příklad vrstvení zvukových stop

[1]) `https://www.konami.com/`

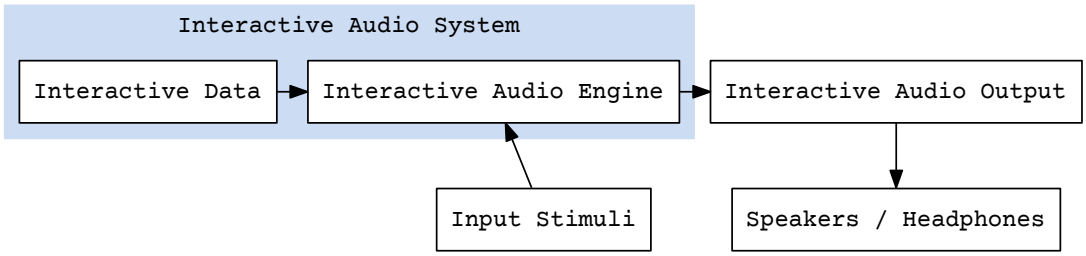
Stijn Frishert se ve svém článku [3] domnívá, že vertikální reorchestrace je pravděpodobně nejstarší způsob provedení flexibilní, adaptivní hudby, která byla použita např. v počítačových hře *Banjo-Kazooie* z roku 1998. Dále uvádí, že výhodou této techniky je její téměř okamžitá reakce, neboť se jako v případě horizontálního resekvencování nemusí čekat na začátek dalšího taktu, dalšího opakování skladby apod. Zároveň však připouští, že okamžitá změna v přehrávané hudbě nemusí být vždy žádoucí. Jako nevýhodu předkládá absenci možnosti přechodů mezi scénami.

1.2.5 Dynamická hudba

Dynamická hudba je v kontextu nelineární hudby synonynem k termínu „adaptivní hudba“. V klasické hudební terminologii však tímto termínem chápeme skladby, které během svého průběhu mění přízvuky (fortissimo, pianissimo atd.), intenzitu, důraz nebo tempo.

1.2.6 Interaktivní hudba

Jak uvádí článek [4], interaktivní hudba je zvuk produkovaný interaktivním audio systémem, což je audio systém navržený tak, aby svým předem stanoveným chováním v reálném čase ovlivnil zvukový výstup v reakci na události, a je složený z interaktivního audio enginu a interaktivních dat. Interaktivním audio enginem rozumíme sadu softwaru (nebo hardwaru), který produkuje interaktivní zvukový výstup na základě vnějších vstupních podnětů a sadě pravidel, podle kterých má být taková hudba vytvořena – interaktivních dat.



Obrázek 1.4. Blokové schéma interaktivního audio systému. Převzato z [4]

Pojem „interaktivní hudba“ je běžně uváděn jako synonymum k termínu „adaptivní hudba“, ale podle [5] takové označení vyjadřuje spíše situaci, kdy uživatel komunikuje s aplikací na základě přehrávané hudby, která se zase mění v závilosti na jeho krocích. Slovo „interaktivní“ tudíž chybně implikuje přímou uživatelskou interakci s hudbou. Příkladem takových aplikací, kdy hráč reaguje na přehrávanou hudbu, můžou být například počítačové hry *Guitar Hero*, *PaRappa the Rapper*, nebo *This Exquisite Music Engine*[1]). Adaptivní hudba je tedy dle Clarka zobecněním hudby interaktivní.

1.2.7 Generická hudba

Pod termínem generická hudba (angl. *generative music*) rozumíme přehrávanou nelineární hudbu, jejíž rytmická část a melodie jsou generovány v reálném čase.

1.2.8 Reaktivní hudba

Reaktivní hudba je generická hudba, která reaguje na uživatelské nastavení různých parametrů nebo na vnější okolnosti; při přehrávání se plynule mění, vyvíjí se. Nejznámější

[1]) `http://www.dinahmoe.com/projects/this-exquisite-music-engine-by-dinahmoe/`

implementace reaktivní hudby na webu [6] vznikly za přispění vizuálního programovacího jazyka Pure Data[1]).

V článku [4] je předneseno tvrzení, že ačkoliv reaktivní hudba reaguje na vnější podněty, nejedná se o hudbu interaktivní. Podle definice interaktivní hudby (viz kapitolu 1.2.6) totiž interaktivní audio systém umožňuje podle změn vstupního chování ovlivnit podobu zvukového výstupu, zatímco reaktivní systém jen přehrává svou hudbu bez jakéholiv přizpůsobení se uživatelským podnětům.

1.3 Existující řešení

1.3.1 Wwise

Produktová řada Wwise od společnosti Audiokinetic[2]) umožňuje celkovou kontrolou nad přehrávanými zvuky – neřeší tedy jen scénickou hudbu, ale i přímé, nepřímé a environmentální zvuky (což odpovídá rozdělení uvedeného v článku [7]). Importuje zvukové soubory, které mají být použity v počítačové hře, poskytuje různé zvukové filtry, umožňuje mix v reálném čase, nabízí možnost definice herních stavů a následnou simulaci. To vše přes grafické uživatelské rozhraní.

Na webové stránce [8] je uvedeno, že Wwise má přímou podporu pro využití ve frameworku Unity Pro (další integrace je možná v enginech Unreal 3, Unreal 4, Orochi 3, Infernal Engine, GameBryo). Je též multiplatformní – společnost Audiokinetic na svém webu [9] tvrdí, že Wwise podporuje následující platformy:

- Android,
- iOS,
- Mac,
- Nintendo 3DS,
- PlayStation 3,
- PlayStation 4,
- PlayStation Vita,
- Wii,
- Wii U,
- Windows (XP, Vista, 7 a 8),
- Windows Phone 8,
- Xbox 360,
- Xbox One.

V ceníku na webové stránce [10] je uvedeno, že cena licence pro komerční projekty je určena celkovým výrobním rozpočtem, a pro nekomerční účely (tj. prototyp hry, vývoj freeware, akademické projekty) je zdarma.

Podle [11] je Wwise použitý např. v těchto počítačových hrách:

- *Assassin's Creed 3*,
- *Assassin's Creed: Brotherhood*,
- *Batman: Arkham City*,
- *Bioshock Infinite*,
- *Dishonored*,
- *Dragon Age II*,

[1]) `http://puredata.info/`
[2]) `http://www.audiokinetic.com/`

- *Dungeon Siege 3*,
- *Disney Epic Mickey*,
- *Ghostbusters the Video Game*,
- *Halo Wars*,
- *Mass Effect 3*,
- *Metal Gear Rising*,
- *Red Faction Armageddon*,
- *Remember Me*,
- *Saints Row: The Third*,
- *Star Wars: The Old Republic*,
- *Total War: Rome II*,
- *Zoo Tycoon*.

V rozhovoru [12] se můžeme dočíst, že v roce 2010 byl Wwise nasazen na přehrávání hudby při divadelním představení *Dom Duardos* (podle předlohy portugalského dramatika a básníka Gila Vicenta). Hudbu zkomponoval skladatel a zvukový designér Pedro Macedo Camacho[1]).

1.3.2 iMUSE

iMUSE byl vyvinut v devadesátých letech Michaelem Landem and Peterem McConnellem ve společnosti LucasArts[2]), která ho využívá ve svých hrách. Je součástí herního enginu SCUMM. Podle [13] byl tento engine poprvé využit v roce 1991 v počítačové hře *Monkey Island 2: LeChuck's Revenge*. iMUSE používá MIDI, jak uvádí [14].

1.3.3 DirectMusic

Dalším zvukovým systémem z devadesátých let je DirectMusic, což je komponenta DirectX API od společnosti Microsoft, která je – podle [15] – dnes označená jako zastaralá. S odkazem na [16], první verze tohoto softwaru byla pod názvem *Interactive Music Architecture* vydána v roce 1996 jako součást frameworku ActiveX[3]). Od února 1999 pak byla představena jako součást DirectX 6.1.

DirectMusic umí načíst a přehrát zvukové soubory MIDI, WAV a vlastní formát programu *DirectMusic Producer*. Dokáže přehrávat i více zdrojových souborů naráz (každý s vlastním časováním a nástroji), používat DLS, lokalizovat zvuky v 3D prostředí, aplikovat některé zvukové efekty (např. reverb[4])) aj.

1.3.4 JET Interactive Music Engine

JET je přehrávač adaptivní hudby pro malé vestavěné systémy (angl. *embedded system*). Podle manuálů [17–18] je nejčastěji používán pro platformu Google Android[5]). JET využívá formáty MIDI a DLS.

1.4 Unity

Unity[6]) je framework určený k tvorbě multimediálních aplikací, nejčastěji pak počítačových her. Při psaní skriptů lze využít jazyky C#, JavaScript, Boo aj. a integrované

[1]) `http://www.musicbypedro.com/`
[2]) `http://www.lucasarts.com/`
[3]) `http://www.microsoft.com/com/tech/activex.asp`
[4]) Simulace dozvuku v uzavřených prostorech.
[5]) `http://www.android.com/`
[6]) `https://unity3d.com/`

IDE MonoDevelop. Prostředí obsahuje řadu nástrojů k tvorbě 2D a 3D obsahu a navíc existuje obchod Asset Store[1]), který nabízí širokou nabídku assetů (zvuků, textur, 3D modelů, pluginů, skriptů, animací, …). Na webové stránce [19] je uvedeno, že Unity umožňuje nasazení na následující platformy:

- Android,
- BlackBerry 10,
- iOS,
- Linux,
- Mac,
- PS3,
- Wii U,
- Windows,
- Windows Phone 8,
- Xbox 360.

Navíc je díky rozšíření Unity Web Player[2]) možné spouštět multimediální aplikace vytvořené v Unity přímo ve webovém prohlížeči. Na operačních systémech Windows (XP, Vista, 7 a 8) a Mac OS X (10.5 nebo novější) jsou v současné době podporované prohlížeče Chrome, Opera, Firefox, Safari a Internet Explorer, jak napovídá [20].

[1]) `https://www.assetstore.unity3d.com/`
[2]) `http://unity3d.com/unity/multiplatform/web`

Kapitola 2
Analýza a návrh řešení

2.1 Návrh modelu adaptivní hudby

V této kapitole určíme základní principy, podle kterých bude fungovat vlastní přehrávací systém adaptivní hudby.

Přehrávací systém musí podporovat přehrávání všech nejpoužívanějších formátů (WAV, FLAC, MP3, …) a měl by umět přehrávat několik zvukových souborů najednou a umožňovat nastavovat jejich vzájemně různou hlasitost – tedy aby implementoval vertikální reorchestraci. Tyto hudební vzorky musí mít stejné BPM, stejný počet čtvrťových not v jednom taktu a stejný počet taktů, po kterých se má skladba začít opakovat. Stopy, které mají zmíněné atributy shodné, se mohou shlukovat do vzorů. Vzor nemusí obsahovat žádné zvukové soubory. Opakující se zvukové soubory musí mít při opakování možnost po posledním taktu doznít (např. dozvuk perkusního zvuku, reverb, ozvěna).

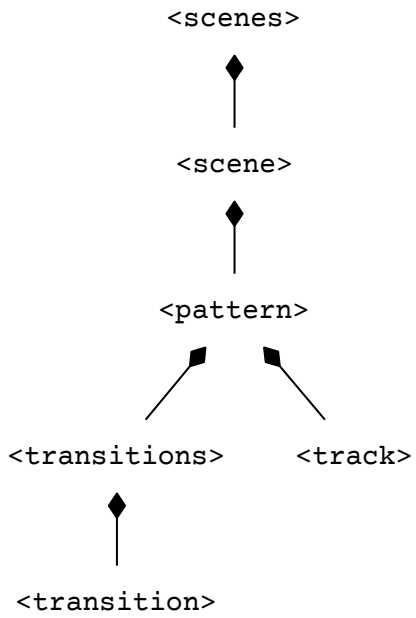
Po splnění určité předem definované podmínky systém přestane přehrávat dosavadní soubory a začne přehrávat nějaké další, což odpovídá principu horizontálního resekvencování. Zmíněné podmínky jsou výrazy skládající se z operandů (čísel a proměnných) a operátorů (aritmetických, logických a relačních), které můžou být vyhodnoceny jako pravda (`true`), nebo nepravda (`false`). Je nutné respektovat prioritu jednotlivých operací a správně vyhodnocovat závorky. Všechny informace potřebné k přehrávání (názvy zvukových souborů, BPM atd.) musí být zapsané v jednom nebo více souborech.

