VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

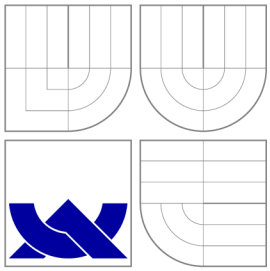
# INTELIGENTNÍ HUDEBNÍ PŘEHRÁVAČ PRO ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

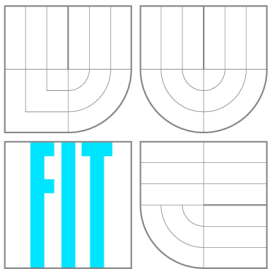
AUTOR PRÁCE
AUTHOR

MICHAL TOMÁŠEK

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

# INTELIGENTNÍ HUDEBNÍ PŘEHRÁVAČ PRO ANDROID

INTELLIGENT MUSIC PLAYER FOR ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL TOMÁŠEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. RÓBERT KALMÁR
SUPERVISOR

BRNO 2014

# Abstrakt

V dnešní době je na trhu velké množství hudebních přehrávačů. Žádný z nich však neposkytuje funkci predikce, která by pracovala v reálném čase přímo při běžném používání přehrávače.

Bakalářská práce se zabývá návrhem a implementací hudebního přehrávače s mechanismem predikce. Důraz je kladen jak na vytvoření kvalitní architektury aplikace, tak na zvolení vhodného algoritmu, který bude zajišťovat predikci. Výběru algoritmu předchází výběr kategorie metod strojového učení. Výsledný prediktivní algoritmus je rozdělen do několika sekcí a je tedy nutné zvolit či vymyslet řešení pro všechny jeho části.

Výstupem této práce je hudební přehrávač pro OS Android umožňující provádění predikce a učení se z interakce v reálném čase.



# Abstract

Today's market is full of different kinds of music players. However there is no music player using prediction, which would be able to work with usual usage and in real time.

This bachelor's thesis deals with the design and implementation of a music player with implemented prediction. The creation of high quality architecture for this application and the choice of appropriate prediction algorithm are emphasized. At first, the category of methods of machine learning and afterwards the algorithm are determined. The final prediction algorithm is divided into a several sections, therefore it is necessary to pick or create a solution for all its parts.

The outcome of this project is the music player for Android OS which would be able to predict and learn from a user interaction in real time.



# Klíčová slova

OS Android, hudební přehrávač, predikce, posilované učení, popularita skladeb, ohodnocování

# Keywords

OS Android, music player, prediction, reinforcement learning, popularity of songs, valuation


# Citace

Michal Tomášek: Inteligentní hudební přehrávač pro Android, bakalářská práce, Brno, FIT VUT v Brně, 2014

# Inteligentní hudební přehrávač pro Android

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Róberta Kalmára.

.....
Michal Tomášek
19. května 2014

## Poděkování

Rád bych tímto poděkoval vedoucímu práce Ing. Róbertu Kalmárovi za konzultace, zpětnou vazbu a odbornou pomoc. Dále bych rád poděkoval svojí rodině za podporu a všem, kteří se zúčastnili testování.

# Obsah


**1 Úvod** **3**

**2 Existující řešení** **4**
2.1 Poweramp Music Player . . . . . 5
2.2 PlayerPro Music Player . . . . . 5
2.3 Google Play Music . . . . . 6

**3 Operační systém Android** **7**
3.1 Aplikace . . . . . 7
3.1.1 Komponenty . . . . . 7
3.1.2 Komunikace komponent . . . . . 8
3.1.3 Zdroje . . . . . 8
3.1.4 Manifest . . . . . 8
3.2 Přehrávání médií . . . . . 9
3.2.1 Stavy objektu MediaPlayer . . . . . 9
3.2.2 Správa Audio Focus . . . . . 11

**4 Metriky** **12**

**5 Metody strojového učení** **13**
5.1 Charakteristiky problému . . . . . 13
5.2 Výběr kategorie metod . . . . . 13
5.3 Posilované učení . . . . . 15
5.4 Aplikace posilovaného učení . . . . . 15
5.4.1 Ohodnocovací funkce . . . . . 16
5.4.2 Odměňovací funkce . . . . . 19
5.4.3 Rozhodovací funkce . . . . . 20

**6 Návrh řešení** **22**
6.1 Mobilní aplikace . . . . . 22
6.2 Algoritmus predikce . . . . . 23
6.2.1 Ohodnocování . . . . . 24
6.2.2 Selekce . . . . . 25

**7 Optimalizace a srovnání algoritmů predikce** **26**
7.1 Možnosti optimalizace . . . . . 26
7.1.1 Sběr dat a offline zpracování . . . . . 27
7.1.2 Simulace provádění činů a online zpracování . . . . . 27

7.2 Metriky kvality ohodnocení skladeb . . . . . 28
7.3 Optimalizace . . . . . 29
7.3.1 Nalezení vhodných hodnot parametrů . . . . . 29
7.3.2 Srovnání jednotlivých algoritmů . . . . . 34
7.4 Aplikovaná vylepšení . . . . . 35

**8 Implementace aplikace** **37**
8.1 Struktura aplikace . . . . . 38

**9 Testování aplikace** **41**
9.1 Porovnání predikce s módem shuffle . . . . . 41
9.2 Uživatelské testování . . . . . 42

**10 Závěr** **45**

**A Obsah CD** **48**

# Kapitola 1

# Úvod

V současné době je nabídka běžných hudebních přehrávačů pro OS (operační systém) Android dostatečně zaplněna. Mezi nejlepší patří například *Poweramp*, *PlayerPro* nebo také *Google Play Music*. Vynikají především svým jednoduchým ovládáním, příznivou cenou a příjemným uživatelským rozhraním. Požadavky uživatelů se však stále zvyšují, přičemž je kladen důraz na nadstandardní funkce těchto aplikací. Náročnost klientely, z hlediska požadavků na výrobek, je nutno uspokojit, a tak je třeba vytvářet stále nové aplikace umožňující naplnění těchto požadavků. Management vývoje takovýchto aplikací tedy směřuje k nabídce co nejzajímavějších návrhů. Jeden z možných nadstandardních návrhů lze spatřit v „chytrých“ aplikacích zaměřující se na skutečné potřeby uživatele. Konkrétně v zaměření na jeho oblíbené hudební skladby, které dokáže aplikace automaticky rozpoznat pouze v závislosti na chování uživatele. Díky této aplikaci by uživatel nebyl nucen manuálně přepínat méně oblíbené skladby.

Cílem této práce je nastínit teoretické základy a možnosti pro vývoj hudebního přehrávače, rozšířeného o predikci, pro operační systém Android. Stěžejním úkolem pak je implementovat tento přehrávač tak, aby byla jeho schopnost predikce pozorovatelně přínosná. Výstupem této práce je poukázat na možnosti nástavby běžných hudebních přehrávačů v OS Android.

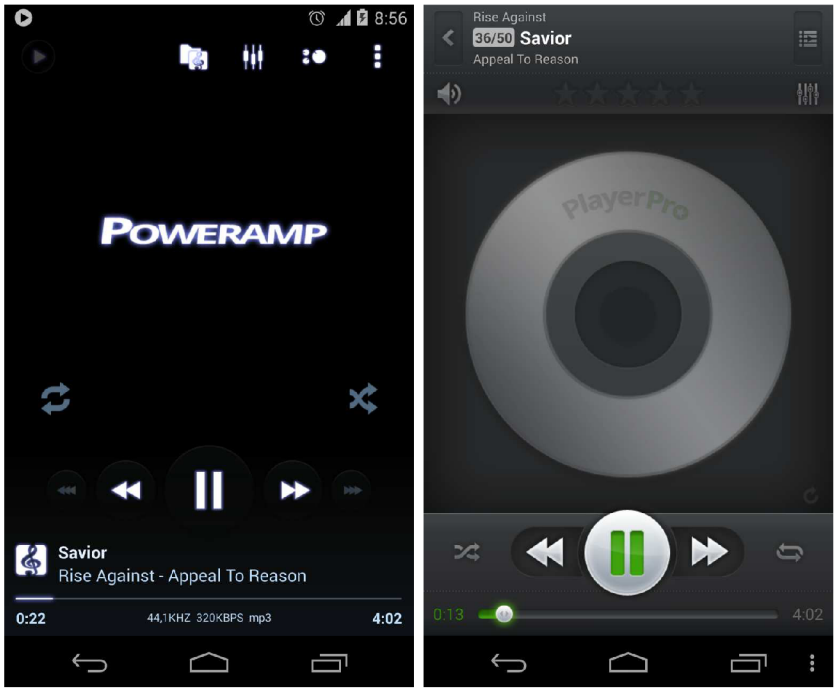
V této práci je nejprve poukázáno (v kapitole 2) na současný stav hudebních přehrávačů. Text následující kapitoly 3 se zabývá vývojem aplikací pro OS Android. Tato kapitola je zaměřena především na základní principy vývoje aplikací pod tímto OS, ale také na důležité informace přímo související s vývojem hudebního přehrávače. V kapitole 4 jsou zmíněny metriky, které lze pro rozpoznávání popularity skladeb použít. Kapitola 5 pojednává o řešeném problému z pohledu metod strojového učení. Jsou zde nastíněny konkrétní vlastnosti řešeného problému a také zde probíhá výběr kategorie metod strojového učení a aplikace vybrané kategorie na řešený problém. Následující kapitola 6 provádí shrnutí dosavadních zjištění a návrh možného řešení. Obsah této kapitoly se soustředí také na návrh algoritmu predikce, pro který existuje několik variant. Jednotlivé přijaté varianty algoritmu predikce byly optimalizovány a srovnány. Možnosti optimalizace, metriky pro určení kvality ohodnocení skladeb a samotný průběh včetně výsledků optimalizace je obsažen v kapitole 7. Kapitola 8 poté popisuje implementaci výsledné aplikace. Tento popis obsahuje i grafické znázornění v podobě diagramu tříd finální implementace. V kapitole 9 je uvedeno testování implementované aplikace. Toto testování se zabývá jak algoritmem predikce, tak i vlastní aplikací, kterou bylo zapotřebí vyzkoušet v praxi na různých zařízeních a různých verzích OS Android. Práci uzavírá kapitola 10 se závěrečným zhodnocením dosažených výsledků a popisem možných zlepšení aplikace či algoritmu predikce.

# Kapitola 2

# Existující řešení

V současné době je nabídka hudebních přehrávačů pro OS Android rozsáhlá. Existující aplikace se typicky snaží zaměřit především na snadnost ovládání a estetiku uživatelského rozhraní. Co se nadstandardních funkcí týče, většinou se aplikace zaměřují především na kvalitní ekvalizér, podporu co největšího množství formátů audio souborů a v poslední době i možnost zobrazení textu přehrávané skladby. Možnost predikce by tedy mohla zvýšit uživatelský komfort při používání přehrávače a tím by daný přehrávač mohl potenciálně získat konkurenční výhodu.

Níže popsané aplikace tedy nemají implementovaný žádný mechanismus predikce, s výjimkou základního hudebního přehrávače OS Android. U zmíněného přehrávače bude jeho mechanismus predikce popsán podrobněji.

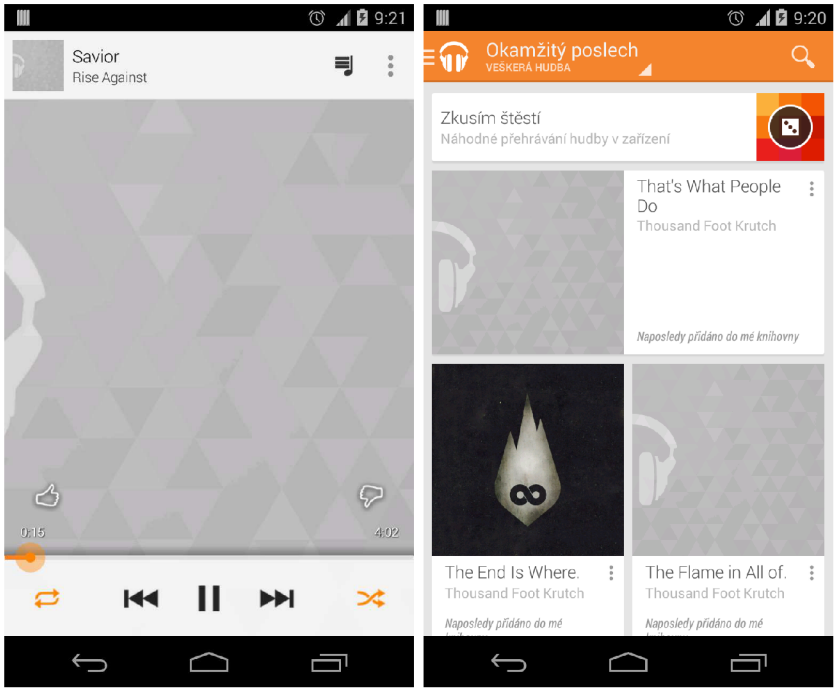



Obrázek 2.1: Snímky obrazovky. Poweramp vlevo, PlayerPro vpravo.

## 2.1 Poweramp Music Player

*Poweramp Music Player* [10] je jedním z nejpopulárnějších hudebních přehrávačů pro OS Android. Díky svému intuitivnímu ovládání a možností upravovat vzhled přehrávače pomocí množství stažitelných vzhledů si tento přehrávač získal svou vysokou popularitu. V oblasti nadstandardních funkcí obsahuje propracovaný ekvalizér, podporuje velké množství formátů audio souborů, umí zobrazit text přehrávané skladby, a také např. stáhnout ze serveru obrázek alba, pokud pro dané album není k dispozici.

Tento přehrávač je k dispozici ve tzv. „zkušební“ verzi, která dovoluje vyzkoušet si aplikaci se všemi funkcemi po dobu 15 dní. Poté je nutné zakoupit placenou verzi. U placené verze občas vzniká problém u zařízení, které nemají stálý nebo alespoň častý přístup k internetu. Placená verze totiž jednou za určitou dobu kontroluje platnost zakoupení, pro případ krádeže softwaru. Pokud v době kontroly, která probíhá typicky při zapnutí přehrávače, nemá zařízení přístup k internetu, může dojít k zablokování přehrávače, dokud se kontrola neprovede.




Obrázek 2.2: Snímky obrazovky aplikace Google Play Music. Obrazovka s predikcí vpravo.

## 2.2 PlayerPro Music Player

Další z velmi oblíbených hudebních přehrávačů je *PlayerPro Music Player* [9]. Opět obsahuje možnost zobrazit si text přehrávané skladby, upravovat vzhled aplikace, a také stahovat obrázky alb. Pro tento přehrávač existují i rozšíření, která se netýkají pouze vzhledu. Jedna z nepříliš obvyklých funkcí je například možnost úpravy popisků skladby přímo v přehrávači. Také je zde navíc podpora synchronizace se servery, které se snaží uchovávat poslechové návyky.

*PlayerPro Music Player* lze opět získat ve „zkušební“ verzi. Ta je u tohoto přehrávače omezena na 10 dní. Po uplynutí této zkušební doby se přehrávač zablokuje do zakoupení placené verze. U tohoto přehrávače však nevzniká problém s kontrolou platnosti placené verze, protože není na rozdíl od přehrávače *Poweramp Music Player* stahována dodatečně pouze aplikace, která odblokuje neplacenou verzi, nýbrž celá aplikace bez časového omezení.