Na základě předchozích požadavků si vymezíme potřebné pojmy, které jsou základními entitami přehrávacího systému:

- **Stopa** (angl. *track*) je zvukový soubor.
- **Přechod** (angl. *transition*) definuje podmínku (ve formě výrazu), po jejímž splnění dojde k přechodu do daného vzoru.
- **Vzor** (angl. *pattern*) je skupina přechodových podmínek a stop, které musejí mít stejné BPM, stejný počet čtvrťových not v jednom taktu a stejný počet taktů, po kterých se má skladba začít opakovat.
- **Scéna** (angl. *scene*) je skupina vzorů. Scén může být definováno i více.
- Každá stopa může mít specifickou **hlasitost** (angl. *volume*), která bude udávat, jak hlasitě se má hudební soubor přehrávat.
- **Předpis** je popis jedné nebo více scén. Popisem rozumíme názvy jednotlivých scén, vzorů a stop, zápis přechodových podmínek, BPM vzorů atd.

2.2 Popis scén

Pro popis scén je nejvhodnější využít značkovací jazyk XML. Základní elementy byly vytvořeny podle pojmů definovaných v kapitole 2.1 (tj. `<scene>`, `<pattern>`, `<transition>` a `<track>`). Přehled všechn elementů a jejich atributů je v tabulce 2.1.



Obrázek 2.1. Taxonomie XML elementů

Tabulka 2.1. Seznam XML elementů a jejich atributů

Element	Atributy
`<scenes>`	
`<scene>`	`name`, `default-pattern`
`<pattern>`	`name`, `bpm`, `measure`, `duration`, `fade-out-duration`
`<transitions>`	
`<transition>`	`pattern`, `when`
`<track>`	`name`, `file`, `volume`

2.2.1 Element `<scenes>`

Element `<scenes>` je kořenový element (angl. *root element*) každého předpisu. Obsahuje alespoň jeden element typu `<scene>` (viz kapitolu 2.2.2). Smysl jeho existence je možnost mít v jednom XML dokumentu více scén, přičemž je samozřejmě možné mít více vstupních XML dokumentů.

Tento element nemá žádné atributy.

2.2.2 Element `<scene>`

Element `<scene>` označuje reálnou scénu ve hře. Musí obsahovat jeden nebo více elementů typu `<pattern>` (viz kapitolu 2.2.3).

- Atribut `name` slouží jako jedinečný identifikátor scény.
- Atribut `default-pattern` odkazuje na název vzoru, který bude přehráván po spuštění této scény.

2.2.3 Element <pattern>

Element `<pattern>` zastupuje vzor, což je skupina přechodových podmínek a stop, které musejí mít stejnou délku a stejné BPM. Může obsahovat jeden element `<transitions>` (viz kapitolu 2.2.4) a elementy typu `<track>` (viz kapitolu 2.2.6). Pokud není zadána žádná stopa, při přehrávání vzoru bude znít jen ticho.

- Atribut `name` slouží jako jedinečný identifikátor vzoru.
- Atribut `bpm` definuje BPM.
- Atribut `duration` udává, po kolika taktech se mají stopy v současném vzoru začít opakovat.
- Atribut `measure` vyjadřuje, kolik čtvrťových not má jeden takt.
- Atribut `fade-out-duration` je nepovinný a definuje počet taktů, během kterých bude skladba hrát „do ztracena“ (tzn. do nulové hlasitosti, po jejímž dosažení bude přehrávání dosavadního vzoru vypnuto) po zapnutí jiného vzoru.

2.2.4 Element <transitions>

Element `<transitions>` seskupuje všechny přechodové podmínky – tj. minimálně jeden element typu `<transition>` (viz kapitolu 2.2.5) – daného vzoru.

Tento element nemá žádné atributy.

2.2.5 Element <transition>

Element `<transition>` pomocí atributu `when` definuje podmínku, po jejímž splnění nastane přechod do vzoru zadaného v atributu `pattern`. Tato podmínková formule je výraz, který může obsahovat názvy proměnných, reálné hodnoty, závorky a aritmetické, relační a logické operátory.

2.2.6 Element <track>

Element `<track>` představuje zvukovou stopu. Hudba v daném souboru má jasně udanou hodnotu BPM, počet čtvrťových not v jednom taktu a počet taktů, po kterých se má skladba začít opakovat. Tyto údaje jsou potřebné k tomu, aby zvuky přesahující délku jednoho taktu v posledním taktu měly možnost přirozeně doznít (např. dozvuk perkusního zvuku).

- Atribut `name` krátkým textovým řetězcem jednoznačně identifikuje stopu.
- Atribut `file` obsahuje cestu ke zvukovému souboru.
- Každá stopa může volitelně obsahovat atribut `volume`, který určuje hlasitostní úroveň přehrávaného souboru. Pokud bude zmíněný element v tomto atributu obsahovat např. `VOLUME`, hlasitost stopy bude závislá na hodnotě proměnné `VOLUME`. V případě, že namísto něj bude kupříkladu hodnota `90`, hladina hlasitosti bude konstantní – hodnota atributu `volume` tedy vyjadřuje **procentuální míru hlasitosti** (v tomto případě 90 %). Atribut může ale obsahovat i funkci (výraz bez logických a relačních operátorů), kupříkladu `MASTERVOLUME * VOLUME`, `BANANA + 13` apod.

Soubor `amen.dtd` (viz přílohu C) obsahuje popis struktury XML v jazyce DTD. Pokud tedy uživatel připojí k XML dokumentu soubor DTD pomocí direktivy

```
<!DOCTYPE scenes SYSTEM "amen.dtd">
```

při psaní mu nějaké rozumné IDE (např. NetBeans[1])) bude napovídat v souladu s definovanou formální gramatikou. Obsah souboru je následující:

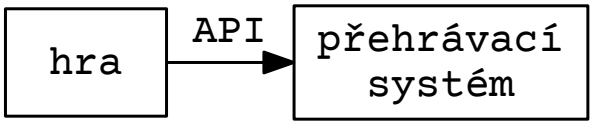
[1]) `https://netbeans.org/`

```
<!ELEMENT scenes (scene+)>

<!ELEMENT scene (pattern+)>

<!ATTLIST scene
  name             ID     #REQUIRED
  default-pattern  IDREF  #REQUIRED
>

<!ELEMENT pattern (transitions?|track*)>

<!ATTLIST pattern
  name               ID     #REQUIRED
  bpm                CDATA  #REQUIRED
  measure            CDATA  #REQUIRED
  duration           CDATA  #REQUIRED
  fade-out-duration  CDATA
>

<!ELEMENT transitions (transition+)>

<!ELEMENT transition>

<!ATTLIST transition
  pattern  IDREF  #REQUIRED
  when     CDATA  #REQUIRED
>

<!ELEMENT track>

<!ATTLIST track
  name    ID     #REQUIRED
  file    CDATA  #REQUIRED
  volume  CDATA
>
```

2.3 Komunikace mezi hrou a enginem

Počítačová hra bude s přehrávacím systémem komunikovat skrze jeho API (tento princip je znázorněn na obr. 2.2), které bude poskytovat jen několik málo jednoduchých funkcí. Uveďme si, jaké funkcionality od enginu požadujeme:

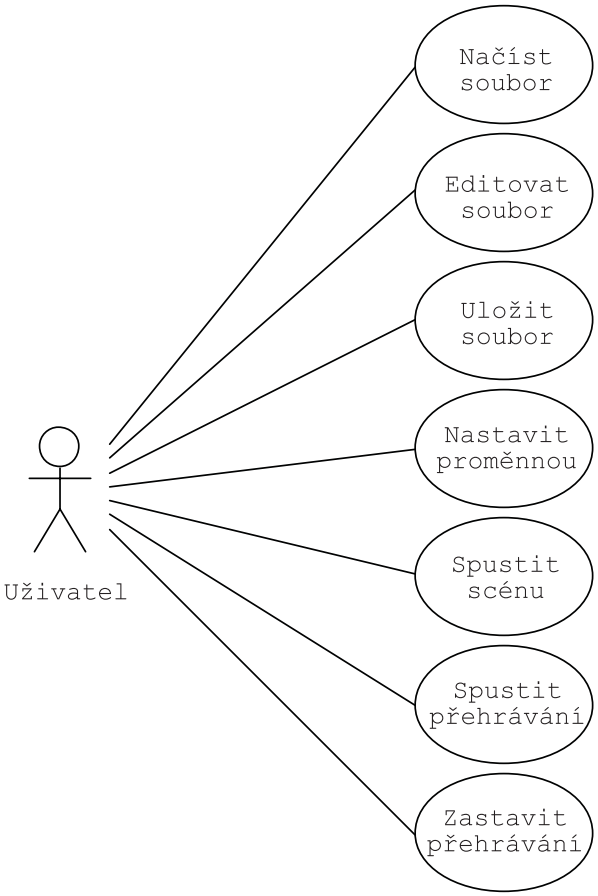
- Načtení předpisu (zadaný XML dokument bude zpracován a uložen do operační paměti počítače),
- spuštění scény (začne přehrávání dané scény),
- zastavení scény (ukončí se přehrávání dané scény),
- nastavení hodnoty proměnné (určené proměnné se přiřadí zadaná hodnota),
- ukončení veškerého přehrávání.



Obrázek 2.2. Princip komunikace mezi hrou a hudebním enginem

2.4 Návrh prototypovacího nástroje

Smyslem tohoto prototypovacího nástroje je vyzkoušet si, jak bude výsledná adaptivní hudba podle napsaných předpisů znít, aniž by knihovna musela být reálně zasazena do nějakého projektu – půjde tedy vlastně o jakýsi hudební debugger. Uživatel si vybere vstupní XML dokumenty a aplikace v nich rozpozná jednotlivé proměnné. Pomocí grafického rozhraní potom bude možné spouštět scény a nastavovat hodnoty jednotlivých proměnných. Diagram případů užití (angl. *use case diagram*) je znázorněn na obr. 2.3.

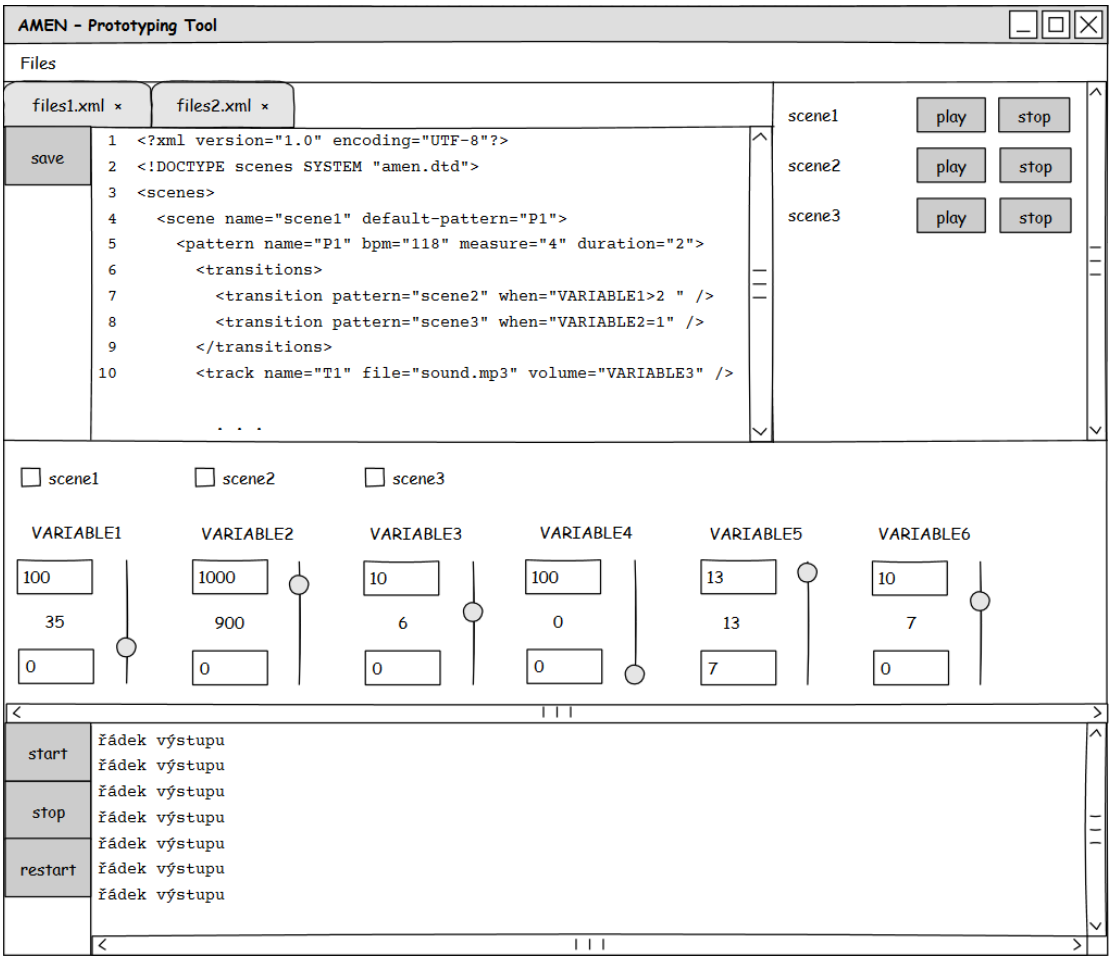


Obrázek 2.3. Diagram případů užití

Grafické uživatelské rozhraní prototypovacího nástroje se bude skládat ze čtyř panelů (viz obr. 2.4):

- Do prvního z nich (vlevo nahoře) se budou do záložek otevírat načtené předpisy, které půjdou editovat a také bude možné provedené změny uložit.
- V panelu vpravo nahoře bude seznam všech scén a tlačítka k přehrání a zastavení jednotlivých scén.

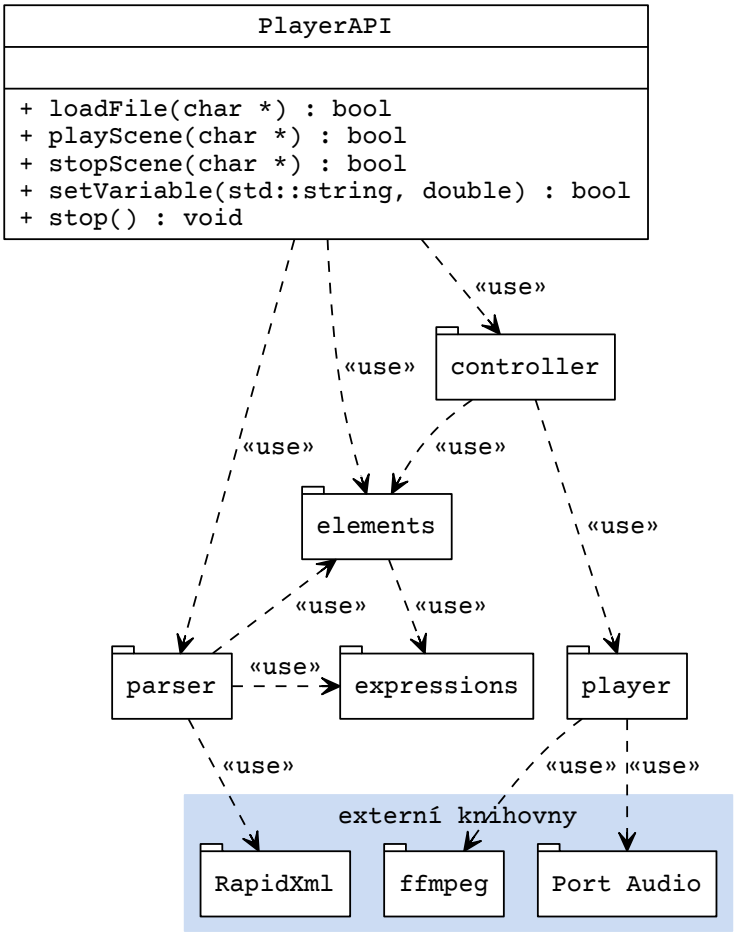
- Uprostřed okna bude panel, ve kterém půjdou na vertikálních jezdcích nastavovat hodnoty proměnných a nastavovat mezní hodnoty jezdce – minimální a maximální možnou hodnotu dané proměnné.
- Spodní část okna patří textovému poli, které bude zobrazovat standardní výstup přehrávacího systému. Také bude obsahovat nezbytná tlačítka pro spuštění a zastavení přehrávání.



Obrázek 2.4. Drátěný model (angl. *wireframe*) GUI

Kapitola 3
Implementace přehrávacího systému

Implementace přehrávacího systému byla realizována v jazyce C++. Zdrojový kód se skládá z pěti balíčků: `player`, `controller`, `expressions`, `elements` a `parser`. Závislosti mezi jednotlivými balíčky jsou znázorněny na obr. 3.1.



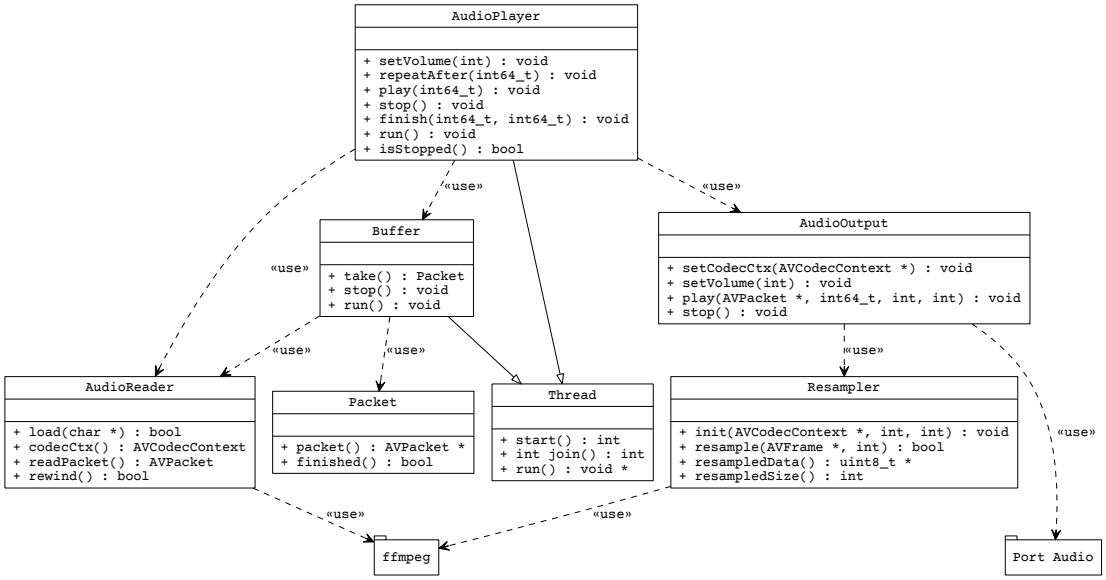
Obrázek 3.1. Závislosti mezi balíčky

3.1 Balíček `player`

Balíček `player` je skupina tříd, která zajišťuje přehrávání zvukových souborů. Nejdůležitější třídou v tomto balíčku je `AudioPlayer`, která za pomoci ostatních tříd řídí celý proces přehrávání jednoho zvukového souboru. Posloupnost operací je následující: Třída `AudioReader` čte – za pomoci knihovny ffmpeg[1]) – zvukový soubor. Data načtená ze souboru se ukládají do vyrovnávací paměti třídy `Buffer` pomocí čtecího vlákna, které

[1]) `http://www.ffmpeg.org/`

zajišťuje asynchnonní načítání a přípravu dat pro přehrání. Další, hrací vlákno (třída `AudioPlayer`), následně data odebírá, zajišťuje časování a předá je třídě `AudioOuput` k dekódování, případnému převzorkování a přehrání pomocí multiplatformní knihovny PortAudio[1]).



Obrázek 3.2. UML diagram tříd balíčku `player`

3.1.1 Čtení zvukových souborů

Existuje mnoho různých formátů pro ukládání zvukových dat (WAV, FLAC, MP3, ...), které lze všechny použít díky knihovně ffmpeg. Hudební soubory obsahují kódovaná data (neplatí v případě WAV), díky čemuž lze dosáhnout velice dobrého poměru komprese dat. K dekódovaní dat se opět využije knihovny ffmpeg. U zvukových souborů nelze např. do souboru MP3 uložit data kódovaná kodekem FLAC. Jinak je tomu u video kontejnerů, které mohou obsahovat několik různých stop s různými kodeky, a i z takových souborů je pak možné přehrávat zvukovou stopu. Vždy ovšem záleží jaké moduly jsou v ffmpeg zakompilované. Od toho se odráží podpora pro čtení formátů (kontejnerů) a jednotlivé kodeky. Knihovna ffmpeg navíc podporuje čtení z mnoha síťových protokolů. Pro použití v této práci je výhodné zkompilovat ffmpeg pouze s podporou čtení z lokálního úložiště a jen pro zvukové souborů a kodeky, čímž se docílí obrovského snížení velikosti této knihovny. Konkrétně se z ffmpeg využívá následujících balíků:

- `avcodec`, který obsahuje kodeky (tzn. kodéry a dekodéry různých formátů),
- `avformat` obsahující popisy, jak číst jednotlivé formáty souboru,
- `avutil`,
- `swresample`, což je knihovna pro softwarové prevzorkování zvuku.

Přehrávací systém umožňuje načítat soubory ve formátech AAC, AC-3, APE, ASF, FLAC, Matroska, MP3, MPC, MOV, MPC8, OGG, TTA, WAV a WV.

Proces čtení ze souboru je vždy stejný, ať už jde o jakýkoliv formát. Nejdříve je nutné inicializovat si tzv. *format context* (datová struktura v C), který nám bude zprostředkovávat přístup ke čtenému souboru. Poté se – za pomoci *format context* – otevře zvukový soubor. V tento moment se dozvíme, zda ffmpeg umí číst daný zvukový formát nebo zda nedošlo k nějaké chybě. Pokud je vše v pořádku, tak je nutné do *format context*

[1]) `http://www.portaudio.com/`

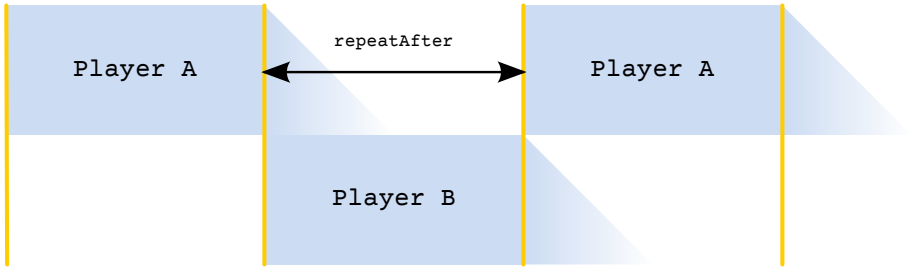
načíst seznam stop ze souboru a vybrat takovou, která je typu audio. Z vybrané audio stopy se získá *codec context* – datová struktura s informacemi o nalezené audio stopě, která obsahuje použitý kodek, přenosovou rychlost (angl. *bitrate*), počet kanálů atd. Najde se kodek, který umí dekódovat nalezenou stopu, a inicializuje se. Pokud kterýkoli z předchozích kroku selže, vrátí se hodnota `false`, jinak `true`, což je znamení, že se muže ze zvukového souboru číst.

Čtení dat z otevřeného souboru se provádí po „balíčcích“ dat (`AVPacket`). Pomocí *format context* se postupně čtou jednotlivé datové balíčky ze souboru. K získání jednoho balíčku je nutné číst data cyklicky – čeká se na takový, který náleží audio streamu, který byl vybrán při otevírání souboru (což by u audio souboru měl být každý). Pro nalezená data se prepočte PTS a délka z interních jednotek knihovny ffmpeg do reálných hodnot – do mikrosekund –, a následně je nalezený paket vrácen do vyšší vrstvy, která o něj požádala.

3.1.2 Přehrávání zvukových souborů

Třída `AudioPlayer` ovládá celý proces přehrávání zvukového souboru. Daný soubor předá třídě `AudioReader`, a pokud ho ta dokáže načíst, tak vyrobí instance třídy `Buffer` k asynchronnímu čtení z `AudioReader` a plnění vyrovnávací paměti, ze které se bude číst (viz kapitolu 3.1.3), a třídy `AudioOutput`, která zajišťuje přehrání paketu (viz kapitolu 3.1.4).