## 2.3 Google Play Music

*Google Play Music* [8] je hudební přehrávač, který je součástí balíku aplikací od firmy Google Inc., a měl by tedy být předinstalován na většině zařízení s OS Android. Hlavní výhodou tohoto přehrávače oproti ostatním je, že nabízí přístup k online hudební knihovně. To platí za předpokladu, že je tato funkce podporovaná v dané zemi a přehrávač má přístup na internet.

Další neobvyklou funkcí je snaha predikce dle hudební knihovny a používání přehrávače. Tato predikce však ovlivňuje přehrávač pouze tím způsobem, že zde přibyla nová obrazovka aplikace, která zobrazuje doporučené skladby pro další poslech. Predikce tedy nijak přímo neovlivňuje výběr následující skladby při přehrávání. Dává uživateli pouze možnost výběru ze skladeb, které by teoreticky měly být jeho oblíbené. Tento výběr ale musí uživatel provést sám a ručně, tudíž zde nejspíše není snaha o zvýšení komfortu používání přehrávače tím, že se sníží množství interakcí s přehrávačem, nýbrž snaha o nabídnutí zajímavé funkce, která v této podobě může některým uživatelům vyhovovat.

# Kapitola 3

# Operační systém Android

Operační systém Android je v současné době velmi rozšířený. Jedná se o otevřený systém. Vývojem pod tímto OS se zabývá široká základna vývojářů, díky čemuž lze velké množství řešení problémů dohledat a vývoj se tak stává snazším. K vývoji aplikací pro zařízení s OS Android je potřeba mít sadu nástrojů, která je obsažena v balíku Android SDK (Software Development Kit). K těmto nástrojům lze přistupovat přímo pomocí IDE (Integrated Development Environment) Eclipse, které je součástí tohoto balíku. Je doporučené pro vývoj používat zmíněné IDE, neboť je jednodušší volat nástroje, potřebné při vývoji, z něj, než je spouštět z příkazové řádky. Zdrojové kódy jsou psány v jazyce Java, který není nijak speciálně upraven, takže programátor, který má předchozí zkušenosti s tímto programovacím jazykem se nemusí nijak speciálně připravovat. Musí se pouze obeznámit se způsobem práce v OS Android, a také se životním cyklem aplikace a jejích komponent. Má také k dispozici mnoho knihoven, které nabízejí již hotové řešení častých problémů. Ty jsou přímo součástí operačního systému, takže není třeba je jakkoliv dodatečně získávat [1].

## 3.1 Aplikace

Aplikace jsou sestaveny jako kombinace různých komponent, které mohou být vyvolávány nezávisle na sobě. Mezi komponentami (i různých aplikací) může probíhat asynchronní komunikace prostřednictvím zpráv. Aplikace může být tvořena nejen zdrojovými kódy, ale může využívat i tzv. zdroje. To mohou být například multimediální soubory, soubor s texty, které se používají v uživatelském rozhraní pro komunikaci s uživatelem (lze tedy snadno vytvořit multilingvální aplikaci), soubor s definicí rozložení prvků na zobrazovací ploše, a další. Každá aplikace pro OS Android musí také obsahovat tzv. manifest, který poskytuje systému důležité informace o dané aplikaci [2].

### 3.1.1 Komponenty

Komponenty, které se podílejí na implementaci jednotlivých funkcí aplikace, se rozdělují do těchto skupin [15]:

- Aktivity (*Activities*) – Aktivita prezentuje vizuální uživatelské rozhraní. Poskytuje zobrazovací plochu, na které jsou umístěny ovládací prvky uživatelského rozhraní. Aktivita by například mohla prezentovat seznam položek, ze kterých může uživatel vybírat.

- Poskytovatelé obsahu (*Content providers*) – Poskytovatel obsahu vytváří specifickou množinu dat aplikace, která je přístupná i ostatním aplikacím. Tato data mohou být uložena v souborovém systému, *SQLite* databázi, nebo jiným způsobem, který zachová jejich logické uspořádání.
- Služby (*Services*) – Služba nemá vizuální uživatelské rozhraní, ale pracuje na pozadí po neurčitou dobu. Například může služba na pozadí přehrávat hudbu a během této doby se může uživatel věnovat jiným záležitostem.
- Příjemci broadcast (*Broadcast receivers*) – Příjemce broadcast je komponenta, která pouze přijímá a reaguje na broadcast oznámení. Většina těchto oznámení pochází z operačního systému. Konkrétně se může jednat o oznámení vybité baterie, změny nastavení používaného výchozího jazyka, změny časové zóny, a další.

### 3.1.2 Komunikace komponent

Komponenty mezi sebou mohou asynchronně komunikovat pomocí tzv. úmyslů (*intents*) [15].

- Úmysl (*Intent*) – Aktivity, služby a příjemci broadcastu jsou aktivováni asynchronní zprávou, nazývající se úmysl. Úmysl je objekt typu `Intent`, který nese obsah dané zprávy. Z jedné komponenty tedy lze zapnout další pomocí úmyslu. Dokonce lze takto zapnout i komponenty jiné aplikace. Tento model poskytuje více vstupních bodů pro jednu aplikaci. Zároveň povoluje jakékoliv aplikaci, aby vystupovala jako uživatelova výchozí, takže může vyvolat ostatní aplikace. Příklad může být sdělení požadavku aktivitě, aby zobrazila uživateli vybraný obrázek.
- Filtr úmyslů (*Intent Filter*) – Objekt úmyslu může explicitně jmenovat cílovou komponentu, které má být doručen. Pokud tomu tak je, operační systém vyhledá tuto komponentu a aktivuje ji. Pokud není cíl explicitně jmenován, musí být nalezena nejlepší komponenta, která na tento úmysl umí reagovat. A právě v tomto případě je nalezení nejlepší komponenty prováděno pomocí porovnávání objektu úmyslu a filtrů úmyslů potenciálních cílů. Filtr úmyslů komponenty tedy informuje operační systém, jaké druhy úmyslů dokáže daná komponenta zpracovávat.

### 3.1.3 Zdroje

Jedná se o externí části aplikace, které ji typicky parametrizují. Důvod proč je oddělit od kódu aplikace je pro jejich snadnější údržbu, ale také proto, že některé zdroje mohou být binární, nikoliv textové. Výhoda tohoto oddělení spočívá také v možnosti poskytovat alternativy zdrojů pro další specifické konfigurace různých zařízeních. Může se jednat například o různé komunikační jazyky, nebo rozdílné rozložení ovládacích prvků podle velikosti obrazovky zařízení. Všechny tyto zdroje jsou umístěny do speciální složky v projektu a lze je sdružovat do podsložek, díky čemuž jsou zdroje rozdělené pro určité konfigurace zařízení. Typické zdroje jsou například obrázky, rozložení ovládacích prvků, nebo textové řetězce, které se zobrazují v uživatelském rozhraní [4].

### 3.1.4 Manifest

Manifest poskytuje důležité informace o aplikaci a ukládá se do souboru `AndroidManifest.xml` v kořenovém adresáři aplikace. Tyto informace musí mít operační systém k dispozici před

vlastním spuštěním aplikace. Tento soubor se stará například o tyto záležitosti [3]:

- Pojmenovává Java balík aplikace. Toto jméno slouží jako unikátní identifikátor dané aplikace.
- Popisuje komponenty aplikace.
- Určuje, které procesy budou hostit komponenty aplikace.
- Deklaruje, která oprávnění musí aplikace mít pro přístup ke chráněným částem API (Application Programming Interface) a pro interakci s ostatními aplikacemi.
- Deklaruje oprávnění pro ostatní aplikace, které chtějí interagovat s komponentou aplikace.
- Deklaruje minimální verzi Android API, která je aplikací vyžadována.
- Vyjmenovává knihovny, se kterými musí být aplikace spojena.

## 3.2 Přehrávání médií

Operační systém Android obsahuje API `MediaPlayer` pro práci s multimédii. Podporuje většinu běžných formátů mediálních souborů, takže lze tyto soubory přímo přehrávat, či zobrazovat. Zmíněné API zahrnuje dvě třídy. Hlavní třídu `MediaPlayer`, která přehrává zvukové soubory i video soubory, a třídu `AudioManager`, která spravuje audio-vstupy/výstupy zařízení [6].

Mediální soubory, které objekt typu `MediaPlayer` dokáže přehrát, mohou být uloženy jako [6]:

- lokální zdroje aplikace
- soubory v zařízení adresované pomocí interní URI (Uniform Resource Identifier)
- soubory umístěné mimo zařízení adresované pomocí externí URL (Uniform Resource Locator)

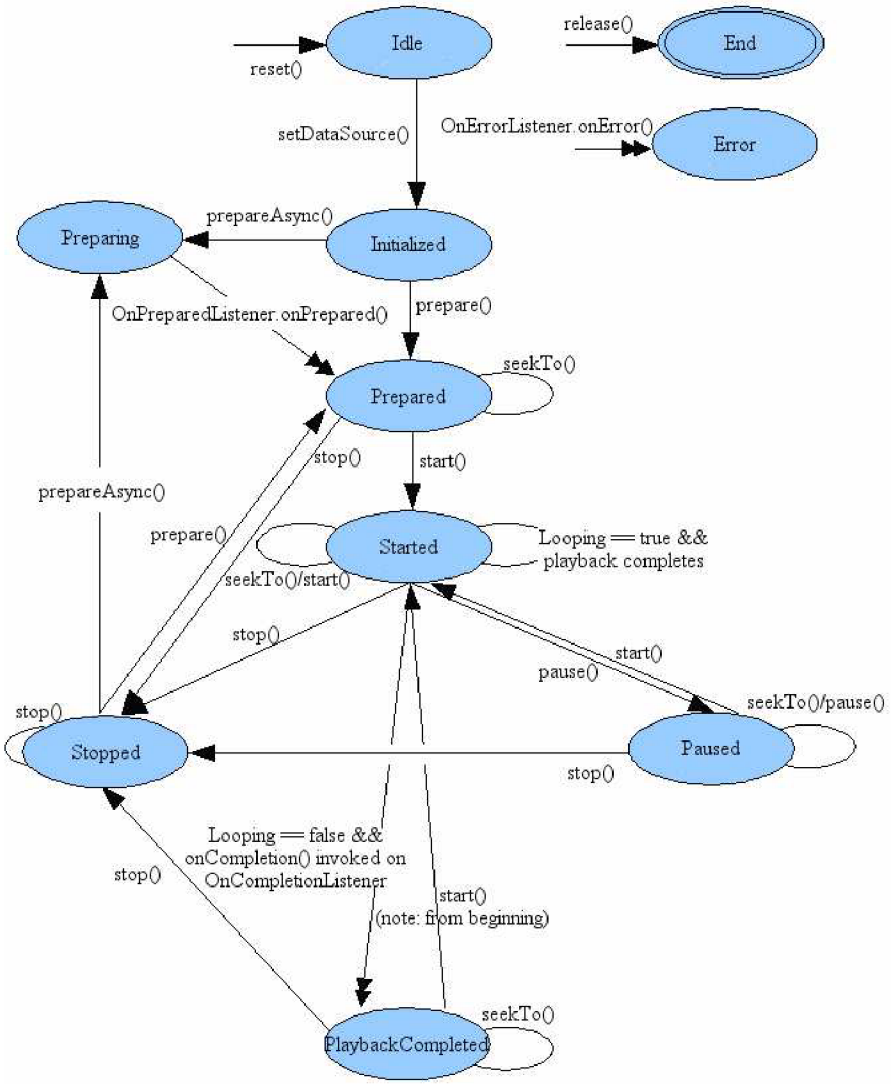
### 3.2.1 Stavy objektu MediaPlayer

Je důležité brát v úvahu to, že objekt `MediaPlayer` je stavový. Tento fakt je důležitý z toho důvodu, že některé operace, které lze nad tímto objektem provádět, jsou validní pouze v určitých stavech. V případě použití nevalidní operace může dojít k výjimce vyvolané operačním systémem [6].

Na obrázku 3.1 jsou vidět jednotlivé stavy objektu `MediaPlayer`, a také metody, které provádějí přechody mezi těmito stavy.

Z tohoto diagramu lze tedy například vyčíst, že v době po vytvoření nové instance třídy `MediaPlayer`, se tato instance nachází ve stavu *Idle*. Pomocí volání metody `setDataSource()` lze změnit stav objektu na stav *Initialized*. Poté musí být provedena fáze přípravy, a to pomocí metody `prepare()` nebo `prepareAsync()`.

Po provedení přípravy je objekt ve stavu *Prepared*, ze kterého již lze zavolat metoda `start()` pro přehrání zvoleného mediálního souboru. Nyní se může objekt pohybovat mezi stavy *Started*, *Paused* a *PlaybackCompleted* díky metodám `start()`, `pause()` a `seekTo()`.




Obrázek 3.1: Stavový diagram objektu `MediaPlayer` [7].

Avšak pokud je zavolána metoda `stop()`, nesmí se volat metoda `start()`, dokud se opět neprovede fáze přípravy.

Před přehráváním musí být vždy provedena fáze přípravy. V této fázi se systém pokouší získat alespoň část souboru, který byl zvolen jako zdroj pro přehrávání. Pokud by tento soubor byl umístěn například na vzdáleném serveru, může se stát, že tato fáze bude trvat dlouhou dobu, a to by mohlo mít za následek nereagující uživatelské rozhraní.

Z tohoto důvodu podporuje objekt `MediaPlayer` i tzv. asynchronní přípravu. Tato příprava je prováděna v novém vlákně, nikoliv v hlavním vlákně, které provádí obsluhu událostí uživatelského rozhraní. O dokončení fáze přípravy je hlavní vlákno informováno asynchronní zprávou [6].

### 3.2.2 Správa Audio Focus

Tato sekce čerpá ze zdroje [6].

Přestože může být aktivní v daný čas vždy pouze jedna aktivita, OS Android umožňuje zpracování více úloh současně. To znamená, že mohou jednotlivé aplikace (typicky jejich služby), které vyžadují audio výstup, o tento výstup soupeřit. Jelikož je audio výstup pouze jeden, je nutné, aby přístup k němu byl řízen. Mohla by totiž vzniknout například taková situace, že uživatel poslouchá hudbu a jiná aplikace chce uživatele informovat o něčem důležitém. Pokud není k audio výstupu řízen přístup, může se stát, že uživatel informační tón, kvůli hlasité hudbě neuslyší. Proto existuje mechanismus, který aplikacím nabízí možnost vyjednání si používání audio výstupu. Tento mechanismus se nazývá *Audio Focus*.

Před použitím audio výstupu je tedy nutné, aby si aplikace vždy prvně požádala o *audio focus*. Začít používat audio výstup může poté, co získá *audio focus*, avšak stále musí sledovat jeho změny. Pokud totiž v průběhu používání audio výstupu *audio focus* aplikace ztratí, měla by ihned zastavit přehrávání, nebo snížit úroveň hlasitosti svého výstupu. V případě ztráty *audio focus* lze pokračovat v přehrávání, nebo lze provést návrat hlasitosti na původní hodnotu, až po jeho opětovném získání.

*Audio focus* má kooperativní charakter. To znamená, že je očekáváno dodržování pokynů *audio focus* ze strany aplikací. Tyto pravidla však nejsou vynucovány. Pokud bude nějaká aplikace používat hlasitě audio výstup i po ztrátě *audio focus* systém tomu nijak nezabrání. Takové aplikace jsou ale často považovány za nekvalitní, protože je jejich chování matoucí.