Při přehrávání se opakovaně čtou pakety z vyrovnávací paměti. U každého paketu se ověří, zda neobsahuje příznak `finished` značící konec souboru. Když ještě nenastal konec souboru, tak se paket předá třídě `AudioOutput` k přehrání. V případě, že je nastaven efekt fade out, tak jsou navíc pro `AudioOutput` vypočítány potřebné informace pro tento paket. Za podmínky, že je přijat paket s příznakem o konci souboru, přehrávání pokračuje až po uplynutí doby `repeatAfter`. Důvod je ten, že pro přehrávání jednoho zvukového souboru jsou třídou `PlayingTrack` (viz kapitolu 3.2.3) vytvořeny dvě instance třídy `AudioPlayer` kvůli dozvukům. Tento princip je znázorněn na obr. 3.3.



Obrázek 3.3. Opakované přehrávání jednoho zvukového souboru

3.1.3 Vyrovnávací paměť

Pro plynulé přehrávání hudebních vzorků je nutné pakety přečtené třídou `AudioReader` nejdříve ukládat do vyrovnávací paměti, aby nebylo zapotřebí čekat na načítání dat z pevného disku.

`Buffer` je třída implementující rozhraní vlákna (třída `Thread`), která je spouštěna třídou `AudioPlayer`. Ukládá data do vyrovnávací paměti, aby se dala okamžitě číst zmíněnou třídou. Má pevně definovanou svou velikost a zajišťuje, že nedojde k jeho přetečení a podtečení, k čemuž využívá pole jako kruhový buffer. Jako synchronizační prostředek využívá algoritmus *vzájemné vyloučení*, který pomocí mutexů a tzv. *wait conditions* zajišťuje, že nedojde k nekonzistenci ukládaných dat.

Buffer blokuje volání čtení, dokud není ve frontě alespoň jeden paket. Pokud je fronta plná, blokuje volání zápisu, dokud se neuvolní místo. Díky tomu tak volání funkcí pro čtení a zápis paketů mohou být jednoduché cykly, které nemusí kontrolovat stav bufferu, protože ten jejich volání bude blokovat.

Zápisem do bufferu se zamkne mutex, a pokud je zjištěno, že je vyrovnávací pamět plná, tak je mutex uvolněn pro čtení a čeká na signál, že z bufferu někdo četl, čímž uvolnil místo. V opačném případě uloží paket předaný v parametru funkce a vyvolá signál o zápisu, takže `AudioPlayer` má k dispozici pakety pro další čtení.

Stejně jako v předchozí funkci, i při čtení z bufferu se nejdříve zamkne mutex. Pokud se zjistí, že v bufferu není žádný paket, tak se mutex uvolní pro zápis a čeká se na signál, že bylo do vyrovnávací paměti něco zapsáno – a je tedy co číst. V opačném případě (tj. když buffer není prázdný) se odebere paket, vyvolá se signál o čtení, kterým se dá najevo, že se uvolnilo místo, a nakonec funkce vrátí vybraný paket.

Plnění bufferu se děje ve vlákně, které přečte paket ze třídy `AudioReader`. Když dostane paket, tak ho pouze zabalí do vlastní třídy `Packet`. Když se zjistí, že nastal konec souboru, tak je vyroben objekt `Packet` s příznakem `finished`, který slouží jako značka o konci, a ukazatel format contextu se posune na začátek souboru, aby se poté mohla plnit vyrovnávací paměť dalšímy daty k přehrání. Na konci cyklu se do fronty zapíše vytvořený paket, který zajišťuje ochranu proti přetečení sám, díky čemuž není nutné dělat nějaké pauzy.

3.1.4 Zvukový výstup

Třída `AudioOutput` je na samotném konci řetězce průběhu přehrávání zvukového souboru. Knihovna Port Audio[1]) zajišťuje přehrávání „surových“ dat určených pro vyslání do zvukové karty. Pomocí zmíněné knihovny je v počítači nalezeno zařízení pro přehrávání zvuku, ke kterému se otevře výstupní proud zvukové karty počítače s parametry zjištěnými díky *codec context*. Je nutné ověřit, s jakými parametry byl výstupní proud doopravdy otevřen, protože parametry zadané při otevření nemusí výstupní zařízení zvládnout (v dosavadní formě) přehrát. Získané informace se předají třídě `Resampler`, aby věděla, co přítomné zařízení dokáže přehrát.

Pakety k přehrání jsou vybírány z vyrovnávací paměti. Data z tohoto paketu jsou zakódovaná, takže je nutné je dekódovat pomocí knihovny ffmpeg a *codec context* ze třídy `AudioReader`. Dekódovaná data nemusí mít správný formát (viz kapitolu 3.1.5), proto je výstup dekodéru předán třídě `Resampler`, který vrátí ukazatel na – pokud to je nutné – převzorkovaná data, nebo na originální data z dekódovaného paketu. Po nastavení hlasitosti (v případě aplikace efektu fade out je vypočítán sestupný pokles) jsou data předána knihovně Port Audio, která začne blokovat aktuální vlákno programu, dokud nebude potřebovat další data z vyrovnávací paměti.

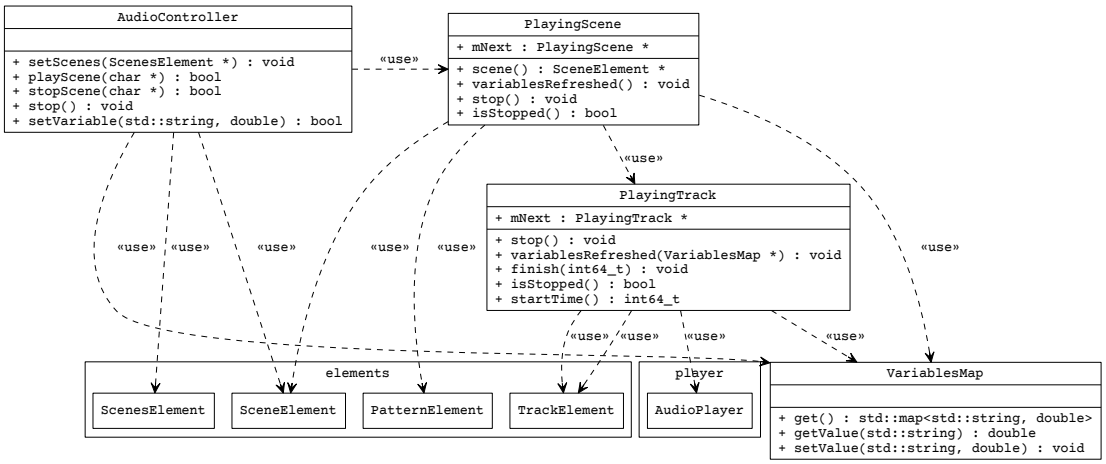
3.1.5 Převzorkování

Po otevření výstupního proudu zvukové karty počítače může být zjištěno, že tento proud nemá shodné parametry jako dekódovaný zvukový soubor určený k přehrání. Těmito parametry rozumíme počet kanálů, vzorkovací frekvenci (angl. *sample rate*) a rozlišení (angl. *sample format*). Pokud parametry nejsou totožné, vytvoří se *resampler context* a přijatá data budou převzorkována. V opačném případě (tj. pokud budou parametry shodné) vrátí vyšší vrstvě třída `Resampler` ukazatel na původní data.

[1]) `http://www.portaudio.com/`

3.2 Balíček controller

Balíček `controller` řídí přehrávání scén, vzorů a zvukových stop, spuštění a zastavení přehrávání scén, přepínání vzorů a řeší správu nastavených proměnných. Jeho role připomíná dirigenta v symfonickém orchestru, jen namísto hudebníků řídí přehrávání hudebních vzorků. Taktéž se nedrží notového zápisu v partituře, nýbrž předpisu ve formátu XML a v něm definovaných přechodových podmínek.



Obrázek 3.4. UML diagram tříd balíčku `controller`

3.2.1 Přehrávání scén

Třída `AudioController` ovládá spuštění a ukončení přehrávání scén, které mu předá třída `PlayerAPI`. `PlayerAPI` načte scény pomocí třídy `Compositor`, a pokud při zpracování vstupního XML dokumentu nedojde k chybě, tak nalezené scény poskytne třídě `AudioController`.

Když `AudioController` dostane příkaz ke spuštění scény, nejdříve se ji podle jejího jedinečného názvu pokusí nalézt v seznamu scén. Pokud daná scéna není nalezena, je na požadavek k přehrání scény vrácena negativní odpověď. V případě, že je nalezena, je do interního seznamu přidán element `PlayingScene`, který inicializuje přehrávání scény.

Třída `AudioController` umožňuje zastavit zadanou scénu, nebo všechny scény najednou a úplně tak vypnout činnost přehrávacího systému.

Struktura použitá pro udržování aktuálního kontextu proměnných je implementována ve třídě `VariablesMap`. Při nastavení hodnoty proměnné se nejdříve ověří, zda došlo ke změně hodnoty zadané proměnné – zda je nutno do paměti přidat hodnotu novou, nebo změnit stávající a vyvolat událost o změně na všech scénách.

3.2.2 Přehrávání vzorů

Každá scéna musí mít definovaný implicitní vzor, který je automaticky přehráván po jejím spuštění. Třída `PlayingScene` tento vzor při svém vytvoření načte a předá k přehrávání.

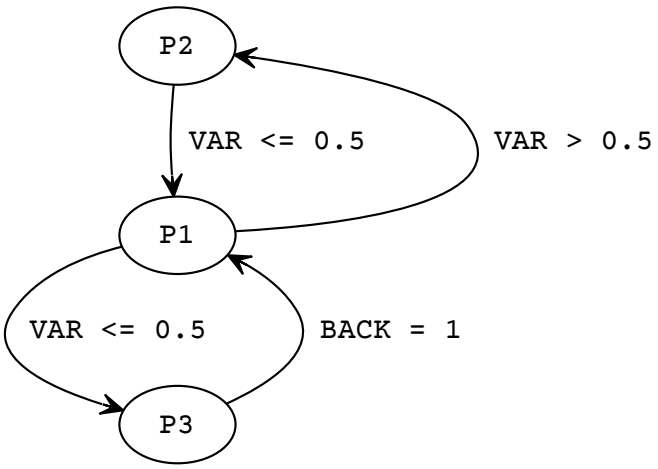
Když `AudioController` dá třídě `PlayingScene` vědět o proběhlé změně v mapě proměnných, přepočítá se hlasitost všech hrajících stop. Také vezme aktuální vzor a ověří, zda není splněna některá z přechodových podmínek – pokud ano, provede se skok na nový vzor.

Jak již bylo řečeno; při nastavení nové hodnoty proměnné se vyšle všem prvkům scény signál o změně mapy proměnných, takže nastane nové vyhodnocování podmínkových formulí. Vzory a přechodové podmínky předpisu tvoří orientovaný graf, přičemž

vzory jsou jeho uzly a přechodové podmínky orientované hrany. Ty můžou vytvořit cykly, jak názorně ukazuje scéna `corridors` z ukázkového předpisu `corridors.xml` (viz přílohu C):

```
4     <scene name="corridors" default-pattern="P1">
5       <pattern name="P1" bpm="118" measure="4" duration="2"
6         fade-out-duration="0">
7         <transitions>
8           <transition pattern="P2" when="VAR > 0.5 " />
9           <transition pattern="P3" when="VAR <= 0.5" />
10        </transitions>
11        <track name="TR1" file="assets/p1-tr1.flac" volume="VOLUME" />
12        <track name="TR2" file="assets/p1-tr2.flac" volume="VOLUME" />
13      </pattern>
14      <pattern name="P2" bpm="118" measure="4" duration="2">
15        <transitions>
16          <transition pattern="P1" when="VAR <= 0.5" />
17        </transitions>
18        <track name="TR1" file="assets/p2-tr1.flac" volume="VOLUME" />
19        <track name="TR2" file="assets/p2-tr2.flac" volume="VOLUME" />
20      </pattern>
21      <pattern name="P3" bpm="200" measure="1" duration="1">
22        <transitions>
23          <transition pattern="P1" when="BACK == 1" />
24        </transitions>
25      </pattern>
```

Grafické znázornění vzorů a přechodů scény `corridors` je na obr. 3.5.



Obrázek 3.5. Graf modelové scény `corridors`

V důsledku to znamená, že se může v různých konkrétních podobách odehrát následující scénář:

- Vzor `P1` obsahuje přechody (závislé na hodnotě proměnné `VAR`) do vzorů `P2` a `P3`. Vzor `P3` se přepne do `P1` za podmínky, že proměnná `BACK` bude mít hodnotu „1“.
- Po načtení předpisu do přehrávacího systému mají všechny proměnné (tedy i `VAR`) implicitní hodnotu „0“, čímž dojde okamžitě k přepnutí do vzoru `P3`, ačkoliv jako implicitní vzor byl ve vstupním XML dokumentu nastaven `P1`.
- Nyní se (za účelem přechodu do vzoru `P1`) určí `BACK` jako „1“.

V tuto chvíli nastává problém: Nacházíme se ve vzoru `P1` a je splněna podmínka přechodu do `P3` (`VAR` je „0“). V tom je ale přechodová podmínka do `P1` také vyhodnocena kladně, jelikož hodnota proměnné `BACK` je stále „1“. Mělo by tedy dojít k přechodu do `P1`, pak `P3`, následovně `P1`, opět `P3`… Mít dostatek času, mohli bychom takto pokračovat donekonečna – jelikož graf obsahuje smyčku (patrné na obr. 3.5), došlo v našem modelovém scénáři k zacyklení.

Tato situace je v implementaci přehrávacího systému ošetřena tak, že při vyhodnocování přehodů mezi vzory se jednotlivé vzory přidavají do zásobníku a ověřuje se, zda se již vzorem neprošlo – dojde k hledání cyklu v grafu. Pokud se průchodem dojde k vzoru, který je již v zásobníku, považuje se tento vzor za konečný. Při jakékoliv další změně v mapě proměnných, která neovlivní průchod grafem, se tak při vyhodnocování podmínek dojde (z důvodu předešlého zacyklení) opět k výchozímu vzoru, a nedojde tak k žádné změně v přehrávání.

Podobně jako třída `AudioController`, i `PlayingScene` umožňuje okamžité zastavení přehrávání aktuálně hrajícího vzoru.

3.2.3 Přehrávání zvukové stopy

Při přehrávání jednoho zvukového souboru se vytvoří dvě reálné instance přehrávače souboru, třídy `AudioPlayer`. Důvodem je to, že zvuková stopa se neopakuje až po jejím úplném dohrání, ale opakování začíná po zadaném počtu taktů, zatímco první iterace ještě přehrává dozvuky (tato skutečnost je zachycena na obr. 3.3, str. 15).

Pokud dojde ke změně v mapě proměnných ve třídě `VariablesMap`, tak se pomocí tříd balíčku `expressions` (viz kapitolu 3.3) vypočítá nová hlasitost zvukové stopy, která je následně předána přehrávači.

Při přepnutí vzoru lze dohrát zvukovou stopu dvěma způsoby:

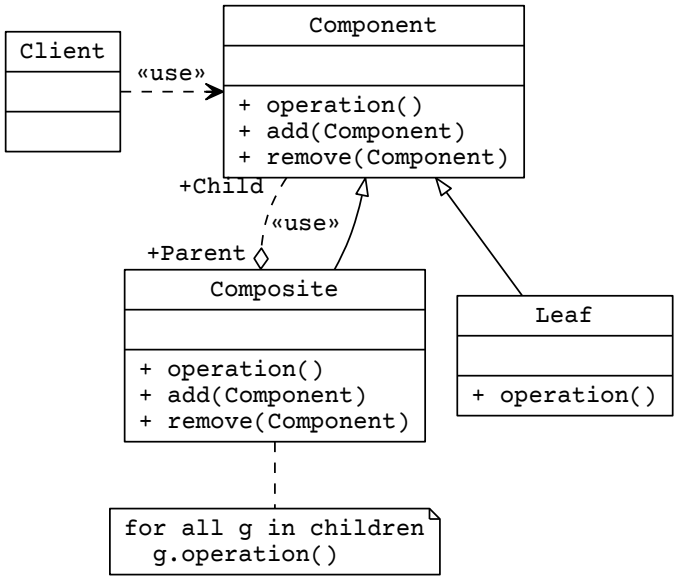
- Prvním z nich se aplikuje volitelné pravidlo vzoru `fade-out-duration` z předpisu (viz kapitolu 2.2.3). Pokud je jeho hodnota vyšší než 0, s dalším taktem po vyslání požadavku o ukončení přehrávání zvukové stopy se začne aplikovat efekt fade out. Zvláštním případem je nulová hodnota atributu `fade-out-duration`, což má za následek, že zvuková stopa je ukončena ihned.
- V případě, že v předpisu není u vzoru nadřazeném současné zvukové stopě zadán zmiňovaný atribut, zvukový soubor dohraje až do konce (tj. včetně dozvuku).

Kvůli funkci ukončení přehrávání scény je možné i okamžité ukončení přehrávání zvukové stopy.

3.3 Balíček `expressions`

Balíček `expressions` obsahuje třídy zastupující všechny symboly, které se mohou vyskytovat ve výrazu určenému ke zpracování třídou `ExpressionCompositor` (viz kapitolu 3.5), tzn. čísla (včetně čárky nebo tečky oddělující desetinná místa), písmena (tvořící názvy proměnných), závorky a aritmetické, logické a relační operátory.

Jedná se implementaci návrhového vzoru „Composite“ (viz obr. 3.6), v němž abstraktní třída `Expression` zastává pozici třídy `Component`. Abstraktní třída `BinaryExpression` je potomkem `Expression`, který obsahuje kolekci jiných výrazů, z nichž může každý být buď složený výraz, nebo operand. Celý výraz tak tvoří hierarchickou stromovou strukturu, jejíž uzly jsou operátory a listy představují operandy. Dojde tak na rozklad na syntaktický strom.



Obrázek 3.6. Obecné schéma návrhového vzoru „Composite“

3.3.1 Vyhodnocování výrazů

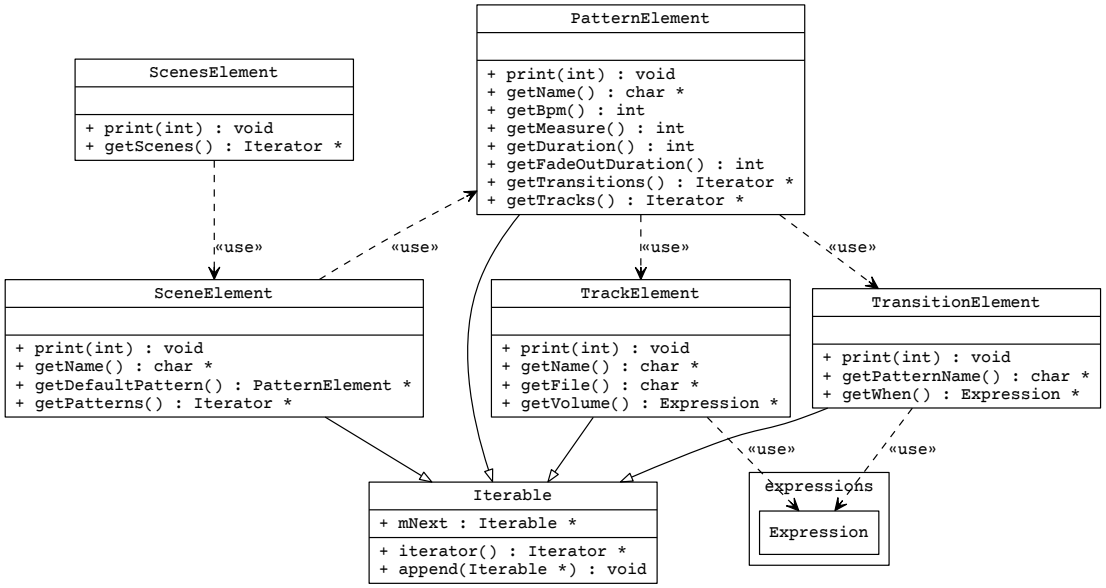
Při vyhodnocování výrazů se využívá techniky *rozděl a panuj*, což je metoda, při které se větší výpočetní problémy dělí na menší úlohy, které se vypočítají snáz. Přímo se zde využívá sestavený syntaktický strom, nad kterým se provádí algoritmická operace.

Všechny třídy balíčku `expressions` implementují metodu pro výpočet výsledku, která přebírá mapu proměnných. Výsledek lze získat v kterémkoliv vrcholu stromu, kdy vyhodnocovaný uzel vyvolává výpočet na podvýrazech a list vrací konečnou hodnotu. Zvláštním případem listu je proměnná, který pro získání výsledku musí najít svou hodnotu v mapě proměnných. Tímto se sestaví výsledek až po kořen syntaktického stromu, kde se získá vypočítaná hodnota výrazu pro aktuální stav proměnných.

3.4 Balíček `elements`

Balíček `elements` sdružuje třídy reprezentující jednotlivé elementy, které se mohou vyskytovat v předpisu ve formátu XML (viz kapitolu 2.2). Přebírají atributy XML elementů. Právě z instancí těchto tříd vytváří balíček `parser` (pro více informací o syntaktické analýze viz kapitolu 3.5) v paměti počítače strom zachovávající hierarchii vstupních dat, který je později využíván pro rozhodování při přehrávání adaptivní hudby.

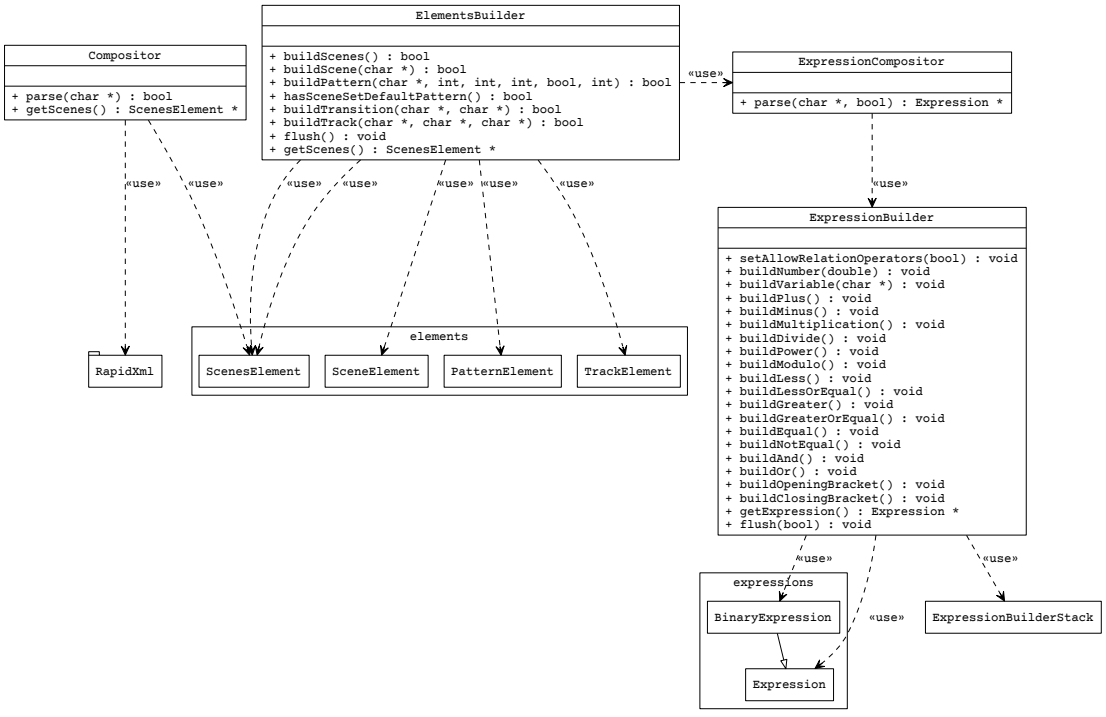
Elementy, jejichž skupiny je zapotřebí procházet, implementují rozhraní `Iterable`.



Obrázek 3.7. UML diagram tříd balíčku `elements`

3.5 Balíček `parser`

V balíčku `parser` se nachází třídy, které slouží k syntaktické analýze vstupních XML dokumentů.