*Audio focus* je součástí třídy `AudioManager` a může být vůči aplikaci ve stavu:

- *AUDIOFOCUS_GAIN* – získán *audio focus*
- *AUDIOFOCUS_LOSS* – *audio focus* ztracen, pravděpodobně na delší dobu
- *AUDIOFOCUS_LOSS_TRANSIENT* – *audio focus* ztracen, pravděpodobně krátce
- *AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK* – *audio focus* ztracen, pravděpodobně krátce, lze pokračovat v přehrávání na nižší úrovni hlasitosti

# Kapitola 4

# Metriky

Při práci algoritmu predikce, který bude zajišťovat vlastní inteligenci přehrávače, bude nutné stanovovat oblíbenost respektive neoblíbenost hudebních skladeb umístěných v zařízení uživatele. Aby bylo možné vyčíslit tuto popularitu skladeb, bude zapotřebí nalézt vhodné metriky. Tyto metriky tedy mohou sloužit pro výpočet ohodnocení konkrétní skladby. Ohodnocení pak bude přímo reprezentovat popularitu dané skladby u uživatele.

Metriky, které lze použít pro výpočet ohodnocení hudebních skladeb:

- počet přehrání dané skladby
- počet přeskočení dané skladby
- počet výběrů z knihovny dané skladby
- celkový počet činů souvisejících s danou skladbou
- častost určitých činů souvisejících s danou skladbou

Metrika „častost určitých činů souvisejících s danou skladbou“ by vyjadřovala například jak často uživatel tuto skladbu vybírá z knihovny. Tuto četnost by mohla vyjadřovat pomocí rozdílu aktuálního času a času posledního výběru z knihovny. Problematika použití takovéto metriky tkví v tom, že pro zachování přesných hodnot, by musela být přepočítávána všechna ohodnocení každou časovou jednotku, která by byla pro vyjádření častosti použita.

Při použití ostatních zmíněných metrik by žádné potíže nastat neměly, jelikož všechny tyto metriky lze implementovat jako počítadlo reprezentované celočíselnou proměnnou. Aktualizace libovolného z počítadel by probíhala vždy při konkrétní události vztahující se k dané skladbě, které počítadlo náleží. Takže by nevznikal žádný problém s aktualizací hodnot, jako u metriky „častost určitých činů souvisejících s danou skladbou“.

Hlavní rozdíl a zároveň i důvod problematiky použití poslední ze zmíněných metrik je ten, že událost u této metriky nastává pro všechny skladby v zařízení současně. Konkrétně se jedná o událost oznamující změnu aktuálního času. U ostatních metrik naopak platí, že událost se vždy vztahuje právě vůči jedné skladbě v zařízení.

Existuje také patent zabývající se právě problematikou ohodnocování hudebních skladeb. Používá algoritmus, který využívá především častosti přehrávání skladeb, ale je rozšířen i o další informace. O jaké informace se jedná není v tomto patentu napsáno [17].

Stupnice, které se běžně pro určování popularity používají, lze nalézt typicky pouze ty, kde hodnocení zadává přímo uživatel. Obvykle je hodnocení na stupnici 1 až 5, nebo 1 až 100, kde první stupnice bývá reprezentována hodnocením pomocí „hvězdiček“ a druhá reprezentuje obvykle procenta [16].

# Kapitola 5

# Metody strojového učení

Před samotným výběrem metody strojového učení bylo zapotřebí prvně specifikovat charakteristiky daného problému, který byl za pomoci konkrétní metody řešen. Metody strojového učení se rozdělují do několika kategorií. A to podle způsobu, jak se stroj učí. Bylo tedy nutné prvně vybrat vhodnou kategorii metod pro daný problém.

## 5.1 Charakteristiky problému

U tohoto konkrétního problému platí, že informace budou získávány postupně. Prostředí a jeho odezvy se mohou měnit, přičemž algoritmus bude muset data, na základě kterých provádí rozhodování, upravovat v průběhu jejich používání. Jedná se totiž o trvalou úlohu, nikoliv o epizodní.

Trvalá úloha není rozdělena na epizody (menší pod-části). Pokud by se jednalo o epizodní úlohu, bylo by možné provádět úpravy dat až po dokončení probíhající epizody. Epizodní úloha je taková úloha, kde se neustále opakuje určitý úkol a lze tedy stanovit počáteční a koncový stav [20, s. 58].

Konkrétní charakteristiky problému tedy jsou:

- inkrementální – data pro učení jsou získávána průběžně postupem času [12]
- nestacionární – jedná se o proces, jehož statistické vlastnosti se mění [14]
- ergodický – existence nenulové pravděpodobnosti přechodu z jakéhokoliv stavu do jakéhokoliv jiného stavu [18, s. 467]
- trvalá úloha – interakce kontinuálně pokračuje bez omezení a není přirozeně dělitelná do identifikovatelných epizod [20, s. 58]

## 5.2 Výběr kategorie metod

Vhodná kategorie by měla obsahovat právě takové metody, jež lze aplikovat při řešení specifikovaného problému. Je však důležité, aby tyto metody byli i efektivní s danými charakteristikami problému, tzn., aby byla rychlost učení algoritmu co nejvyšší. Zároveň však nesmí dojít k zacyklení, kdy by algoritmus vybíral pouze malé množství skladeb s velmi vysokým ohodnocením a nikdy žádné jiné.

Výčet kategorií metod strojového učení [11, s. 19]:

- Učení s učitelem
  - generuje funkci, která mapuje vstupy na požadované výstupy
  - učený se učí z příkladů poskytovaných od informovaného externího učitele
- Učení bez učitele
  - modeluje množiny vstupů
  - na rozdíl od předchozí kategorie nemá k dispozici vyřešené trénovací příklady
- Kombinované učení
  - generuje mapovací funkci nebo klasifikátor
  - kombinuje použití vyřešených a nevyřešených příkladů
- Posilované učení
  - algoritmus se učí strategii, která určuje jak pracovat s prostředím
  - každá akce má nějaký dopad na prostředí poskytující zpětnou vazbu
  - tyto metody se učí z interakce s prostředím
- Transdukce
  - snaha predikovat výstup
  - podobné jako učení s učitelem, avšak nevytváříme mapovací funkci
  - používáme trénovací vstupy, výstupy, ale také nové vstupy
- Učení se učit
  - snaží se naučit své vlastní induktivní tendence/sklony
  - učí se z předchozích zkušeností

Jelikož je tento problém inkrementální a nestacionární, není možné efektivně použít žádnou z metod používající trénovací vzorky dat, nebo metody snažící se o klasifikaci do určitých množin.

Je zapotřebí použít právě takové metody, které se dokáží učit z interakce s prostředím, protože nejsou předem k dispozici žádná trénovací data. Také je důležité, aby se metody dané kategorie dokázali vyrovnat s nestacionárním prostředím. Nesmí tedy uvažovat zpětnou vazbu prostředí jako neměnnou a trvale platnou, protože na stejné akce může systém obdržet v různých časech různou zpětnou vazbu.

Nejvhodnější kategorií pro tento konkrétní problém je kategorie posilované učení. Tato kategorie se učí průběžně z interakce s prostředím, přičemž nevyžaduje žádná trénovací data. Většina metod z této kategorie se dokáže efektivně učit i v nestacionárním prostředí.

## 5.3 Posilované učení

Problém, který posilované učení řeší, je vždy rozložen na 4 části [20, s. 7]:

1. stavy
2. akce
3. strategie
4. model prostředí

Při řešení se tedy provádí prohledávání stavového prostoru, kdy mezi jednotlivými stavy provádíme přechody pomocí akcí. Volbu konkrétní akce v daném stavu určuje strategie. Model prostředí, který je jako jediná část nepovinný, poskytuje možnost plánování, jelikož napodobuje chování skutečného prostředí ve kterém se pracuje [20, s. 7-9].

U tohoto konkrétního problému budou stavy odpovídat skladbám, akce pak přechodům mezi skladbami a strategie mechanismu výběru skladby. V tomto případě model vytvářen nebude, protože modelovat chování člověka, reprezentujícího prostředí systému, deterministicky přesně nelze.

Typicky lze obecný problém řešit dvěma způsoby [20, s. 68-69]:

1. ohodnocováním stavů
2. ohodnocováním párů stav-akce

U tohoto problému lze říci, že je vhodnější provádět ohodnocování stavů, protože pro tento problém směřuje právě $n - 1$ akcí do každého z $n$ stavů. Pro celkový počet párů stav-akce tedy platí:

$$n \cdot (n - 1)$$

Důvod je ten, že z jakéhokoliv stavu lze provést přechod na jakýkoliv jiný stav.

U ohodnocování párů stav-akce by tedy vznikal problém ohodnocení tak velkého množství entit. Přestože by při přechodu na konkrétní skladbu byl vždy systém negativně odměněn, dokud by toto nebylo neprovedeno pro všechny akce ve všech stavech, které vedou na danou skladbu (kterých je $n - 1$), systém by nevěděl, že skladbu nemá vybírat.

Při ohodnocování stavů, by docházelo k ohodnocování každé ze skladeb. Z hlediska výpočetních a paměťových nároků je tedy tato varianta vhodnější.

## 5.4 Aplikace posilovaného učení

Během učení se bude muset vyhodnotit v nestacionárním prostředí především ohodnocení stavů. Stavy, akce a odměny budou předem dané. Je tedy potřeba implementovat ohodnocovací, odměňovací a rozhodovací funkci.

Ohodnocovací funkce bude mít za úkol vypočítat, na základě určitých parametrů, ohodnocení dané skladby. Z obecného pohledu se jedná o funkci ohodnocující stavy.

Odměňovací funkci bude realizovat uživatel, jelikož svými činy[1] bude pozitivně nebo negativně odměňovat systém. Odměny budou právě jedním z parametrů, které se podílejí na ohodnocení skladeb. Zároveň budou jejich hodnoty parametrizovatelné, což bude měnit

[1] Záměrně není použito slovo akce, aby bylo odlišeno to, co provádí uživatel, od toho, co provádí systém, kde akce reprezentují přechod mezi skladbami.

chování implementovaného systému. Odměny mají za úkol vést systém k požadovanému chování.

Rozhodovací funkce bude realizovat výběr skladby pro přehrání. Realizuje tedy tzv. strategii, která bude provádět konkrétní přechody ve stavovém prostoru skladeb.

V následujících oddílech budou jednotlivé části řešení popsány podrobněji. Především budou vyjmenovány jednotlivé varianty řešení, které lze potenciálně použít.

### 5.4.1 Ohodnocovací funkce

V této sekci budou popsány dva způsoby jak řešit ohodnocování skladeb:

1. použití konkrétních vzorců pro výpočet ohodnocení
2. použití kompletních metod posilovaného učení

Rozdíl mezi těmito způsoby je především ten, že první z nich umožňuje použití libovolné strategie (rozhodovací funkce). Naopak druhý způsob přesně definuje jak ohodnocovací tak rozhodovací funkci. V tomto oddílu však budou popisovány pouze informace týkající se ohodnocování.

#### Konkrétní vzorce pro výpočet ohodnocení

První variantou je použití obecného vzorce pro výpočet ohodnocení stavů posilovaného učení jakožto ohodnocovací funkce pro výpočet ohodnocení skladeb. Tento vzorec bere v potaz pouze hodnoty odměn. Právě ty totiž určují odhad ohodnocení daného stavu. Jedná se vlastně o vážený průměr předchozích odměn a zásadní vliv má i počáteční odhad ohodnocení stavu. Vzorec je následující [20, s. 38]:

$$Q_k = (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha (1-\alpha)^{k-i} r_i$$

V tomto vzorci je $Q_k$ aktuální odhad ohodnocení, $Q_0$ je počáteční odhad ohodnocení, $r_i$ je *i-tá* odměna a $\alpha$ je konstantní parametr, který určuje velikost kroku, a platí pro něj $0 < \alpha \leq 1$. Tento vztah by ale vyžadoval kompletní historii odměn daného stavu/skladby. Tyto ohodnocení však lze počítat i inkrementálně pomocí tohoto vzorce [20, s. 38]:

$$Q_{k+1} = Q_k + \alpha [r_{k+1} - Q_k]$$

V tomto vzorci značí $Q_{k+1}$ nový odhad ohodnocení, $Q_k$ je aktuální odhad ohodnocení, $r_{k+1}$ nově získanou odměnu a $\alpha$ je parametr velikosti kroku. Výpočty tedy musí být prováděny inkrementálně, protože jinak by výpočty ohodnocení byly velmi náročné a zdlouhavé pro dlouhou historii záznamů.

Zmíněný vzorec definuje parametry, které značně ovlivňují kvalitu a rychlost učení tohoto algoritmu. Jedná se o $Q_k$, které má při prvním ohodnocování funkci počátečního ohodnocení, dále parametr alfa, a také výše jednotlivých odměn. To znamená, že před srovnáním této varianty s jinou je nutné prvně nalézt nevhodnější hodnoty pro zmíněné parametry.

U této varianty ohodnocovací funkce platí, že nejsou přímo používány žádné metriky, ale pouze odměny od uživatele. Pracuje se především na principu akce-reakce, kdy akcí je obdržení odměny od uživatele a reakcí je přepočítání aktuálního ohodnocení pomocí konstanty $\alpha$, odměny a aktuálního ohodnocení. V systému se tedy neuchovávají žádné

dodatečné proměnné jako celkový počet činů uživatele související s konkrétní skladbou či jiné.

Druhá varianta je použití tzv. Wilsonova (skórového) intervalového odhadu (*Wilson score interval*) [19]. Stanovíme-li tento vztah:

$$p = \frac{x}{n}$$

kde $x$ je počet kladných činů uživatele a $n$ je počet všech činů uživatele, pak Wilsonův intervalový odhad používá vzorec:

$$\pi_D = \frac{p + \frac{z^2_{\alpha/2}}{2n} - z_{\alpha/2}\sqrt{\frac{p\,(1-p) + \frac{z^2_{\alpha/2}}{4n}}{n}}}{1 + \frac{z^2_{\alpha/2}}{n}}$$

V tomto vzorci je $z_{\alpha/2}$ $\alpha$-kvantil normovaného normálního rozdělení. Pomocí tohoto vzorce probíhá výpočet odhadu spodní hranice intervalu a právě ta lze použít jako ohodnocení skladby [19].

Tento odhad existuje i s tzv. korekcí na spojitost (*continuity correction*). Použití korekce na spojitost vede ke konzervativnímu intervalovému odhadu. Takový odhad mírně nadhodnocuje všechna ohodnocení, takže by neměly vznikat tak ostré rozdíly mezi ohodnoceními skladeb. Vzorec pro výpočet spodní hranice intervalu pak vypadá takto [19]:

$$\pi_D = \max\left\{0;\, \frac{2np + z^2_{\alpha/2} - z_{\alpha/2}\sqrt{z^2_{\alpha/2} - \frac{1}{n} + 4np\,(1-p) + (4p-2)} + 1}{2\left(n + z^2_{\alpha/2}\right)}\right\}$$

Tento vzorec lze tedy považovat za třetí variantu algoritmu pro výpočet ohodnocení. Proměnné v uvedeném vzorci odpovídají svým významem proměnným ze vzorce předchozího.