Obrázek 3.8. UML diagram tříd balíčku `parser`

3.5.1 Graf scény

Třída `Compositor` provádí lexikální analýzu zadaného předpisu. K tomu vyžívá služeb knihovny RapidXml[1]) – v současné době nejrychlejšího XML parseru pro programovací

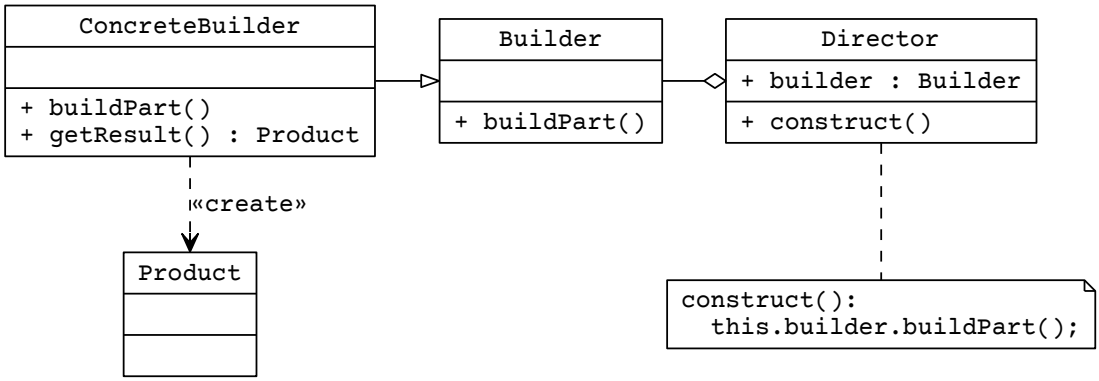
[1]) `http://rapidxml.sourceforge.net/`

jazyk C++. V tabulce 3.1 je srovnání rychlosti XML parserů (všechny výsledky jsou v počtu cyklů CPU na jeden znak zdrojového textu).

Při úspěšném nalezení potřebných atributů u konkrétního elementu jsou získaná data předána třídě `ElementsBuilder`, která z dat ze vstupního předpisu vybuduje v paměti počítače syntaktický strom ze tříd balíčku `elements` (viz kapitolu 3.4). Jedná se o aplikovaný návrhový vzor „Builder“ (viz obr. 3.9).

Tabulka 3.1. Srovnání rychlosti XML parserů za použití kompilátoru GCC 4.1.1. Data jsou převzata z [21].

Platforma	`strlen()`	RapidXml	pugixml 0.3	pugxml	TinyXml
Pentium 4	0,8	6,1	9,5	67,0	413,2
Core 2	0,6	4,6	5,4	28,3	229,3
Athlon XP	0,9	8,2	9,2	33,7	265,2
Pentium 3	1,0	6,7	8,9	35,3	316,0



Obrázek 3.9. Obecné schéma návrhového vzoru „Builder“

Pokud bude nahrán předpis, který obsahuje název už dříve načtené scény, dojde ke kontrole aktuálně přehrávaných scén, a všechny hrající scény s tímto názvem se ukončí a definice scény se přepíše dle obsahu nového předpisu.

3.5.2 Vyhodnocování výrazů

Jelikož aplikace pracuje s proměnnými, čísly a podmínkami a jejich vyhodnocováním, bylo nutné zhotovit pro vyhodnocování výrazů vlastní lexikální a syntaktický analyzátor. Třída `ExpressionCompositor` postupně čte jednotlivé znaky textového řetězce obsahujícího podmínkovou formuli a předává je třídě `ExpressionBuilder`. Ta implementuje algoritmus, který pro daný řetězec sestaví syntaktický strom. Opět je zde využit návrhový vzor „Builder“ (viz obr. 3.9).

Knihovna umožňuje vyhodnocovat aritmetické, logické a relační operátory. Přehled implementovaných operátorů, které jdou využít v předpisech, a jejich možný zápis je v tabulkách 3.2 až 3.3.

Tabulka 3.4. Logické operátory

Třída	Symbol(y)
`And`	`&` nebo `&&`
`Or`	`|` nebo `||`

Tabulka 3.5. Ostatní symboly

Třída	Symbol(y)
`Brackets`	(a)

Tabulka 3.2. Aritmetické operátory

Třída	Symbol(y)
`Plus`	`+`
`Minus`	`-`
`Multiplication`	`*`
`Power`	`**` nebo `^`
`Divide`	`/`
`Modulo`	`%`

Tabulka 3.3. Relační operátory

Třída	Symbol(y)
`Equal`	`=` nebo `==`
`NotEqual`	`<>`
`Less`	`<`
`LessOrEqual`	`<=`
`Greater`	`>`
`GreaterOrEqual`	`>=`

3.6 API

Potřebné funkce API byly navrženy v kapitole 2.3. Pro potřeby generování knihovny pro Unity jsou v souboru `static_api.c` vytvořeny statické funkce, které využívají API definované ve třídě `PlayerAPI`. Knihovna nabízí následující funkce:

```
bool AMEN_loadFile(char *filename);
bool AMEN_playScene(char *sceneName);
bool AMEN_stopScene(char *sceneName);
bool AMEN_setVariable(char *variableName, double variableValue);
void AMEN_stop();
void AMEN_setPrintInfo(bool value);
```

- Funkce `AMEN_loadFile()` má jako formální parametr textový řetězec `filename`, který obsahuje cestu k XML souboru s popisem scény, nebo více scén. Pokud načtení souboru proběhne v pořádku, vrací hodnotu `true`, jinak `false`.
- Funkce `AMEN_playScene()` má jako formální parametr textový řetězec `sceneName`, který obsahuje název scény, která se má začít přehrávat. Pokud začne přehrávání, funkce vrací hodnotu `true`, jinak `false`.
- Funkce `AMEN_stopScene()` má jako formální parametr textový řetězec `sceneName`, který obsahuje název scény, jejíž přehrávání se má ukončit. Pokud se zadaná scéna zdárně ukončí, funkce vrací hodnotu `true`, jinak `false`.
- Funkce `AMEN_setVariable()` nastaví proměnnou `variableName` na hodnotu `variableValue` datového typu `double`. Pokud vše proběhne, funkce vrací hodnotu `true`, jinak `false`.
- Funkce `AMEN_stop()` ukončí veškeré přehrávání. Nemá žádné formální parametry a návratovou hodnotu.
- Funkce `AMEN_setPrintInfo()` zapne výpis ladících hlášek, pokud je předán parametr `value` s hodnotou `true`. S `false` je výpis zakázán.

3.7 Multiplatformnost

Jelikož se předpokládá využití přehrávacího systému ve frameworku Unity, který umožňuje deployment do celé řady rozdílných platforem (viz kapitolu 1.4), je žádoucí zajistit bezproblémovou implementaci alespoň na nejrozšířenější systémy (jmenovitě OS Windows od společnosti Microsoft a systémy s linuxovým jádrem). Pro více informací o kompilaci viz přílohu B.

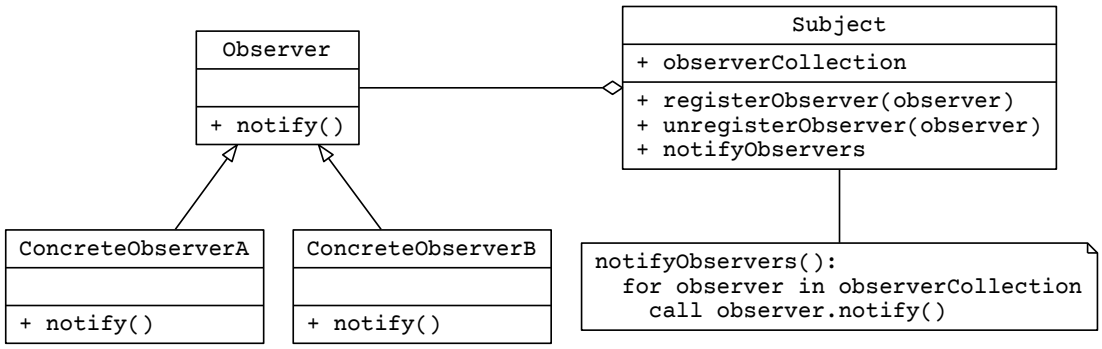
Použité externí knihovny – ffmpeg, Port Audio a RapidXml – byly vybrány s ohledem na jejich multiplatformnost.

Kapitola 4
Implementace prototypovacího nástroje

Prototypovací nástroj je aplikace implementovaná v jazyce Java s grafickým uživatelským rozhraním, které uživateli umožňuje přehrávat adaptivní hudbu dle zadaných popisů scén. Návrh tohoto programu byl popsán v kapitole 2.4.

4.1 Popis uživatelského rozhraní

Hlavní okno aplikace se skládá ze čtyř samostatných komponent – ve zdrojovém kódu jsou označeny jako *widgety*. Náhled tohoto okna je na obr. 4.2. Každý widget je třída, která přijímá signály z nadřazené třídy (tj. od `MainWindow`), nebo jí je naopak posílá. K tomu je využit návrhový vzor „Observer“ (viz obr. 4.1).



Obrázek 4.1. Obecné schéma návrhového vzoru „Observer“

Komponenta `WidgetFiles` při otevření předpisu načte soubor do záložky, která obsahuje textové pole `RSyntaxTextArea`[1]) umožňující zvýraznění zvolené syntaxe – v našem případě XML. Otevřené předpisy lze upravovat a následně ukládat a je možné i vytvořit soubory nové.

Při načítání vstupních XML dokumentů jsou v obsahu atributů `name` (u elementů typu `<scene>`), `when` (u elementů typu `<transition>`) a `volume` (u elementů typu `<track>`) detekovány názvy scén a proměnných. Ty jsou poté předány komponentám `WidgetScenes` a `WidgetVariables`.

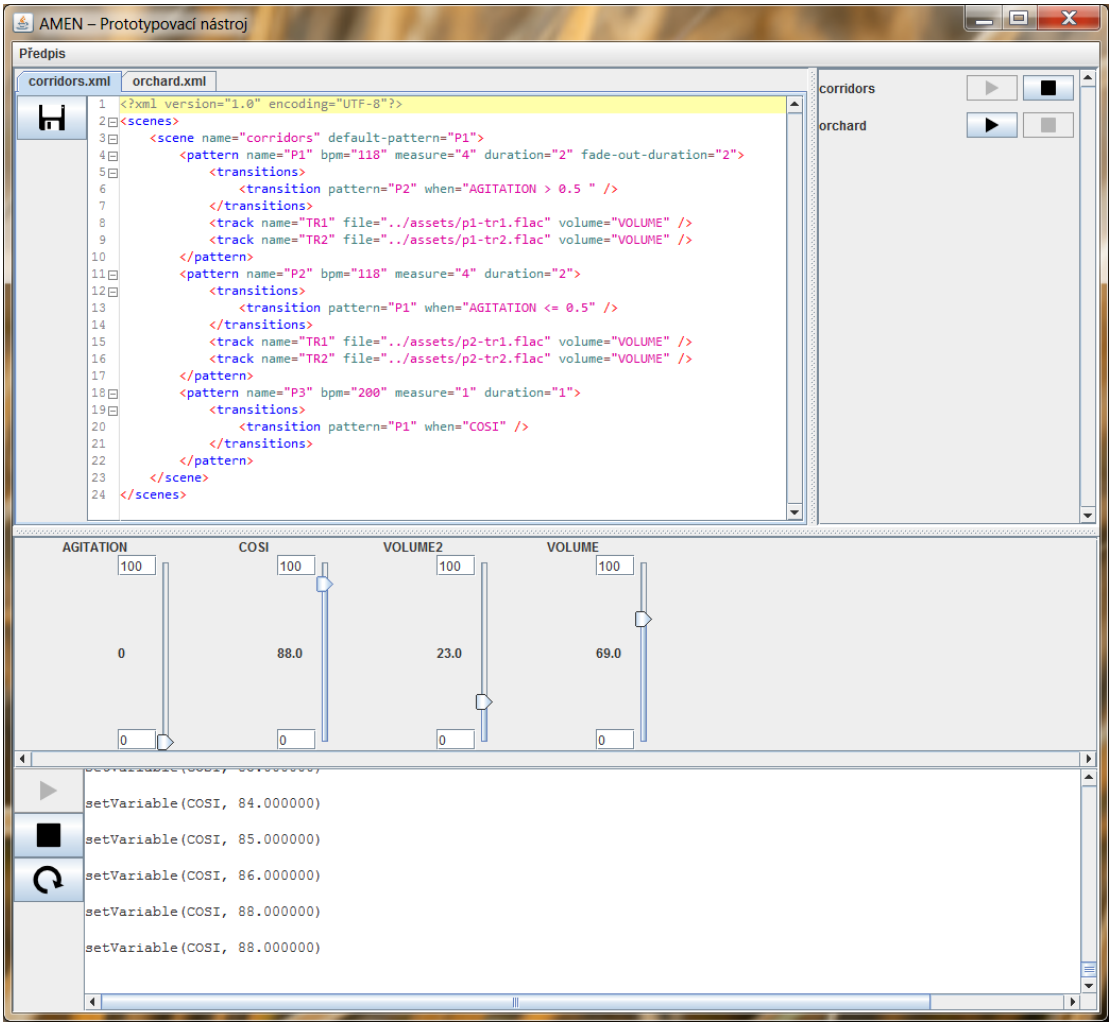
Komponenta `WidgetScenes` obsahuje seznam všech dostupných scén. Jednotlivé řádky seznamu obsahují název scény a tlačítka pro spuštění a zastavení scény, která jsou dostupná pouze v případě, že je (třídou `WidgetAMEN`) spuštěno přehrávání adaptivní hudby.

Pro nalezené proměnné se ve widgetu `WidgetVariables` vytvoří komponenty, které obsahují název proměnné, posuvník pro nastavení hodnoty proměnné a textová pole pro zadání horní a dolní meze hodnoty proměnné.

Komponenta `WidgetAMEN` zapouzdřuje třídu, která komunikuje s knihovnou pro přehrávání adaptivní hudby. Obsahuje textové pole, které zobrazuje výstupní ladící hlášky

[1]) `http://fifesoft.com/rsyntaxtextarea/`

přehrávacího systému, a tlačítka sloužící k spuštění a zastavení přehrávání. V případě, že je spuštěno přehrávání, jsou aktivována tlačítka v komponentě `WidgetScenes`.



Obrázek 4.2. Hlavní okno prototypovacího nástroje

Mezi ladící hlášky patří např. informace o syntaktické chybě v předpisu a jejím přesném umístění:

```
ERROR: Syntax error in "AGITATION > 0.5 > 2" at 16
ERROR:                                    ^

ERROR: Syntax error in "VOLUME++" at 7
ERROR:                          ^
```

4.2 Komunikace s přehrávacím systémem

Třída `LibraryConnector` zajišťuje spojení mezi prototypovacím nástrojem a přehrávacím systémem – spouští podproces, který na svém standardním vstupu přijímá příkazy, které volají funkce API popsané v kapitole 3.6. Běžný uživatel prototypovacího nástroje tyto příkazy samozřejmě znát nemusí, protože je za něj volá sama aplikace. Tabulka 4.1 popisuje jednotlivé příkazy tohoto programu a jejich atributy.

Tabulka 4.1. Seznam XML elementů a jejich atributů

Příkaz	Atribut(y)	Význam
`exit`		Ukončí veškeré přehrávání.
`play`	`scene`	Spustí scénu `scene`.
`load`	`filename`	Načte soubor `filename`.
`set`	`variable value`	Nastaví proměnnou `varible` na hodnotu `value`.
`stop`	`scene`	Zastaví scénu `scene`.

Chceme provést následující kroky:

1) Načíst předpis `scenario.xml`,
2) nastavit proměnnou `VOLUME` na hodnotu „100“,
3) spustit scénu `forrest`,
4) ukončit scénu `forrest`,
5) vypnout přehrávací systém.

Předchozí body provedeme zadáním těchto příkazů do programu `amen_player(.exe)`:

```
load scenario.xml
set VOLUME 100
play forrest
stop forrest
exit
```

Tento program – tvořící adaptér mezi prototypovacím nástrojem a přehrávacím systémem – lze spouštět s inicializačními příkazy, které se provedou okamžitě po spuštění přehrávacího systému. K tomu je možné využít přepínače popsané v tab. 4.2.

Tabulka 4.2. Seznam přepínačů a jejich parametrů

Přepínač	Atribut(y)	Význam
`-h`		Zobrazí nápovědu.
`-l`	`filename`	Načte soubor `filename`.
`-p`	`scene`	Spustí scénu `scene`.
`-s`	`variable=value`	Nastaví proměnnou `varible` na hodnotu `value`.

Pokud tedy např. chceme po spuštění přehravacího systému načíst soubor `scenario.xml`, nastavit proměnnou `VOLUME` na hodnotu „100“ a spustit scénu `forrest`, uděláme to následovně:

```
amen_player.exe -l scenario.xml -s VOLUME=100 -p forrest
```

Kapitola 5
Integrace s projekty v Unity

Jedním z požadavků zadání práce (viz kapitolu 1.1) je zajištění toho, aby přehrávací systém bylo možné integrovat do projektů vyvíjených v prostředí Unity. Toho se docílí tak, že knihovna bude importována do projektu. Takové knihovně se v Unity říká *plugin* (česky *rozšíření*). Pluginy jsou podporovány **pouze v Unity Pro**.

Ukázkový skript v jazyce C# `PluginImport.cs` (viz přílohu C) názorně demonstruje, jak použít přehrávací systém přímo v projektu. Do třídy, která bude využívat API přehrávacího systému (viz kapitolu 3.6), je nutné vložit tyto importovací příkazy:

```
6     [DllImport ("amen")]
7     private static extern bool AMEN_loadFile(
8       [MarshalAs(UnmanagedType.LPStr)] string filename
9       );
10
11    [DllImport ("amen")]
12    private static extern bool AMEN_playScene(
13      [MarshalAs(UnmanagedType.LPStr)] string sceneName
14      );
15
16    [DllImport ("amen")]
17    private static extern bool AMEN_stopScene(
18      [MarshalAs(UnmanagedType.LPStr)] string sceneName
19      );
20
21    [DllImport ("amen")]
22    private static extern bool AMEN_setVariable(
23      [MarshalAs(UnmanagedType.LPStr)] string variableName,
24      double variableValue
25      );
26
27    [DllImport ("amen")]
28    private static extern void AMEN_stop();
29
30    [DllImport ("amen")]
31    private static extern void AMEN_setPrintInfo(bool value);
```

Některé funkce API mají formální parametry datového typu `char *` (ukazatel na `char`), který jazyk C# nepodporuje. Bylo je tedy nutné je uvést jako `[MarshalAs(UnmanagedType.LPStr)] string`, jak je vidět na předchozím úseku kódu. Kvůli direktivám `DllImport` a `MarshalAs` je navíc zapotřebí na začátku skriptu zadat tento příkaz:

```
3   using System.Runtime.InteropServices;
```

Nyní je možné využívat jednotlivé příkazy. Skript `PluginImport.cs` v inicializační části (v proceduře `Start()`) načte předpis `corridors.xml`, spustí scénu `corridors` a nastaví

proměnnou `VOLUME` na hodnotu „100“. Uživatel poté může klávesami `A`, `B` a `C` měnit hodnotu zmíněné proměnné a mezerníkem ukončit přehrávání. Konkrétní kód tedy vypadá následovně:

```
33      void Start () {
34        AMEN_loadFile("corridors.xml");
35        AMEN_playScene("corridors");
36        AMEN_setVariable("VOLUME", 100.0);
37      }
38
39      void Update () {
40        if (Input.GetKeyDown("space"))    AMEN_stop();
41        if (Input.GetKeyDown(KeyCode.A))  AMEN_setVariable("VOLUME", 0);
42        if (Input.GetKeyDown(KeyCode.B))  AMEN_setVariable("VOLUME", 50);
43        if (Input.GetKeyDown(KeyCode.C))  AMEN_setVariable("VOLUME", 100);
44      }
```

Kapitola 6
Testování

Přehrávací systém je nutné řádně otestovat. Cílem každého testování softwaru je odhalení chyb v programu. Testování je samozřejmě možné provádět i během vývoje a úprav aplikace. V této kapitole si popíšeme postup funkčního a nefunkčního testování implementované knihovny a jeho výstupy.

6.1 Funkční testy

Funkční testy slouží k ověření správné činnosti vyvíjeného programu. Jejich zdrojové kódy jsou k nalezení ve složce `tests` (viz přílohu C) a mohou být zkompilovány a následně spuštěny příkazem `make tests`. Pokyny pro kompilaci jsou v příloze B. Všechny testy jsou implementovány jako testy splnění – pro úspěšné dokončení testu je nutné testovaným subjektem vrátit očekávanou hodnotu.

Byly implementovány dvě testovací makra. Jedná se o tzv. *aserce*, které porovnávají dvě hodnoty. Makro `amenASSERT()` zajišťuje výpis na standardní výstup tečku, pokud se první a druhý parametr sobě rovnají, v opačném případě vypíše „`E`“ a číslo řádku v testovacím skriptu, na kterém nastalo selhání. Makro `amenNoASSERT()` má podobné chování, ale liší se v tom, že pro splnění aserce se musí konečné hodnoty zadaných parametrů lišit. Pár názorných triviálních příkladů[1]):

```
amenASSERT(false, false);      // .
amenASSERT(false, true);       // E
amenASSERT(true, false);       // E
amenASSERT(true, true);        // .

amenNoASSERT(false, false);    // E
amenNoASSERT(false, true);     // .
amenNoASSERT(true, false);     // .
amenNoASSERT(true, true);      // E
```

6.1.1 Unit testing

Unit testing (česky občas uváděno jako *testování jednotek* nebo *jednotkové testování*) je pravděpodobně nejrozšířenější metoda funkčního testování, obzvlášť oblíbená u objektově orientovaných projektů, kde jsou *jednotky* snadno určitelné – můžou to být třídy, nebo metody. Jedná se o tzv. *white-box testing*, při kterém známe vnitřní strukturu jednotky a testy ověřují správnou funkčnost elementárních funkcí a metod.

Tyto testy se provedou spuštěním souboru `amen_unit_tests` (na Windows s příponou `.exe`). Program vypíše následující text:

```
..................................................................
.................................
```

[1]) `false` je ekvivalentní nulové hodnotě, `true` je potom jakákoliv nenulová hodnota.

```
Passed tests: 107/107
Failed tests: 0/107
```

Tento výstup nám říká, že všechny testy proběhly úspěšně a žádný z nich neselhal.

6.1.2 Funkcionální testy

Funkcionální testy jsou v souladu s myšlenkou *black-box testing*, kdy nás při testování již nezajímá konkrétní implementace, ale výstupy větších programových celků.

Funkcionální testování uskuteční soubor `amen_functional_tests` (na Windows s příponou `.exe`). Výstupy programu se interpretují stejně jako v předchozí kapitole.

```
.........

Passed tests: 9/9
Failed tests: 0/9
```

Všechny testy proběhly úspěšně.

6.2 Nefunkční testy

Podpůrné zvukové systémy pro počítačové hry by neměly příliš vytěžovat hardware – musí totiž ustoupit výpočetně složitějším úkolům, mezi které může patřit např. výpočty související s fyzikálním modelem, grafika a její vykreslování, nebo chování umělé inteligence. Z tohoto důvodu bylo nutné ověřit, kolik výpočetních prostředků vyžaduje přehrávací systém během své činnosti. Test byl prováděn na zařízení s následující konfigurací:

- CPU – Intel Core i5-2410M (2,30 GHz)[1]),
- RAM – 4 GB,
- OS – Windows 7 Home Premium (Service Pack 1).

Programe Process Explorer v16.02[2]) bylo zjištěno, že přehrávací systém nikdy nevytěžuje procesor na více než 3,5 % (průměrná hodnota je zhruba poloviční).

[1]) Podrobnější specifikace dostupná na `http://ark.intel.com/products/52224/Intel-Core-i5-2410M-Processor-3M-Cache-up-to-2_90-GHz`.
[2]) `http://www.sysinternals.com/`

Kapitola 7
Závěr

7.1 Splnění cílů

Tématem této bakalářské práce je adaptivní hudba v počítačových hrách. V souvislosti s ní se používá několik termínů, jejichž smysl není vždy úplně zřejmý, čímž jejich význam pro nezasvěceného člověka splývá. V kapitole 1.2 (str. 1) jsou proto tyto pojmy osvětleny. Jak napovídá název práce; jako nejvýstižnější se v oblasti této problematiky jeví označení „adaptivní hudba“.

První pokusy s adaptivním přístupem k přehrávání scénické hudby v počítačových hrách můžeme pozorovat už v průběhu osmdesátých let. Od té doby vznikla řada frameworků a ucelených řešení, jak takovou nelineární hudbu implementovat v multimediálním projektu. Nejvýznamnější stávající řešení jsou představeny v kapitole 1.3 (str. 1.3). V současnosti je jedničkou na trhu Wwise (viz kapitolu 1.3.1, str. 4), který je využíván ve spoustě moderních počítačových hrách.