Druhá i třetí varianta ohodnocovací části vyžadují ke své činnosti dvě metriky:

- celkový počet kladných činů souvisejících s danou skladbou
- celkový počet činů souvisejících s danou skladbou

Obě lze realizovat jednoduchým počítadlem. Tyto dvě varianty tedy vyžadují ukládání tří hodnot do paměti. Jedna z nich reprezentuje aktuální ohodnocení skladby, a druhé dvě slouží pro uložení používaných metrik pro výpočet tohoto ohodnocení. Důvod, proč ukládat spolu s metrikami i vypočítanou hodnotu aktuálního ohodnocení je ten, že při selekci by pak muselo dojít k výpočtům všech ohodnocení pro možnost relevantního výběru. Takováto operace by mohla být velmi zdlouhavá. Proto je společně s metrikami ukládáno i aktuální ohodnocení, přestože je z těchto metrik vypočtené.

### Kompletní metody posilovaného učení

Pro tyto metody platí, že definují algoritmus zajišťující výpočet ohodnocení i strategii. Ohodnocovací funkce je většinou jádrem celé metody, protože strategie, neboli rozhodovací funkce, se typicky blíží k tzv. *greedy* strategii. *Greedy* strategie je taková strategie, která provádí pouze *greedy* akce. Modifikovaná verze *greedy* strategie a *greedy* akce budou popsány v sekci 5.4.3.

Vybrané třídy metod, či metody posilovaného učení, jejichž jádrem je právě řešení ohodnocovací funkce [20]:

- třída metod *Dynamic Programming*
  - důležitost spíše z teoretického hlediska – používají se pro vysvětlování pokročilejších metod
  - klíčovou myšlenkou je použití ohodnocovací funkce pro hledání dobrých strategií
  - vyžadován perfektní model prostředí
  - výpočetně náročné
- třída metod *Monte Carlo*
  - není vyžadována znalost prostředí
  - řešení problému pomocí průměrování vzorků výstupu
  - definovány pouze pro epizodní úlohy – pouze po dokončení epizody probíhá aktualizace ohodnocení a strategie
  - učení je inkrementální ve smyslu epizoda za epizodou
- třída metod *Temporal-Difference*
  - metoda *TD(0)*
    - $V(s_t) \leftarrow V(s_t) + \alpha\big[r_{t+1} + \gamma\, V(s_{t+1}) - V(s_t)\big]$,
      kde $V(s_t)$ je ohodnocení stavu $s$ v čase $t$, $\alpha$ je parametr kroku, $r_{t+1}$ je odměna v čase $t+1$ a $\gamma$ je tzv. *discount* parametr (určuje jak moc jsou stavy na sebe závislé)
    - nejjednodušší z TD metod
    - ohodnocují se stavy
    - stavy jsou na sobě navzájem závislé (do výpočtu ohodnocení se přidává i ohodnocení následujícího stavu)
  - metoda *Q-Learning*
    - $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$,
      kde $Q(s_t, a_t)$ je ohodnocení páru stav-akce v čase $t$, $\alpha$ je parametr kroku, $r_{t+1}$ je odměna v čase $t+1$, $\gamma$ je *discount* parametr a $\max\, Q(s_{t+1}, a)$ vybírá z následujícího stavu takovou akci, aby daný pár stav-akce měl nejvyšší ohodnocení
    - ohodnocují se páry stav-akce
    - páry stav-akce jsou na sobě navzájem závislé (do výpočtu ohodnocení se přidává i ohodnocení nejlépe ohodnoceného páru stav-akce z následujícího stavu)
    - ohodnocovací funkce přímo aproximuje optimální ohodnocovací funkci
    - nezávislost na používané strategii
  - metoda *SARSA*
    - $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_{t+1} + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\big]$,
      kde jediný rozdíl oproti předchozímu vzorci je $Q(s_{t+1}, a_{t+1})$, takže se přičítá část ohodnocení následujícího provedeného páru stav-akce

    * ohodnocují se páry stav-akce
    * páry stav-akce jsou na sobě navzájem závislé (do výpočtu ohodnocení se přidává i ohodnocení následujícího páru stav-akce)
    * konvergenční vlastnosti závisí na povaze závislosti strategie na ohodnocovací funkci
  - metoda *Actor-Critic*
    * $\delta_t = r_{t+1} + \gamma\, V(s_{t+1}) - V(s_t)$,
      kde $\delta_t$ je tzv. TD chyba, $r_{t+1}$ je odměna v čase $t+1$, $\gamma$ je *discount* parametr a $V(s_t)$ je ohodnocení stavu $s$ v čase $t$
    * pro ohodnocování využívá TD chybu
    * ohodnocují se stavy
    * stavy jsou na sobě navzájem závislé (do výpočtu TD chyby se přidává i ohodnocení následujícího stavu)
    * dokáže se naučit stochastické strategie
    * skládá se ze dvou částí, kdy první část (*Actor*) provádí akce a druhá část (*Critic*) provádí výpočet TD chyby
  - metoda *R-Learning for Undiscounted Continuing Tasks*
    * $\widetilde{V}^{\pi}(s) = \sum_{k=1}^{\infty} E_{\pi}\{r_{t+k} - \rho^{\pi} | s_t = s\}$,
      kde $\widetilde{V}^{\pi}(s)$ je ohodnocení stavu $s$ pro strategii $\pi$, $E_{\pi}$ je očekávaná odměna, $r$ označuje skutečnou odměnu a $\rho^{\pi}$ je průměrná očekávaná odměna pro konkrétní časový okamžik.
    * ohodnocování je definováno relativně k průměrné očekávané odměně
    * experimentální metoda
    * vhodná pro trvalé úlohy
    * konkrétní hodnoty se porovnávají s průměrnou hodnotou, a tím je určena vhodnost či nevhodnost provedené akce

- upravené metody používající tzv. *eligibility traces*
  - metoda *The Backward View of TD(λ)* pro $\lambda = 0$
    * $\Delta V_t(s) = \alpha \delta_t e(s)$,
      kde $\alpha$ je parametr velikosti kroku, $\delta_t$ je TD chyba a $e(s)$ je *eligibility trace* pro stav $s$, která reprezentuje před jakou dobou byl stav navštíven
    * ohodnocení se upravuje dle *eligibility traces*
    * nastavením parametru $\lambda$ na nula byla zrušena závislost stavů na sobě, protože je na základě poslední obdržené odměny ohodnocován pouze předchozí stav
    * dokáže pracovat i v případě průběžných zápisů změn hodnot, důležitých pro rozhodování (tzv. *on-line* přístup)

### 5.4.2 Odměňovací funkce

Nejdůležitější činy uživatele jsou:

- přehrání skladby do konce

- výběr skladby z hudební knihovny
- přechod na jinou skladbu

Tyto činy jsou důležité z toho důvodu, že jimi uživatel dává najevo, zda danou skladbu má nebo nemá rád. Pokud skladbu uživatel přeskočí, je tento čin považován za negativní odměnu pro systém. V případě druhých dvou činů se jedná o činy s pozitivní odměnou, kde by výběr z hudební knihovny měl mít ještě vyšší odměnu pro danou skladbu, než-li přehrání skladby do konce, protože si danou skladbu uživatel sám aktivně vyhledal.

Odměňovací funkci bude, jak již bylo řečeno, provádět uživatel. Zbývá tedy, co se implementace týče, jen vhodně zvolit odměny pro zmíněné činy uživatele. Výše těchto odměn může mít u ohodnocovacích funkcí velký vliv. Toto se týká především funkcí, používající pro výpočet ohodnocení stavu pouze odměny a počáteční či předchozí odhad ohodnocení. U ohodnocovacích funkcí, které pracují s rozdílem odměny a aktuálního ohodnocení má výše odměn i další význam. Takovéto ohodnocovací funkce totiž generují ohodnocení, která se pohybují pouze na stupnici ohraničené nejnižší a nejvyšší odměnou. V tomto případě tedy odměny určují i ohodnocovací stupnici.

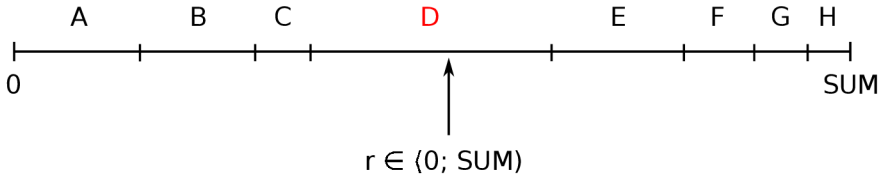
### 5.4.3 Rozhodovací funkce

Rozhodovací funkce posilovaného učení, neboli strategie, je funkce, kde se algoritmy typicky snaží vyvážit prozkoumávání stavového prostoru a využívání nejlépe ohodnocených stavů. Tyto algoritmy tedy provádějí dva druhy akcí. Jedná se o *greedy* a *non-greedy* akce. *Greedy* akce jsou takové akce, které volí jako následující stav ten, který má nejvyšší ohodnocení. Jsou to akce, které se snaží na základě dosud získaných dat co nejvíce zvýšit dlouhodobé množství získaných odměn. Naproti tomu *non-greedy* akce jsou akce náhodné. Jedná se o náhodný výběr následujícího stavu. Úkol těchto akcí je prozkoumávání stavového prostoru pro případ, že budou objeveny lépe ohodnocené stavy, než jsou stávající nejlépe ohodnocené stavy [20, s. 26].

Pro tento problém by teoreticky mohl být použit rozhodovací algoritmus posilovaného učení s názvem $\epsilon - greedy$. Tento algoritmus používá konstantu $\epsilon$, která procentuálně určuje množství *non-greedy* akcí (např. $\epsilon = 0.01$ odpovídá 1 %). Zbytek prováděných akcí je vždy *greedy*. To má za následek, že by se při nízkém parametru $\epsilon$ mohl algoritmus „zaseknout“ a provádět výběr stále stejné skladby [20, s. 28].

Druhou variantou pro výběr následující skladby pro přehrání by mohl být algoritmus známý jako ruleta (*roulette wheel selection*). Tento algoritmus dokáže náhodně vybrat prvek z dané množiny s ohledem na ohodnocení jednotlivých prvků. Výběr probíhá tak, že každý prvek množiny zabírá určité místo v prostoru hodnot. Konkrétně se toto místo rovná ohodnocení daného prvku. Všechny prvky mají své ohodnocení a tedy i svůj pomyslný prostor, který jim náleží. Celkový prostor hodnot má velikost rovnu sumě všech ohodnocení. Pro výběr prvku z množiny tedy stačí vygenerovat číslo z intervalu $\langle 0; SUM)$ a zjistit, kterému prvku toto číslo náleží [13].

Na obrázku 5.1 lze vidět zmíněný jednodimenzionální prostor ohodnocení zobrazený jako číselná osa, úseky připadající jednotlivým skladbám, které jsou označeny písmeny, a také samotný náhodný výběr a určení skladby, které vygenerované číslo náleží. Z této ilustrace je zřejmé, že čím větší prostor daná skladba vlastní, tím vyšší je pravděpodobnost jejího výběru. Jelikož prostor dané skladby přímo odpovídá jejímu ohodnocení, které zároveň určuje její popularitu u uživatele, je jasné, že hůře ohodnocené skladby mají nižší pravděpodobnost výběru, takže tato metoda přímo poskytuje požadované chování. I když




Obrázek 5.1: Ilustrace selekce pomocí rulety.

mají špatně ohodnocené skladby nízkou pravděpodobnost výběru, je přesto tato pravděpodobnost nenulová, takže může být i skladba s nízkým ohodnocením přehrána a získat tak ohodnocení vyšší.

# Kapitola 6

# Návrh řešení

## 6.1 Mobilní aplikace

Aplikace přehrávače musí obsahovat všechny základní funkce hudebního přehrávače pod operačním systémem Android. Mezi takové funkce například patří:

- zobrazovat název přehrávané skladby
- zobrazovat aktuální čas přehrávání
- zobrazovat celkový čas přehrávané skladby
- umožnit posun ve skladbě grafickým posuvníkem
- obsahovat funkční ovládací prvky pro přehrávání:
  - zapnout přehrávání
  - pozastavit přehrávání
  - přechod na předchozí skladbu
  - přechod na následující skladbu
  - ovládání *shuffle*
- zobrazovat na vyžádání hudební knihovnu s možností výběru skladby
- upozorňovat na svou činnost (přehrávání) notifikací

K těmto funkcím je nutné v této konkrétní aplikaci ještě přidat i ovládání predikce. A sice její zapínání a vypínání. Další funkce, které již nejsou na první pohled viditelné jsou například automatizované vyhledání audio souborů v paměti telefonu či plná podpora *Audio Focus*. Aplikace by pro přehrávání skladeb měla využívat službu, jelikož kvůli životnímu cyklu aktivit není vhodné objekt pro přehrávání vytvářet přímo v aktivitě přehrávače.

Protože bude tato aplikace pracovat s predikcí, která vyžaduje ukládání jednotlivých ohodnocení skladeb, je potřeba vyřešit kam a jak tyto hodnoty ukládat. Uložení těchto hodnot musí být perzistentní, jinak by uživatel nesměl aplikaci vypínat, což je téměř neuskutečnitelné. Nabízí se dvě varianty jak ohodnocení ukládat.

První možností je ukládat tyto hodnoty blíže specifikovaným formátem do souboru, který by byl uložen v externí paměti zařízení. Toto řešení není úplně špatné, ale data aplikace by tak teoreticky mohla být přístupná komukoliv, kdo tento soubor najde. Další problém

je ten, že by pro práci s tímto souborem bylo třeba vytvořit rozhraní, které by ukládání, načítání a úpravy ohodnocení provádělo. Implementace takového rozhraní by nemusela být natolik složitá, ale operace jako úprava ohodnocení by mohla být velmi časově náročná.

Druhou variantou je použití *SQLite* databáze, která je součástí OS Android a je přístupná všem aplikacím. U tohoto řešení stačí použít již implementované rozhraní ke zmíněné databázi a skrze něj provádět příkazy, které používají SQL syntaxi. Hlavní výhodou je především implementované a odladěné rozhraní skrze které se komunikuje. Další výhodou je kontrolovaný přístup k databázi, který neumožňuje neautorizovaný přístup. Také je výhoda možnost uzavřít operace do transakce, čímž je zajištěna atomičnost prováděných úprav.

Tato varianta tedy bude při implementaci použita, přičemž pro aplikaci bude při její instalaci automaticky vytvořena tabulka v databázi, která aplikaci náleží. Zmíněná databázová tabulka s názvem `Prediction` bude mít strukturu odpovídající popisu v tabulce 6.1.

| **Název sloupce** | **Datový typ** |
|---|---|
| `_id` | INTEGER |
| `value` | FLOAT |
| `last_sync` | TIMESTAMP |
| `genre_id` | INTEGER |

Tabulka 6.1: Struktura databázové tabulky `Prediction`.

Sloupec `_id` bude sloužit pouze jako primární klíč tabulky, takže bude jednoznačně identifikovat každou ze skladeb uložených v tabulce. Sloupec `value` bude obsahovat uložené ohodnocení skladby. Ohodnocení budou ukládána jako čísla s plovoucí řádovou čárkou kvůli dosažení vyšší přesnosti. Sloupec `last_sync` bude sloužit k uložení časového razítka poslední synchronizace záznamu. Tato synchronizace bude probíhat vždy po zapnutí přehrávače a bude mít za úkol synchronizovat tabulku `Prediction` s aktuálním stavem skladeb v zařízení.

Poslední sloupec `genre_id` bude mít za úkol ukládat identifikační číslo žánru skladby. To bude možno použít při selekci, která bude brát v potaz žánr skladeb. Tato informace totiž není dostupná přímo v tabulce skladeb, kterou spravuje OS Android, takže musí být evidována zde. Důvod, proč není informace o žánru přímo dostupná u skladby v databázi systému Android je nejspíše ten, že skladba může spadat do více žánrů současně, nelze tedy identifikační čísla těchto žánrů ukládat do jednoho sloupce. Jelikož je funkcionalita zohlednění žánru při selekci spíše doplňková, bude zanedbán fakt, že skladba může patřit do více žánrů současně a bude tedy ukládán pouze jeden. Žánr však velmi často vyplněn není, pro takové případy bude nastavena výchozí hodnota -1.

## 6.2 Algoritmus predikce

Algoritmus, který zajišťuje funkci predikce se skládá ze dvou částí. První část provádí výpočet ohodnocení skladby dle určitých kritérií. Druhá část provádí výběr skladby pro přehrání s respektováním ohodnocení všech skladeb. Tyto části jsou na sobě nezávislé. V následujících dvou oddílech bude proveden výběr algoritmů pro obě části. Výběr bude probíhat z možností popsaných v sekci 5.4.

### 6.2.1 Ohodnocování

Ohodnocovací funkce vhodná pro tento problém, by měla provádět ohodnocování dle odměn. Měla by fungovat i v případě, že se jedná o rozsáhlý nestacionární problém. Také je zapotřebí, aby dokázala pracovat v tzv. kontinuálním problému. Toto jsou specifika problému, který je řešen, a ohodnocovací funkce se s nimi musí vyrovnat. Z pohledu kompletních metod posilovaného učení je navíc zapotřebí, aby docházelo k ohodnocování stavů, nikoliv párů stav-akce. Dále musí platit, že stavy jsou na sobě nezávislé, takže navzájem neovlivňují své ohodnocování. Metoda také nesmí vyžadovat model prostředí, jelikož ten u řešeného problému vytvářen nebude.

Třídu metod *Dynamic Programming* lze vyřadit především kvůli tomu, že vyžaduje model prostředí. Navíc díky své vyšší výpočetní náročnosti není vhodná pro rozsáhlé problémy. Také není příliš vhodná pro nestacionární problémy. Žádná metoda z této třídy tedy nemůže být použita pro řešení tohoto problému.

Třída metod *Monte Carlo* sice nevyžaduje znalost prostředí, vyžaduje však rozdělení úlohy na epizody. Není tudíž vhodná pro kontinuální problémy, takže nelze ani z této třídy metod žádnou metodu použít.

Velké množství metod posilovaného učení uvažuje pouze stavy, které na sobě navzájem závisí. To v tomto případě ale neplatí. Nelze tedy použít tyto kompletní metody posilovaného učení:

- metoda *TD(0)*
- metoda *Q-Learning*
- metoda *SARSA*
- metoda *Actor-Critic*

Pro všechny tyto metody totiž platí, že při ohodnocování konkrétního stavu či páru stav-akce započítávají do ohodnocení i informace z jiných stavů nebo párů stav-akce.

Další problém je ten, že většina metod se snaží o ohodnocování párů stav-akce. Jinými slovy se snaží správně odhadnout hodnotu $Q(s, a)$, kde $a$ je akce ve stavu $s$. U tohoto konkrétního problému by však stačilo ohodnocovat stavy, protože závislost akcí na konkrétním stavu systému je v tomto případě irelevantní. Mohly by se však používat akce globálním způsobem, kdy jakákoliv akce, která provádí přechod na určitý stav $s$, by měla ohodnocení stejné jako ohodnocení stavu $s$. Bez ohledu na to, z jakého stavu daná akce vychází. V takovém případě je však jednodušší přímo použít ohodnocovaní stavů. Tento způsob řešení by se tedy týkal pouze metod, které jsou definovány pouze pro ohodnocování párů stav-akce.

I z tohoto důvodu nejsou metody *Q-Learning* a *SARSA* příliš vhodné. Obě provádí ohodnocování párů stav-akce a muselo by tedy docházet k určité úpravě jejich chování.

Metoda *R-Learning for Undiscounted Continuing Tasks* by mohla být zajímavým řešením, přestože se jedná o experimentální metodu. U této metody by byl problém jen v tom ohledu, že by její výpočty ohodnocení pro velké množství stavů mohli být značně výpočetně náročné. Proto její implementace do aplikace, kterou budou používat hlavně mobilní zařízení, není příliš vhodná. Navíc metoda používá relativní ohodnocovaní, čímž by mohla snížit rozdíly mezi ohodnoceními, což by bylo nežádoucí. Snížení rozdílů mezi ohodnoceními by mělo za výsledek delší dobu učení a nižší míru ovlivnění výběru skladeb.

Metody používající tzv. *eligibility traces* nejsou příliš vhodné právě z důvodu použití *eligibility traces*. Hodnota *eligibility trace* se totiž v čase mění a zvýhodňuje častěji navštěvované stavy. Toto chování však není příliš vhodné, protože by se musely ohodnocení pro

každou časovou jednotku přepočítávat. Navíc ve svém původním znění by tyto ohodnocení vedly k opakování právě navštívených skladeb. Pokud by bylo chování těchto hodnot opačné, kdy se jejich hodnota zvedá v případě, že není stav navštíven, mohlo by toto chování zajistit nižší pravděpodobnost opakování stále stejných skladeb. Stále by však setrval problém s nutností přepočítávat všechna ohodnocení pro každou časovou jednotku.

Pro implementaci byly vybrány všechny varianty řešení pomocí konkrétních vzorců pro výpočet ohodnocení. Jejich hlavní výhodou je jednoduchost použití a možnost bezproblémové kombinace s libovolnou rozhodovací funkcí. U těchto vzorců platí, že stačí jejich předpis přepsat do programu a tím je získán celý algoritmus potřebný pro výpočet aktuálního ohodnocení stavu. To stejné nutně nemusí platit v případě kompletních metod posilovaného učení. Typicky se u těchto potenciálně použitelných metod vyskytuje problém s náročností výpočtů a muselo by tedy dojít k zásadní optimalizaci výpočtů. V každém případě však bude muset dojít k optimalizaci hodnot parametrů, které se v první variantě řešení vyskytují, takže se určitá optimalizace provést musí.

### 6.2.2 Selekce

Selektivní (rozhodovací) funkce pro tento problém vyžaduje nejlépe algoritmus náhodného výběru, který bude respektovat ohodnocení všech skladeb. Přestože je nutné skladby s vyšším ohodnocením zvýhodnit, nelze vybírat stále stejnou skladbu s nejvyšším ohodnocením, jak by tomu částečně mohlo docházet v případě $\epsilon - greedy$. Rozhodovací funkce tedy musí splňovat i to, že má každá skladba pravděpodobnost výběru větší jak nula. Tato funkce by tedy měla být schopna vybírat i jiné než *greedy* akce, a to s pravděpodobností, která je přímo úměrná ohodnocení daného stavu kam akce provádí přechod.

Pro implementaci byl vybrán algoritmus ruleta (viz druhá popisovaná varianta v sekci 5.4.3). Tento algoritmus je totiž oproti první variantě v sekci 5.4.3 ($\epsilon - greedy$) mnohem vhodnější, jelikož zachovává stochastičnost výběru skladby na vysoké úrovni, ale zároveň respektuje rozdílnou popularitu jednotlivých skladeb.

Algoritmus $\epsilon - greedy$ je totiž navržen spíše na deterministické výběry stavů, kdy výsledkem výběru je vždy prvek s nejvyšším ohodnocením. To platí pro akce *greedy*. *Non-greedy* akce jsou zase naopak až příliš stochastické, protože provádí náhodný výběr stavu bez ohledu na ohodnocení stavů [20, s. 28].

Algoritmus ruleta byl vybrán především z toho důvodu, že odpovídá výše popsaným požadavkům na selekci. Pro snížení jeho výpočetních či paměťových nároků lze použít *Alias* metodu či její varianty [21].

V obou případech (ruleta i alias) vyžaduje algoritmus kvalitní generátor pseudonáhodných čísel. Aplikace bude používat generátor pseudonáhodných čísel, který přímo poskytuje OS Android. U něj by totiž mělo být možné předpokládat funkčnost bez nutnosti testování.

# Kapitola 7

# Optimalizace a srovnání algoritmů predikce

Před srovnáním jednotlivých algoritmů predikce bylo nejprve nutné provést jejich optimalizaci. Ta spočívala především v odhadnutí nejvhodnějších hodnot parametrů ohodnocovací funkce (viz první varianta v sekci 5.4.1). Poté bylo provedeno srovnání jednotlivých variant algoritmů predikce. Ty se lišily především v použité ohodnocovací funkci. Cílem optimalizace a srovnání bylo nalezení nejlepšího z navrhovaných řešení (viz konkrétní vzorce pro výpočet ohodnocení v sekci 5.4.1).

## 7.1 Možnosti optimalizace

Způsob provedení optimalizace byl vybrán z těchto tří možností:

1. použití genetického algoritmu
2. sběr dat a následné *offline* zpracování optimalizovávaným algoritmem
3. simulace provádění činů a s tím spojené *online* zpracování optimalizovávaným algoritmem

*Offline* zpracování dat v tomto případě znamená, že optimalizovávaný algoritmus predikce nepoužívá funkci pro selekci, takže data, která se již zpracovala nemají vliv na data, která se ještě zpracovávat budou. Zpracovávaná data sice ovlivňují ohodnocení pomyslných skladeb, toto ohodnocení však neovlivňuje (při *offline* zpracování) budoucí chování algoritmu predikce.

Naproti tomu *online* zpracování dat znamená, že zpracovaná data, tím že ovlivnila ohodnocení pomyslných skladeb, pozměnila i budoucí chování algoritmu predikce, jelikož se tyto ohodnocení používají i pro selekci. Jinými slovy se algoritmus predikce chová přesně tak, jakoby byl implementován ve finální aplikaci a používán uživatelem aplikace.

První ze způsobů provedení optimalizace byl kvůli nedostatku času vyloučen. Byl by to ale velmi zajímavý způsob ladění parametrů první varianty ohodnocovací funkce. Pro tento úkol by bylo použití genetického algoritmu nejspíše vhodné a efektivní.

Zbývající dvě možnosti optimalizace jsou podrobně popsány v následujících sekcích 7.1.1 a 7.1.2.

### 7.1.1 Sběr dat a offline zpracování

Základem tohoto způsobu optimalizace bylo vytvoření aplikace přehrávače pro sběr dat od uživatelů. Tyto data představovaly jednotlivé činy uživatelů, které uživatelé provedli při běžném používání přehrávače.

Po získání dostatečného množství dat bylo možné zahájit jejich *offline* zpracování. Zpracování prováděl program `PredictionOfflineTester` napsaný v jazyce Java. Tento program postupně po řádcích načítal jednotlivé činy uživatele ze souboru a zpracovával je dle ohodnocovací funkce. Po zpracování celého souboru byly vypsány informace o tom, jaké ohodnocení jednotlivé skladby získaly. Tyto hodnoty poté bylo možné zpracovávat dále.

Tento způsob optimalizace ale nakonec také nebyl použit, přestože byla aplikace pro sběr dat i program provádějící zpracování dat implementován. Důvod proč nebyl tento způsob optimalizace aplikován je především ten, že způsob, kterým jsou data zpracovávána neodpovídá způsobu zpracování dat prediktivním algoritmem při reálném použití. Výhodou sice může být použití reálných dat z běžného používání přehrávače, nevýhoda *offline* zpracování však převažuje. Další problém tohoto způsobu optimalizace je i nemožnost úprav selektivního algoritmu, protože se při zpracování dat nepoužívá.

### 7.1.2 Simulace provádění činů a online zpracování

Poslední ze způsobů optimalizace, ze kterých probíhal výběr, je simulování provádění činů za uživatele a jejich *online* zpracování optimalizovávaným algoritmem predikce. Právě tento způsob optimalizace byl nakonec použit. Pro účely optimalizace tímto způsobem byl implementován program `PredictionOnlineTester` v jazyce Java. V tomto programu je přítomna implementace kompletního optimalizovávaného algoritmu predikce. Je v něm tedy implementována jak ohodnocovací funkce, tak i funkce zajišťující selekci. Dále je v programu implementována funkce pro inicializaci pole s ohodnocením jednotlivých skladeb a ve funkci `main` se již nachází cyklus, ve kterém se provádí jednotlivé činy. Ty provádí sám program, dle nastavených parametrů simulace, místo uživatele. Parametry, které se v programu vyskytují se dělí na parametry simulace a parametry konkrétního implementovaného algoritmu. Mezi parametry simulace patří:

- počet průběhů simulace na jedno spuštění programu
- počet činů, které se mají provést (po dosažení této hodnoty je simulace ukončena)
- počet neoblíbených skladeb
- celkový počet skladeb
- pravděpodobnost činu „výběr z knihovny“
- pravděpodobnost opačného činu

Parametr *pravděpodobnost činu „výběr z knihovny“* určuje s jakou pravděpodobností bude následující čin výběr z knihovny. Při výběru z knihovny se provádí selekce pouze z oblíbených skladeb. *Pravděpodobnost opačného činu* určuje pravděpodobnost přeskočení oblíbené skladby, nebo naopak přehrání skladby neoblíbené.

Oblíbená skladba je skladba, kterou uživatel nepřeskakuje, nebo jen velmi ojediněle. V tomto programu se zmíněná skladba nepřeskakuje a má nenulovou pravděpodobnost při

výběru skladby z knihovny. Neoblíbená skladba je naopak taková skladba, kterou uživatel téměř vždy přeskočí a v tomto programu se tedy skladba přeskakuje a má nulovou pravděpodobnost při výběru z knihovny.

Simulace jednoho činu začíná tím, že je zavolána funkce algoritmu predikce pro selekci. Tato funkce vrátí index skladby, která byla vybrána s ohledem na ohodnocení všech skladeb. Poté se určí zda tato skladba patří mezi oblíbené, či neoblíbené. Pokud patří mezi oblíbené skladby nastaví se, že odměna kterou má ohodnocovací funkce zpracovat je rovna odměně za přehrání skladby. V případě neoblíbené skladby je odměna nastavena na odměnu za přeskočení skladby. Index této skladby a odměna jí přidělená je následně předána ohodnocovací funkci, která ohodnocení skladby dle své implementace přepočítá a uloží.

V tomto popisu nebylo pro jednoduchost zmíněno použití parametru *pravděpodobnost činu „výběr z knihovny“* ani parametru *pravděpodobnost opačného činu*. Také by byl průběh mírně odlišný pokud by daný čin byl první, nebo naopak poslední. V takovém případě by došlo buď k inicializaci pole skladeb v případě prvního činu, nebo zápisu výsledků do souboru v případě činu posledního.

Způsob optimalizace tímto programem probíhá následujícím způsobem:

1. Implementace optimalizovávaného algoritmu predikce.
2. Nastavení parametrů simulace i algoritmu predikce.
3. Spuštění programu.
4. Zpracování výsledků.

Výsledky jsou ve formátu `.csv`, takže je lze otevřít a dále zpracovávat v jakémkoliv tabulkovém procesoru, který tento formát podporuje. Při optimalizaci byly tyto výsledky typicky průměrovány. Většinou bylo totiž nastaveno větší množství průběhů simulace než-li jeden.

## 7.2 Metriky kvality ohodnocení skladeb

Při optimalizaci byly použity dvě metriky, a sice násobnost a stabilita. Tyto metriky slouží k určení kvality ohodnocení skladeb ohodnocovací funkcí.