Podstatnou částí práce je navržení modelu adaptivní hudby neboli enginu schopného reagovat na vývoj situace ve hře dle předpisu – o tom pojednává kapitola 2 (str. 7). Jsou zde vymezeny funkční a nefunkční požadavky na takový systém a vymezeny pojmy, které hrají důležitou roli např. při popisu scénáře pro přehrávání adaptivní hudby. V jedné z podkapitol je též definována gramatika zápisu popisu scén, pro které jsme zavedli označení „předpisy“. Tyto gramatická pravidla pro XML dokumenty jsou popsány jazykem DTD.

Implementací návrhu modelu přehrávacího systému v programovacím jazyce C++ vznikla knihovna AMEN (zkratka slovního spojení *Adaptive Music Engine*), která poskytuje API umožňující několika jednoduchými příkazy řídit přehrávání adaptivní hudby v počítačové hře. Realizace přehrávacího systému je popsána v kapitole 3 (str. 13). Knihovna využívá mj. knihovnu ffmpeg, která byla zkompilována s podporou čtení pouze z lokálního úložiště a jen se zvukovými kodeky, aby se zmenšila její velikost.

Pro možnost vyzkoušení vytvořených předpisů byl navržen (viz kapitolu 2.4 na str. 11) a následně implementován (viz kapitolu 4 na str. 25) prototypovací nástroj. Jedná se vlastně o „hudební debugger“, kterým je možné ladit popisy scén, nebo vytvářet nové. Grafické uživatelské rozhraní (napsané v programovacím jazyce Java) umožňuje spouštět scény z načtených XML dokumentů a při přehrávání nastavovat hodnoty jednotlivých proměnných.

Jelikož se předpokládá využití přehrávacího systému v reálném projektu, který je vyvíjen v prostředí Unity (viz kapitolu 1.4, str. 5), bylo nutné zajistit bezproblémovou integraci. Řešení tohoto úkolu (včetně názorného příkladu) je popsáno v kapitole 5 (str. 29). Z důvodu, že Unity umožňuje nasazení na nejrůznější platformy, je umožněna kompilace přehrávacího systému na nejrozšířenější operační systémy – program `make` (s pomocí souboru `Makefile`) vygeneruje pro Windows dynamickou knihovnu `amen.dll` a pro Linux `libamen.so`.

Veškerý realizovaný software byl otestován příslušnými funkčními a nefunkčními testy. Popis testování a jeho výstupy jsou popsány v kapitole 6 (str. 31).

7.2 Licenční podmínky

Přehrávací systém je dostupný pod licencí *GNU Lesser General Public License*[1]) (někdy též uváděno jako *GNU Library General Public License*). V praxi to znamená, že knihovna může být užívána programem, který nemá stejnou licenci – tedy ve svobodném i v proprietárním softwaru.

Prototypovací nástroj je uvolněn pod licení *GNU General Public License*[2]) (zkráceně *GPL*), díky čemuž může být libovolně používán, modifikován, šířen atd. To vše ale za té podmínky, že odvozený software bude i po modifikaci podléhat licenci GPL – bude mít volně dostupné zdrojové kódy a bude zdarma.

7.3 Možnosti dalšího rozšíření

Díky důrazu na principy *low coupling* a *high cohesion* (česky občas uváděno jako *nízká vazba* a *veliká soudružnost*) při implementaci je rozšíření funkčnosti přehrávacího systému jednoduché. Mezi nápady na další funkčnost jsou např. master volume, crossfade, flexibilní definice konvexní obálky pro techniku fade out a další.

[1]) V úplném znění je tato licence dostupná na `http://www.gnu.org/licenses/lgpl.html`.
[2]) V úplném znění je tato licence dostupná na `http://www.gnu.org/licenses/gpl-3.0.txt`.

Literatura

[1] Dynamic music. In: *Wikipedia* [online]. 2014 [cit. 2014-04-01]. Dostupné z: http://en.wikipedia.org/wiki/Dynamic_music

[2] VAN NISPEN TOT PANNERDEN, Than, Sander HUIBERTS, Sebastiaan DONDERS a Stan KOCH. *The NLN-player: A system for nonlinear music in games.* Hilversum (Nizozemsko), 2011. Dostupné z: http://download.captivatingsound.com/Nispen-Huiberts-Donders-Koch-Interactive-Music-the-NLN-player.pdf. Diplomová práce. Utrecht School of the Arts.

[3] FRISHERT, Stijn. Vertical Reorchestration. In: *Stijn Frishert: Adaptive game composer* [online]. 2014 [cit. 2014-05-05]. Dostupné z: http://stijnfrishert.com/vertical-reorchestration/

[4] Group Report: What is Is Interactive Audio? And What Should It Be?. In: *The Eighth Annual Interactive Music Conference PROJECT BAR-B-Q 2003* [online]. 2003 [cit. 2014-05-05]. Dostupné z: http://www.projectbarbq.com/bbq03/bbq03r5.htm

[5] CLARK, Andrew. Defining Adaptive Music. In: *Gamasutra* [online]. 2007 [cit. 2014-03-24]. Dostupné z: http://www.gamasutra.com/view/feature/1567/defining_adaptive_music.php

[6] Reactive & Generative Music Bursting. In: *ICrates* [online]. 2011 [cit. 2014-04-30]. Dostupné z: http://www.icrates.org/reactive-generative-music-bursting/

[7] BERNSTEIN, Daniel. Creating an Interactive Audio Environment. In: *Gamasutra: The Art & Business of Making Games* [online]. 1997 [cit. 2014-04-27]. Dostupné z: http://www.gamasutra.com/view/feature/131646/creating_an_interactive_audio_.php

[8] Wwise Integration. *Audiokinectic* [online]. 2014 [cit. 2014-05-03]. Dostupné z: https://www.audiokinetic.com/products/wwise-integration/

[9] Audiokinectic Wwise. *Audiokinectic* [online]. 2014 [cit. 2014-03-24]. Dostupné z: https://www.audiokinetic.com/products/wwise/

[10] Wwise Product Pricing. *Audiokinectic* [online]. 2014 [cit. 2014-03-24]. Dostupné z: https://www.audiokinetic.com/licensing/pricing/

[11] Customers. *Audiokinectic* [online]. 2014 [cit. 2014-05-03]. Dostupné z: https://www.audiokinetic.com/community/customers/

[12] Interview with Pedro Macedo Camacho. In: *Audiokinectic: Customer Profile* [online]. 2010 [cit. 2014-04-01]. Dostupné z: http://www.audiokinetic.com/download/documents/customer_profiles/Audiokinetic_Customer_Profile_Pedro_Macedo_Camacho.pdf

[13] IMUSE. In: *Wikipedia* [online]. 2014 [cit. 2014-04-01]. Dostupné z: http://en.wikipedia.org/wiki/IMUSE

[14] LucasArts' Secret History: Monkey Island 2: LeChuck's Revenge: The Music of LeChuck's Revenge. In: *The International House of Mojo* [online]. 2008 [cit. 2014-04-01]. Dostupné z:
http://mixnmojo.com/features/sitefeatures/LucasArts-Secret-History-Monkey-Island-2-LeChucks-Revenge/9

[15] 64-bit programming for Game Developers. In: *MSDN – The Microsoft Developer Network* [online]. 2013 [cit. 2014-05-05]. Dostupné z:
http://msdn.microsoft.com/en-us/library/ee418798.aspx

[16] DirectMusic. In: *Wikipedia* [online]. 2013 [cit. 2014-05-05]. Dostupné z:
http://en.wikipedia.org/wiki/DirectMusic

[17] SONiVOX JETCreator User Manual. *Android Developers* [online]. ? [cit. 2014-04-16]. Dostupné z:
http://developer.android.com/guide/topics/media/jet/jetcreator_manual.html

[18] JET™ Programming Manual. *Netmite* [online]. 2009 [cit. 2014-04-16]. Dostupné z:
http://www.netmite.com/android/mydroid/1.5/external/sonivox/docs/JET_Programming_Manual.htm

[19] Multiplatform: Publish your game to over 10 platforms. *Unity: Game Engine* [online]. 2014 [cit. 2014-04-30]. Dostupné z:
http://unity3d.com/unity/multiplatform

[20] Unity Web Player. *Unity: Game Engine* [online]. 2014 [cit. 2014-04-30]. Dostupné z:
https://unity3d.com/webplayer

[21] KALICINSKI, Marcin. RapidXml Manual. In: *RapidXml* [online]. 2009 [cit. 2014-03-31]. Dostupné z:
http://rapidxml.sourceforge.net/manual.html

Příloha A
Použité zkratky

2D	„Dvourozměrný“ nebo „dvoudimenzionální“.
3D	„Trojrozměrný“ nebo „trojdimenzionální“.
AAC	*Advanced Audio Coding*.
AC-3	Staré označení *Dolby Digital*.
AMEN	*Adaptive Music Engine*.
APE	*Monkey's Audio*.
API	*Application Programming Interface*.
ASF	*Advanced Systems Format*.
BPM	Počet úderů za minutu (z anglického *beats per minute*).
CD	*Compact Disc*.
CPU	*Central Processing Unit*.
DLS	*Downloadable Sounds*.
DTD	*Document Type Definition*.
FLAC	*Free Lossless Audio Codec*.
GCC	*GNU Compiler Collection*[1]).
GUI	Grafické uživatelské rozhraní (z anglického *Graphical User Interface*).
IDE	Vývojové prostředí (z anglického *Integrated Development Environment*).
iMUSE	*Interactive Music Streaming Engine*.
JET	*Jet Interactive Engine*.
MIDI	*Musical Instrument Digital Interface*.
MOV	*QuickTime File Format*.
MP3	*MPEG Audio Layer III*.
MPC	*Musepack*.
OGG	*Vorbis*.
OS	*Operační systém*.
PTS	*Presentation timestamp*.
SCUMM	*Script Creation Utility for Maniac Mansion*.
TTA	*True Audio*.
VST	*Virtual Studio Technology*.
WAV	*Waveform Audio File Format*.
WV	*WavPack*.
XML	*Extensible Markup Language*.

[1]) `http://gcc.gnu.org/`

Příloha B
Pokyny pro kompilaci

Ke kompilaci přehrávacího systému na operačních systémech Windows a Linux byl zhotoven soubor `Makefile` (viz přílohu C), který určuje závislosti mezi zdrojovými soubory pro překlad programem `make`. Díky tomuto souboru je možné zadávat následující příkazy:

- `make all`
  Zkompiluje přehrávací systém. Na Windows vygeneruje dynamickou knihovnu `amen.dll` a spustitelný soubor `amen_player.exe` pro prototypovací nástroj. Na linuxových systémech jsou vytvořeny soubory `libamen.so` a `amen_player`.
- `make tests`
  Zkompiluje testy a spustí je. Na operačním systému Windows vzniknou soubory `amen_unit_tests.exe` a `amen_functional_tests.exe` a na systémech s linuxovým jádrem `amen_unit_tests` a `amen_functional_tests`.
- `make`
  Totéž jako `make all`.

Zkompilované soubory se nacházejí ve složce `build`. Pokud je na Windows problém s načítáním knihoven, všechny potřebné knihovny jsou ve složce `libs-dll`.

Jaké verze knihoven ffmpeg a Port Audio je možné přehrávací systém zkompilovat a používat naznačuje soubor `build_linux.sh` (viz přílohu C).

Příloha C
Obsah přiloženého CD

Výpis není kompletní – tento seznam obsahuje jen ty adresáře a soubory, které jsou zmíněny v textu nebo jsou důležité.

- /
  - _BP — Text této bakalářské práce
    - images
  - amenPrototyper
    - build
    - icons
    - libs
    - nbproject — NetBeans projekt prototypovacího nástroje
    - src — Zdrojové kódy prototypovacího nástroje
  - assets — Zvukové soubory pro vyzkoušení
  - build — Zkompilované soubory
  - libs — Externí knihovny
  - libs-dll
  - src — Zdrojové kódy přehrávacího systému
    - controller
    - elements
    - expressions
    - parser
    - player
  - tests — Zdrojové kódy testů
    - functionals
    - units
  - amen.dtd — Popis struktury XML dokumentů v DTD
  - build_linux.sh — Ukázka kompilace na Linuxu
  - corridors.xml — Ukázkový předpis
  - Makefile
  - orchard.xml — Ukázkový předpis
  - PluginImport.cs — Skript importující knihovnu AMEN do Unity