Násobnost určuje kolikanásobně vyšší pravděpodobnost výběru má skladba jejíž ohodnocení je rovno průměru všech ohodnocení oblíbených skladeb, vůči skladbě jejíž ohodnocení je rovno průměru všech ohodnocení neoblíbených skladeb. Násobnost pak odpovídá tomuto vzorci:

$$nasobnost = \frac{\frac{\left(\frac{1}{k}\sum_{i=1}^{k} x_i\right)\cdot 100}{n}}{\frac{\left(\frac{1}{l}\sum_{j=1}^{l} x_j\right)\cdot 100}{n}} = \frac{\frac{1}{k}\sum_{i=1}^{k} x_i}{\frac{1}{l}\sum_{j=1}^{l} x_j}$$

V tomto vzorci je proměnná $k$ počet oblíbených skladeb, proměnná $l$ počet neoblíbených skladeb, proměnná $x_i$ ($x_j$) ohodnocení skladby $i$ ($j$) a proměnná $n$ je celkový počet skladeb.

Druhá zmiňovaná metrika, stabilita, určuje velikost rozdílu dvou skladeb $s_1 - s_2$, pro které platí že:

1. obě skladby jsou skladby oblíbené
2. skladba $s_1$ má nejvyšší ohodnocení

3. skladba $s_2$ má nejnižší ohodnocení

Pokud je rozdíl příliš velký, znamená to, že skladba s nejvyšším ohodnocením získala své vysoké ohodnocení díky častějšímu přehrávání na úkor ostatních oblíbených skladeb. A toto vysoké ohodnocení získala typicky již na začátku učení. Díky tomu ostatní skladby zůstaly i například u svého původního ohodnocení, protože je dostatečně nízké oproti tomu nejvyššímu. Tato metrika však byla upravena tak, aby více odpovídala svému názvu.

Název metriky napovídá, že čím vyšší bude hodnota stability, tím bude nastavení stabilnější, ale v původní formě této metriky je tomu přesně naopak, kdy vyšší rozdíl mezi skladbami značí méně stabilní nastavení. Proto se tento rozdíl bude navíc odečítat od hodnoty 100, která odpovídá maximálnímu ohodnocení, a bude tedy hodnota stability rovna:

$$100 - (s_1 - s_2)$$

kde právě výsledná hodnota 100 bude označovat stoprocentní stabilitu, jelikož rozdíl mezi ohodnocením oblíbených skladeb bude nulový.

## 7.3 Optimalizace

Jak již bylo řečeno, pro optimalizaci byl napsán program `PredictionOnlineTester`, který provádí činy jako uživatel a poskytuje tedy téměř automatizovanou optimalizaci algoritmů predikce.

### 7.3.1 Nalezení vhodných hodnot parametrů

Před tím, než bylo možné porovnat jednotlivé algoritmy predikce, bylo potřeba nalézt nejvhodnější hodnoty pro parametry ohodnocovací funkce, která byla popsána jako první varianta pro ohodnocování (viz 5.4.1). Proces hledání nejvhodnějších hodnot byl experimentální.

Parametry jejichž hodnoty je nutné nalézt:

- alfa (step-size) – určuje velikost změny ohodnocení
- počáteční ohodnocení – určuje výchozí ohodnocení všech nových skladeb
- odměny – určují správnost/špatnost provedené selekce
  - odměna za přeskočení skladby – negativní odměna, vždy snižuje ohodnocení, určuje minimální dosažitelné ohodnocení
  - odměna za dokončení skladby – pozitivní odměna, většinou zvyšuje ohodnocení
  - odměna za výběr z knihovny – pozitivní odměna, vždy zvyšuje ohodnocení, určuje maximální dosažitelné ohodnocení

Odměny jsou zvoleny takto:

- odměna za přeskočení skladby = 1
- odměna za dokončení skladby = 80
- odměna za výběr z knihovny = 100

Tyto hodnoty byly vybrány tak, aby vytvářely stupnici, která se pro ohodnocování skladeb běžně používá [16]. Je jimi tedy zvolena stupnice, na které se budou všechna ohodnocení pohybovat. Stupnice je stobodová od nejnižšího ohodnocení 1 až po nejvyšší 100. Tento fakt zároveň určuje, že skladba nejlépe ohodnocená (100) má stonásobně vyšší pravděpodobnost výběru, než-li skladba s ohodnocením 1, tedy nejhůře ohodnocená. Pro parametry alfa a počáteční ohodnocení již byly nejvhodnější hodnoty experimentálně zjišťovány.

Experimenty byly provedeny ve třech fázích:

1. Nalezení vhodného počátečního ohodnocení.
2. Nalezení vhodné hodnoty parametru alfa.
3. Opětovné vyhodnocení počátečního ohodnocení, po změně hodnoty alfa.

### První fáze

V první fázi byly parametry simulace nastaveny dle tabulky 7.1.

| | |
|---|---|
| Alfa | 0,1 |
| Počáteční ohodnocení | $\langle 20; 80 \rangle$ |
| Odměna za přeskočení | 1 |
| Odměna za dokončení | 80 |
| Odměna za výběr z knihovny | 100 |
| Počet činů | 500 |
| Neoblíbené skladby | 8 |
| Oblíbené skladby | 42 |
| Pravděpodobnost výběru z knihovny | 10% |
| Pravděpodobnost opačného činu | 0% |

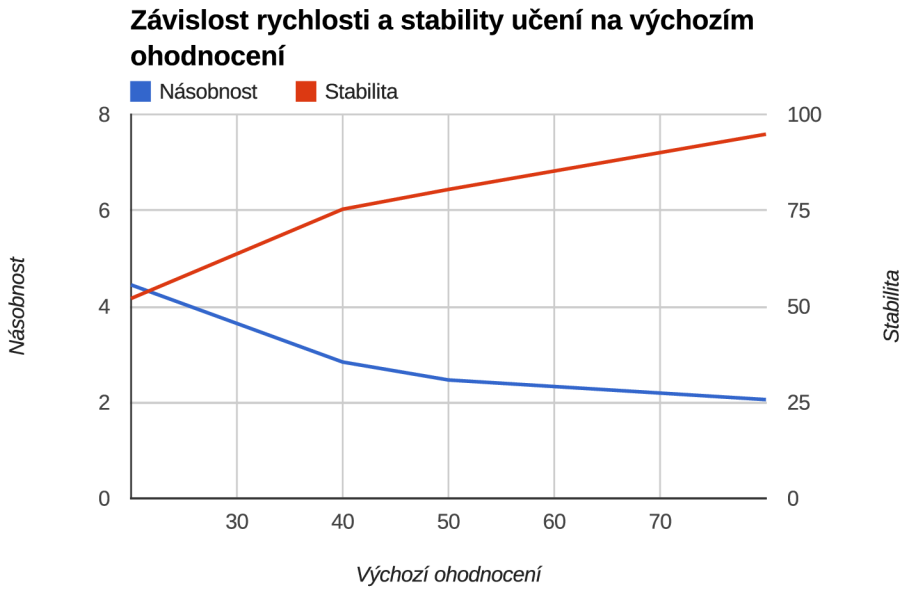
Tabulka 7.1: Parametry simulace pro první fázi optimalizace.

Při tomto nastavení byly parametry alfa, oblíbené i neoblíbené skladby a pravděpodobnost výběru z knihovny zvoleny odhadem. Počet činů odpovídá desetinásobku počtu všech skladeb. Parametr „pravděpodobnost opačného činu“ je roven nule z toho důvodu, že v době provádění první fáze optimalizace ještě tato stochastičnost nebyla implementována.

Protože se v této fázi vyhodnocuje počáteční ohodnocení, není nastaveno pevně, ale je proměnné. To znamená, že v každém běhu programu pro optimalizaci bylo počáteční ohodnocení nastaveno všem skladbám na hodnotu, která byla v daném běhu zkoušena. Výsledné hodnoty obou metrik jsou tedy vztaženy ke zkoušeným hodnotám počátečního ohodnocení.

Na grafu 7.1 lze pozorovat, že zvolené metriky jsou protichůdné a bude tedy nutné zvolit určitý kompromis. Při tomto chování byla zvolena hodnota počátečního ohodnocení 80 jako nejvhodnější, protože i 75% stabilita je velmi nízká. Je nutné si uvědomit, že 75% stabilita znamená rozdíl ohodnocení mezi stejně oblíbenými skladbami o velikosti jedné čtvrtiny ohodnocovací stupnice. Výhodou tohoto ohodnocení také bylo vysoké ohodnocení pro nově přidané skladby, které by jim zajistilo dostatečnou pravděpodobnost pro přehrání.

Počáteční ohodnocení nastavené na hodnotu 80 mělo za následek pomalé učení implementovaného algoritmu. I za předpokladu, že uživatel provede oněch 500 činů, kdy teoreticky připadá na každou skladbu 10 z nich, bude pravděpodobnost selekce oblíbené skladby




Obrázek 7.1: Graf závislosti rychlosti a stability učení na počátečním ohodnocením.

jen dvakrát vyšší, než-li výběr skladby neoblíbené, což by bylo v praxi málo pozorovatelné. Bylo tedy nutné algoritmus vylepšit.

Vylepšení má za úkol zvýšit rychlost počátečního učení popularity skladeb u uživatele. Funguje tak, že pro první ohodnocování skladby algoritmem, z jejího počátečního ohodnocení, je zvýšena hodnota parametru alfa, který určuje jak moc bude ohodnocení upraveno vůči získané odměně a předchozímu ohodnocení. Toto vylepšení sice zvyšuje rychlost učení, avšak je zde předpoklad, že alespoň při prvním přehrání skladby provede uživatel správný čin, kterým dá jasně najevo, zda má, či nemá skladbu rád. Tento předpoklad stojí na tom, že o tomto vylepšení bude uživatel informován, navíc nemůže dojít při prvním přehrání k tomu, že by uživatel skladbu přeskočil z důvodu, že ji přehrávač již mnohokrát přehrál.

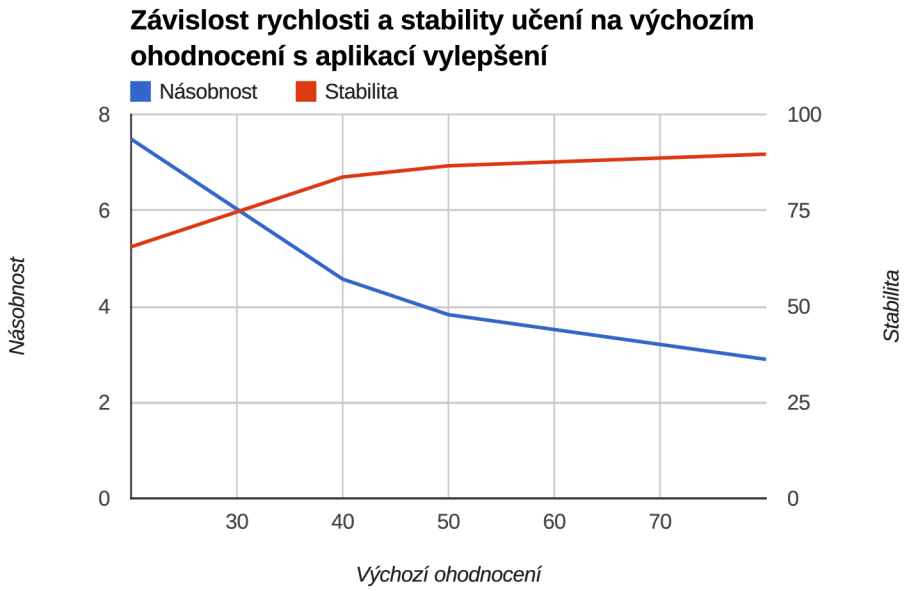
Popsané vylepšení bylo implementováno a znovu proběhlo hledání ideálního počátečního ohodnocení, jelikož se změnil samotný algoritmus ohodnocování. Výsledky po implementaci vylepšení lze pozorovat na grafu 7.2.

V grafu lze vidět zlepšení v oblasti rychlosti učení, jelikož pro stejné parametry, dané fází optimalizace, vzrostla maximální násobnost k hodnotám blížícím se $7,5$ z původní hodnoty $4,4$. Aplikované vylepšení ale způsobilo také snížení stability na maximální hodnotu $90\%$ z původních $95\%$. V oblasti počátečního ohodnocení 50 však došlo k téměř dvojnásobnému zvýšení násobnosti a $7\%$ nárůstu stability, a toto ohodnocení tedy bylo zvoleno jako nejvhodnější. Bylo tedy použito v další fázi, ve které byl vyhodnocován parametr alfa. Použité vylepšení již bylo přítomné u všech dalších fází.

### Druhá fáze

Druhá fáze měla parametry mírně upravené, jsou uvedené v tabulce 7.2.

Pravděpodobnost opačného činu je již v této fázi nenulová. Také byl změněn parametr, který určuje počet činů. Jeho hodnota zde byla upravena na 200, kdy pro 50 skladeb připadá na každou skladbu teoreticky 4 činy uživatele. Doba učení tím tedy byla zkrácena na méně




Obrázek 7.2: Graf závislosti rychlosti a stability učení na počátečním ohodnocením po aplikování prvního vylepšení.

| | |
|---|---|
| Alfa | $\langle 0,01; 0,1\rangle$ |
| Počáteční ohodnocení | 50 |
| Odměna za přeskočení | 1 |
| Odměna za dokončení | 80 |
| Odměna za výběr z knihovny | 100 |
| Počet činů | 200 |
| Neoblíbené skladby | 8 |
| Oblíbené skladby | 42 |
| Pravděpodobnost výběru z knihovny | 10% |
| Pravděpodobnost opačného činu | 5% |

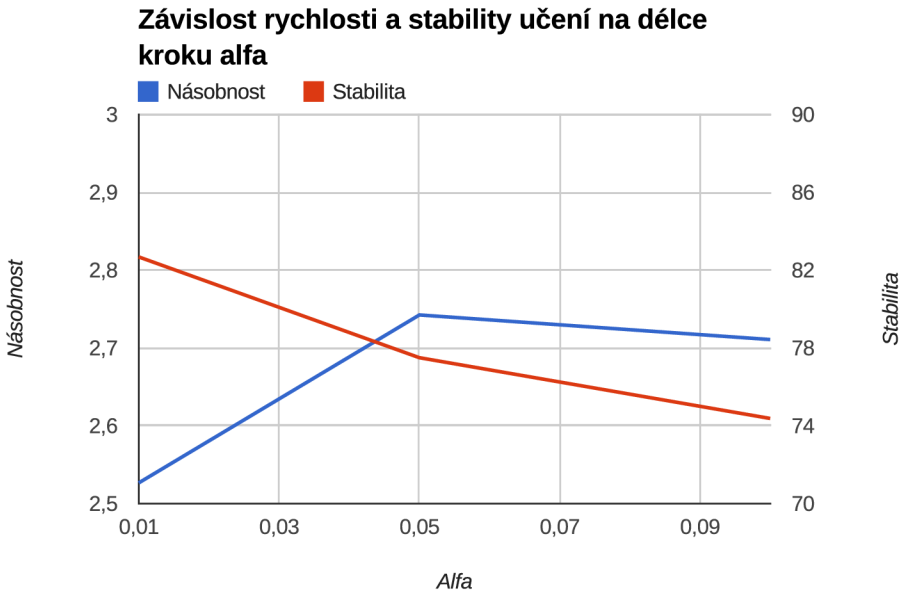
Tabulka 7.2: Parametry simulace pro druhou fázi optimalizace.

než polovinu oproti první fázi optimalizace.

Graf 7.3 zobrazuje závislost měřených metrik na parametru alfa. Lze vidět, že po snížení počtu činů klesla násobnost i stabilita algoritmu. Je však zřejmé, že parametr alfa má nejlepší výsledky při nastavení na hodnotu $0,05$. Opět se jedná o kompromis mezi rychlostí učení a stabilitou. Také lze pozorovat, že občasné opačné akce uživatele změnily rostoucí tendenci násobnosti při zvětšování parametru alfa na tendenci klesající.

Hodnoty v grafu však ze zmíněných důvodů nejsou příliš optimistické a proto bylo nutné opět vymyslet vylepšení, které vyřeší vážné problémy týkající se stability daného řešení, ale i rychlosti učení.

Toto vylepšení tedy zvyšuje především stabilitu učení. Pracuje tak, že algoritmus selekce vždy prvně vybírá ze skladeb, které mají stále své počáteční ohodnocení, pokud takové jsou. Tím je zaručeno, že jsou prvně přehrány všechny skladby právě jednou, než začne algoritmus




Obrázek 7.3: Graf závislosti rychlosti a stability učení na parametru alfa.

selekce pracovat prediktivně.

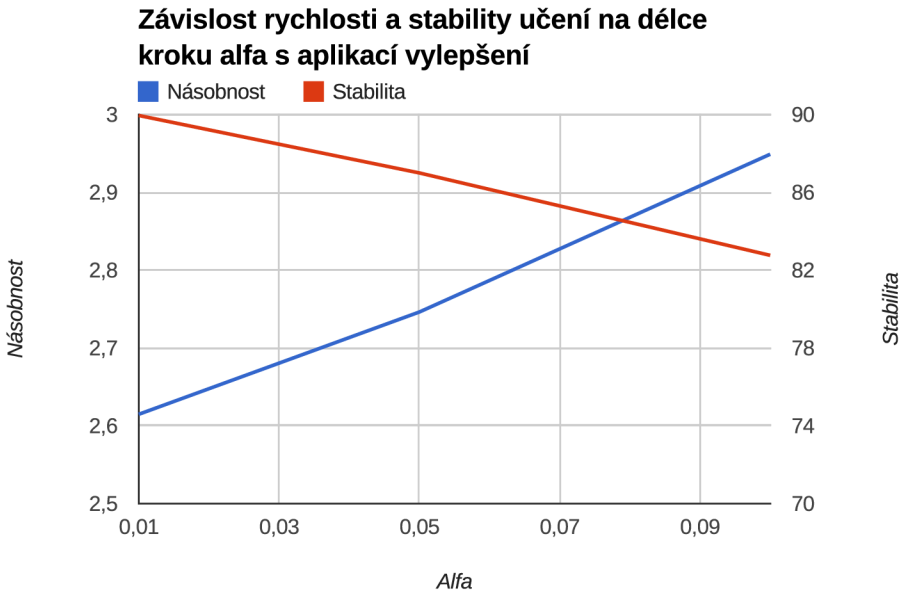
Výsledky z aplikace vylepšení lze pozorovat v grafu 7.4. Zlepšení nastalo v obou metrikách, ale především v oblasti stability. Výsledky jsou lepší, jelikož byl zaznamenán nárůst o téměř 10%. Také lze pozorovat, že parametr alfa více ovlivňuje stabilitu a počáteční ohodnocení zase násobnost. Proto je opět zvolena hodnota parametru alfa $0,05$, která poskytuje 90% v oblasti stability. Lze také vidět, že zvolením alfa $0,1$ by byla stabilita snížena asi o 7%, ale u násobnosti by se velkého zlepšení nedosáhlo.

### Třetí fáze

Poslední, třetí, fáze má shodné parametry s fází předchozí, ale tentokrát bude znovu vyhodnocován parametr počáteční ohodnocení. Parametry této fáze optimalizace jsou popsány v tabulce 7.3.

| | |
|---|---|
| Alfa | 0,05 |
| Počáteční ohodnocení | $\langle 20; 80 \rangle$ |
| Odměna za přeskočení | 1 |
| Odměna za dokončení | 80 |
| Odměna za výběr z knihovny | 100 |
| Počet činů | 200 |
| Neoblíbené skladby | 8 |
| Oblíbené skladby | 42 |
| Pravděpodobnost výběru z knihovny | 10% |
| Pravděpodobnost opačného činu | 5% |

Tabulka 7.3: Parametry simulace pro třetí fázi optimalizace.




Obrázek 7.4: Graf závislosti rychlosti a stability učení na parametru alfa po aplikování druhého vylepšení.

Tato fáze má za úkol zvolit vhodné počáteční ohodnocení, při zachování stability poskytované aplikací obou vylepšení a parametru alfa na hodnotě $0,05$. Právě vhodné počáteční ohodnocení může zvýšit rychlost učení implementovaného algoritmu. Po této fázi by tedy měli být známy ideální hodnoty obou nastavovaných parametrů. Také budou známy hodnoty stability a násobnosti pro toto výsledné řešení.

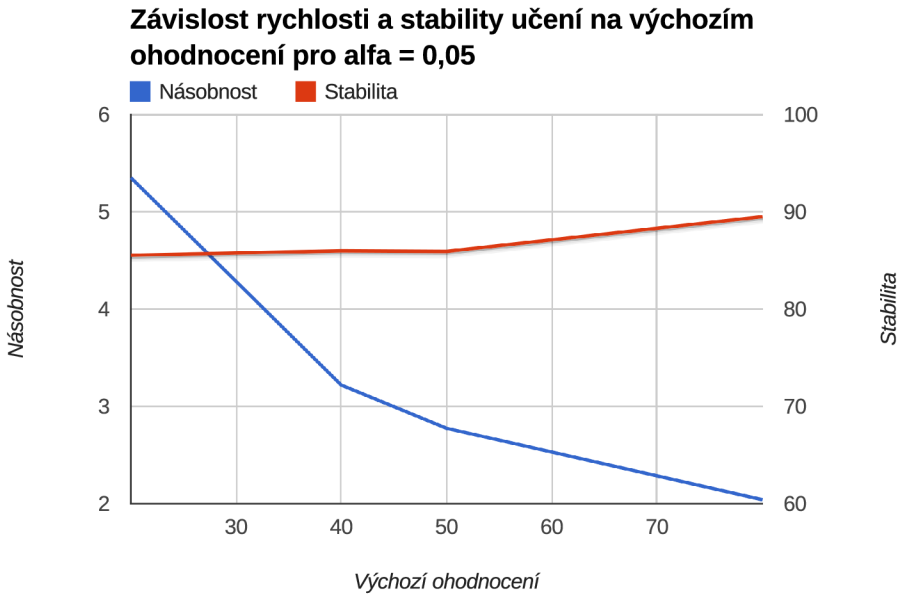
Graf 7.5 opět zobrazuje závislost obou metrik na parametru počátečního ohodnocení, tentokrát ale již s aplikovanými vylepšeními a parametrem alfa nastaveným na hodnotu $0,05$. Lze pozorovat, že rozdíly ve stabilitě jsou v jednotkách procent, takže bylo zvoleno počáteční ohodnocení 20, které má nejlepší poměr obou metrik ze všech dosavadních výsledků.

Při aplikaci obou vylepšení je tedy nejvhodnější počáteční ohodnocení rovno 20 a parametr alfa roven $0,05$. Takto zvolené hodnoty poskytují násobnost ve výši $5,4$ a stabilitu na úrovni $85,6\%$.

### 7.3.2 Srovnání jednotlivých algoritmů

Po nastavení a vylepšení algoritmu predikce s obecnou ohodnocovací funkcí pro stavy posilovaného učení bylo možné provést porovnání jednotlivých algoritmů predikce mezi sebou. Jedná se o algoritmy predikce lišící se v ohodnocovací funkci, přičemž jednotlivé varianty byly popsány v první části sekce 5.4.1.

V grafu 7.6 lze vidět srovnání všech tří variant ohodnocovacích funkcí. Z grafu je na první pohled zřejmé, že algoritmus s obecnou ohodnocovací funkcí pro stavy posilovaného učení (*Enhanced RL value function*) po odladění parametrů a po aplikaci všech vylepšení je nejlepší volbou. Dosahuje totiž nejvyššího výsledku v oblasti násobnosti i stability. Algoritmus používající Wilsonův intervalový odhad (*Wilson score interval*) má násobnost na




Obrázek 7.5: Graf závislosti rychlosti a stability učení na počátečním ohodnocení pro alfa $= 0,05$.

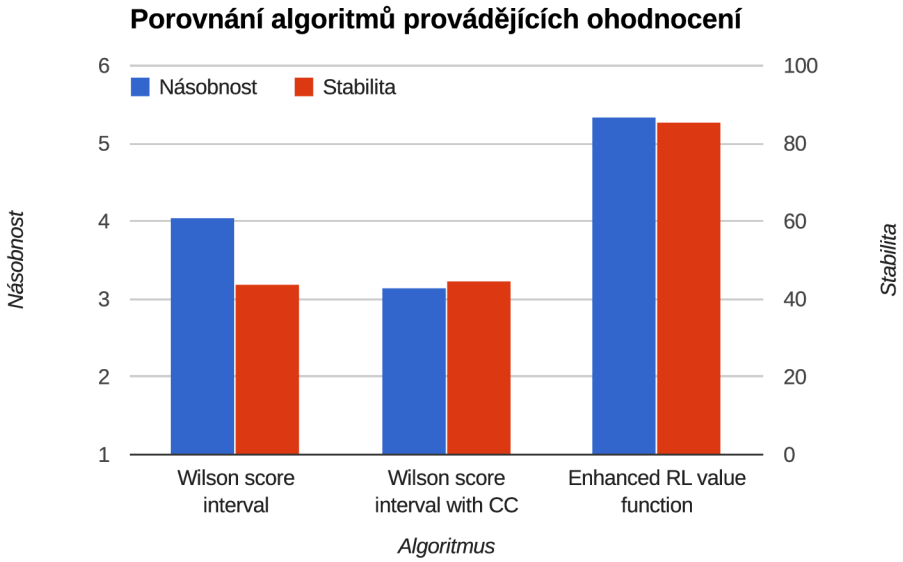
hodnotě, která by byla dostatečně vysoká na použití v aplikaci. U tohoto řešení však vzniká problém se stabilitou výsledných ohodnocení. Tato stabilita je natolik nízká, že nedosahuje ani padesátiprocentní hranice. Nelze tedy zmíněné řešení prakticky použít, jelikož by některé potenciálně oblíbené skladby nedostali možnost relevantně se ohodnotit, protože by jejich hodnocení bylo natolik nízké, že by měli velmi malou pravděpodobnost výběru v porovnání s ostatními, lépe ohodnocenými, oblíbenými skladbami. Algoritmus s ohodnocovací funkcí používající Wilsonův intervalový odhad s korekcí na spojitost (*Wilson score interval with CC*) má nižší násobnost a dalo by se tedy očekávat, že tím získá vyšší stabilitu výsledných hodnot. V grafu však lze pozorovat pouze nepatrný nárůst stability a tento algoritmus je tudíž taktéž nevhodný pro použití v aplikaci z důvodu nízké stability ohodnocení, které generuje.

## 7.4 Aplikovaná vylepšení

Algoritmus predikce byl rozšířen o jistá vylepšení, která by měla mít pozitivní vliv na kvalitu predikce, a tedy i na uživatelský prožitek. Původní algoritmus bez vylepšení byl sice funkční, ale tyto úpravy kladně přispěly k rychlosti a správnosti učení, nebo také k samotnému zvýšení úrovně predikce v reálném čase. Jedna z těchto úprav má vliv hlavně z krátkodobého pohledu, podle nálady uživatele, nikoliv dlouhodobé popularity skladeb.

U prvních dvou vylepšení, které jsou popsány výše (v sekci 7.3.1), je především snaha zdokonalit kvalitu ohodnocování prediktivního algoritmu, na rozdíl od druhých dvou, které mají za úkol vylepšit selektivní algoritmus.

Třetí z aplikovaných vylepšení má tedy za úkol zabránit tomu, aby algoritmus selekce vybral dvakrát po sobě stejnou skladbu. Takový výběr by totiž zapříčinil potřebu uživatele přeskočit danou skladbu, protože právě dohrála. Tento problém je vyřešen tím, že je daná




Obrázek 7.6: Graf porovnávající algoritmy pro ohodnocování.

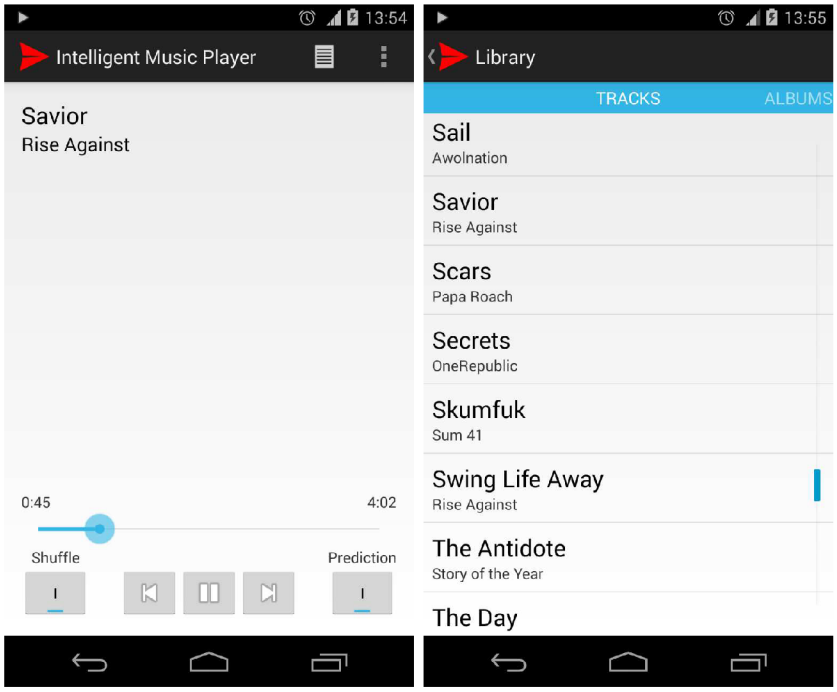
skladba po dobu selekce vyřazena z prostoru ohodnocení skladeb. Není tedy potřeba provádět selekci opakovaně, jelikož stejná skladba nemůže být vybrána, protože pro algoritmus selekce neexistuje.

Poslední vylepšení zvyšuje úroveň inteligence přehrávače, a to tím, že pokud uživatel přeskočí skladbu určitého žánru, je pro ostatní skladby tohoto žánru dočasně (po dobu následné selekce) sníženo ohodnocení. Tím, že uživatel přeskočí skladbu daného žánru může totiž dát najevo i to, že nemá náladu na daný žánr. Při další selekci, pokud nebyla opět způsobena přeskočením skladby určitého žánru, jsou ohodnocení nezměněná. Informace o žánru dané skladby je však ne vždy vyplněná. Pro tyto případy nedojde k tomu, že by bylo dočasně sníženo ohodnocení všech skladeb bez vyplněného žánru. Nedojde k žádnému snížení ohodnocení, protože nejspíše neexistuje žádné relevantní.

# Kapitola 8

# Implementace aplikace

U výsledné aplikace je nutné soustředit se na odezvu uživatelského prostředí při výpočetně náročných operacích. Taktéž je potřeba správně pracovat s aktivitami, kde je důležité respektovat jejich životní cyklus a ošetřit tak všechny vstupní body aplikace. Z těchto a dalších důvodů tato aplikace využívá pro vlastní přehrávání službu běžící na pozadí a výpočetně náročné operace provádí asynchronně pomocí vláken na pozadí. To zabraňuje situaci, kdy grafické rozhraní aplikace nereaguje na uživatelské akce. Na obrázku 8.1 lze vidět jak výsledná aplikace vypadá.

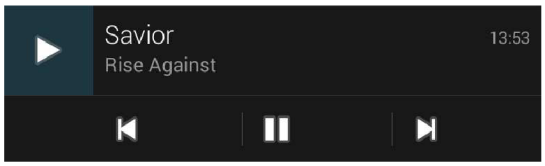



Obrázek 8.1: Snímky obrazovky finální aplikace. (Přehrávač vlevo, knihovna vpravo.)

Další výhodou použité architektury aplikace je i skutečnost, že je o přehrávání uživatel informován notifikací. V novějších verzích operačního systému tato notifikace obsahuje tlačítka, kterými lze přehrávač ovládat bez nutnosti zobrazování celé aplikace.

Tyto notifikace jsou zpětně kompatibilní i se systémy, které tlačítka v notifikacích nepod-

porují, takže nezpůsobí ani na starších verzích OS žádné problémy [5]. Notifikaci s tlačítky lze vidět na obrázku 8.2.




Obrázek 8.2: Notifikace s tlačítky.

## 8.1 Struktura aplikace

Aplikaci tvoří celkem 15 tříd, z nichž těchto 11 implementuje nejdůležitější funkce:

- `MediaPlayerService` – Jádro přehrávače (služba), které obsahuje objekt `MediaPlayer` přehrávající skladby. Dále je zodpovědné za zobrazení správných informací na hlavní obrazovce přehrávače, reaguje na broadcast oznámení OS a na změny *Audio Focus*. Z důvodu plné podpory obsahuje tato třída i výčet, který v sobě udržuje aktuální stav *Audio Focus*. Druhý výčet reprezentuje stav jádra. Udržovat tuto informaci je důležité především proto, že je nutné brát ohled na aktuální stav objektu `MediaPlayer` (viz 3.2.1).
- `AudioFocusManager` – Zajišťuje možnost reakce na změnu *Audio Focus* (viz 3.2.2).
- `MusicBroadcastReceiver` – Slouží pro reakce na broadcast oznámení pocházející z OS. Zpracovává pouze ta oznámení, která se týkají ovládání přehrávače nebo audio výstupu. Umožňuje tedy ovládat přehrávač pomocí tlačítek na tzv. *headset*, ale i pozastavit přehrávání pokud uživatel odpojí sluchátka od zařízení. Cílem je zvýšit komfort při používání implementovaného přehrávače.
- `MusicFilesRetriever` – Generování seznamu skladeb, umístěných v zařízení. Používá se vnitřní databáze OS Android. Seznam skladeb je pole objektů třídy `Track`. Objekt třídy `Track` obsahuje důležité informace o dané skladbě, jako je například název, album, celková délka či absolutní cesta k souboru dané skladby. Tato operace probíhá vždy při spuštění aplikace přehrávače a provádí se asynchronně.
- `FilesRetrieverAsyncTask` – Umožňuje generovat seznam skladeb asynchronně. Je vhodné provádět tuto činnost asynchronně, aby nemohlo dojít ke stavu, kdy uživatelské rozhraní nebude do doby dokončení této operace reagovat. Využívá pro provedení zmíněné operace vlákna v pozadí. O dokončení této přípravy je služba `MediaPlayerService` informována pomocí `TrackListCreatedListener`.
- `Prediction` – Implementovaná predikce prostřednictvím ohodnocovací funkce `value Function` a funkce realizující selekci `selectFunction`. Tato třída také obsahuje parametry upravující chování prediktivního algoritmu. Tyto parametry jsou *public*, lze k nim tedy přistupovat i z jiných tříd. Ohodnocovací funkce ve finální aplikaci odpovídá algoritmu *obecné ohodnocovací funkce pro stavy posilovaného učení* a funkce selektivní implementuje algoritmus *roulette wheel selection*. Tato třída používá pro ukládání ohodnocení jednotlivých skladeb databázi (viz 6.1).

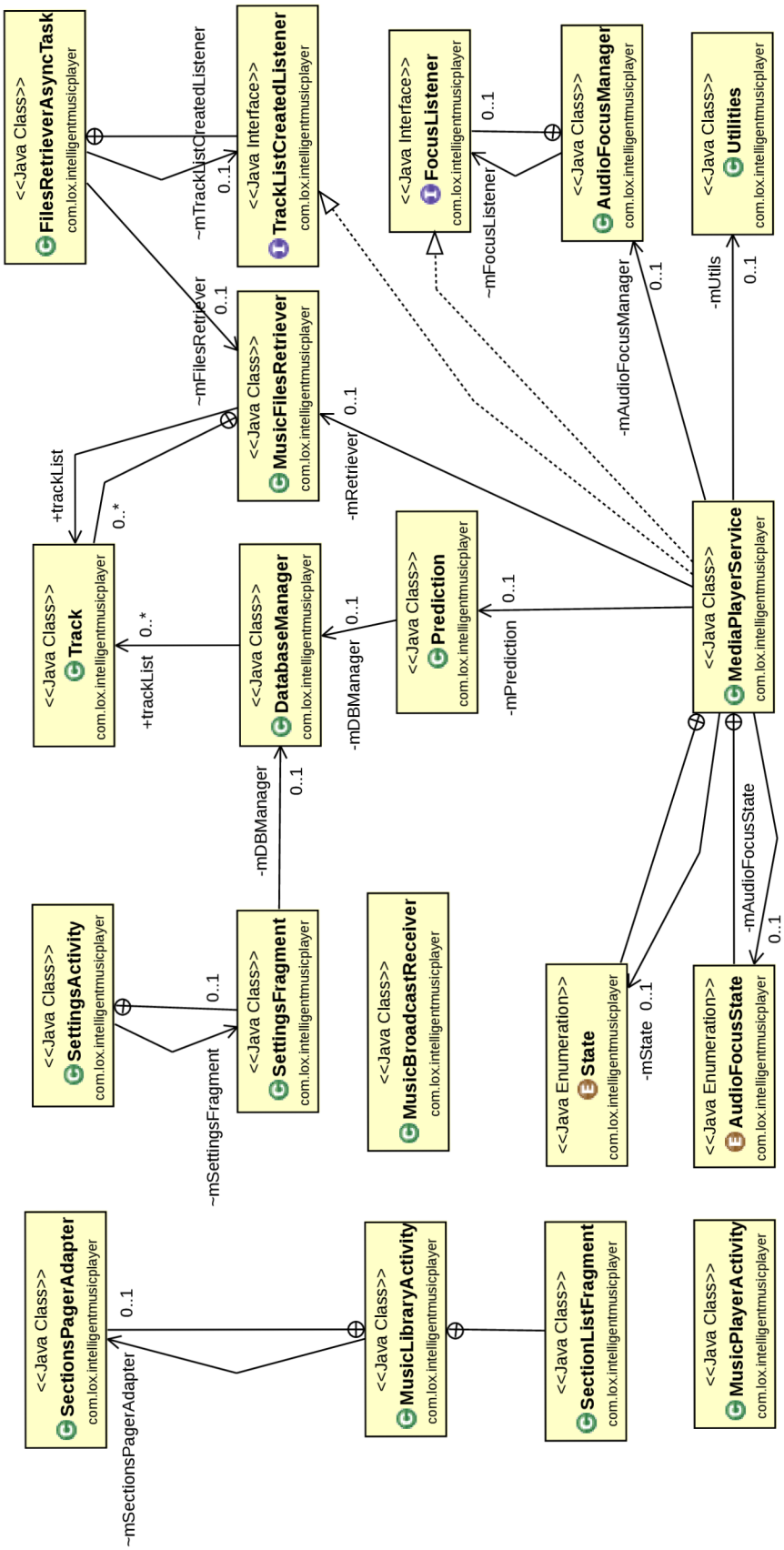
- `DatabaseManager` – Zajišťuje připojení a komunikaci s databází. Rozšiřuje `SQLiteOpenHelper`, čímž umožňuje jednoduché získání přístupu k databázi aplikace, jak v režimu čtení, tak i zápisu. Také zajišťuje synchronizaci databáze s aktuálním seznamem skladeb, které jsou v zařízení přítomné. Synchronizace databáze s aktuálním stavem je důležitá především proto, že je přímo z databáze získáván prostor hodnot, který používá při své práci selektivní funkce.
- `Utilities` – Implementace pomocných funkcí pro převody aktuálního/celkového času skladby, a jiné funkce používané především pro práci s posuvníkem na hlavní obrazovce.
- `MusicPlayerActivity` – Hlavní obrazovka přehrávače. Komunikace služby s hlavní obrazovkou přehrávače probíhá pomocí lokálních broadcast zpráv, které lze zachytit pouze komponentami této aplikace. Zprávy jsou zasílány asynchronně.
- `MusicLibraryActivity` – Obrazovka zobrazující knihovnu skladeb.
- `SettingsActivity` – Obrazovka zobrazující nastavení.

Na obrázku 8.3 je vidět diagram tříd výsledné aplikace.

### Uživatelské rozhraní

Uživatelské rozhraní implementované v jednotlivých obrazovkách aplikace (viz obrázek 8.1) bylo vytvořeno pomocí grafického designéru, který produkuje XML kód reprezentující kompletní rozložení všech prvků na zobrazovací ploše. Tento designér je součástí vývojového balíku SDK. Vytvářet rozložení i přidávat jednotlivé prvky lze provádět i ručně, psaním XML kódu, ale pro méně zkušené vývojáře je práce s designérem snazší.

Obrazovka zobrazující knihovnu skladeb (viz obrázek 8.1) pak navíc rozšiřuje tzv. `FragmentActivity`, což jí umožňuje zobrazovat rozdílné informace pro jednotlivé oddíly, na které je rozdělena. Lze tedy v této aktivitě zobrazit skladby uspořádané jak podle názvu skladby, tak podle názvu alba či jména umělce. Oddíl, který je právě zobrazený, lze rozlišit dle popisku umístěném v modrém pruhu v horní části obrazovky.




Obrázek 8.3: Diagram tříd finální aplikace.

# Kapitola 9

# Testování aplikace

Tato kapitola se bude soustředit na testování implementované aplikace především z pohledu funkčnosti či nefunkčnosti predikce. Ověření správné funkce predikce bude probíhat zejména porovnáním s módem *shuffle*, který provádí čistě náhodný výběr následující skladby. Predikce bude zhodnocena i subjektivně díky uživatelům aplikace, kteří se zúčastnili testování. Správnost implementace aplikace byla ověřována v průběhu implementace jednotlivých částí aplikace, ale v průběhu testování na různých zařízeních od odlišných výrobců a s jinými verzemi operačního systému bylo taktéž několik chyb odstraněno.

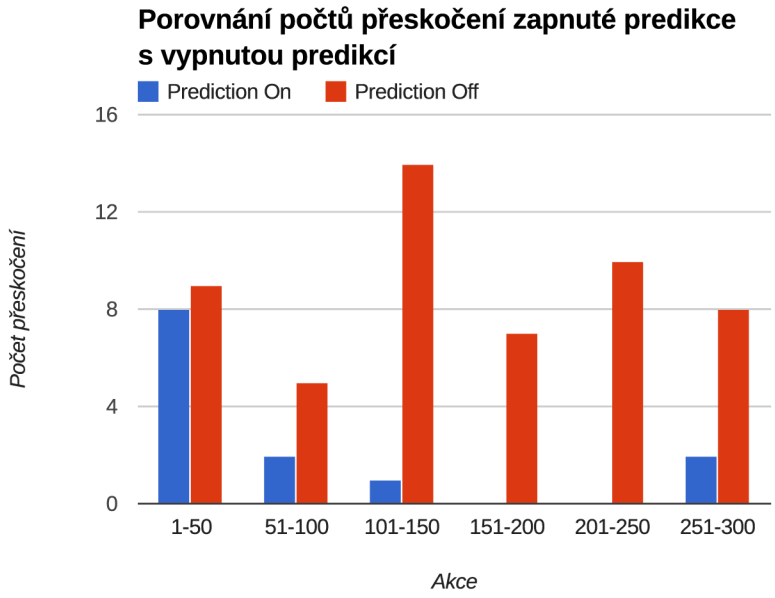
## 9.1 Porovnání predikce s módem shuffle

Mód *shuffle*, který je přítomen nejspíše ve všech dnešních hudebních přehrávačích, má za úkol provádět náhodný výběr skladeb při přehrávání. Predikce tedy musí poskytovat inteligentnější výběr skladeb, který zajistí nižší množství manuálních přeskočení skladby uživatelem. Při porovnání s módem *shuffle* je tedy vhodné se nejvíce zaměřit na množství těchto manuálních přeskočení skladby.

Porovnání těchto hodnot proběhlo po provedení tří set uživatelských činů, jak se zapnutou, tak i vypnutou predikcí. Při vypnuté predikci, měla aplikace stále zapnutý mód *shuffle*. V obou případech bylo množství skladeb rovno 50-ti z čehož právě 8 skladeb bylo neoblíbených. Provedení a zaznamenání těchto činů bylo uskutečněno v rámci běžného používání aplikace autorem této práce. Jednotlivé hodnoty počtů přeskočení lze vidět v grafu 9.1.

Porovnání bylo provedeno pro každých 50 činů zvlášť především z důvodu aplikovaného vylepšení prediktivního algoritmu (viz druhé z popsaných vylepšení v sekci 7.3.1), které má za následek, že v prvních 50-ti výběrech bude postupně přehráno všech 50 skladeb. A právě proto je počet přeskočených skladeb v prvních 50-ti činech uživatele rovno osmi. Tento počet totiž odpovídá množství neoblíbených skladeb.

Po prvních 50-ti činech uživatele lze pozorovat, že při zapnuté predikci postupně klesá množství manuálních přeskočení skladeb, a tím tedy i počet výběrů neoblíbených skladeb. Na rozdíl od módu *shuffle*, jehož průměrný počet přeskočení na 50 činů dosahuje hodnoty $8,83$. Tato hodnota mírně převyšuje očekávanou hodnotu (8), která je i tak příliš vysoká. Očekávaná hodnota by přesně odpovídala pro rovnoměrné pravděpodobnostní rozložení. V případě zapnuté predikce je průměr přeskočení na 50 činů roven $2,16$. Množství manuálních přeskočení, které uživatel v módu *shuffle* vykoná, je tedy čtyřikrát vyšší. A v případě, kdy nebude započítáno prvních 50 činů u obou variant, kdy dochází při zapnuté predikci k postupnému přehrání všech skladeb, bude rozdíl téměř devítinásobný. V průměru by tak




Obrázek 9.1: Graf porovnávající počty manuálních přeskočení při zapnuté a vypnuté predikci.

totiž připadalo při zapnuté predikci na 50 činů jedno manuální přeskočení, v porovnání s 8, 8 manuálních přeskočení na 50 činů s vypnutou predikcí.

Při průměrné délce skladby asi 4 minuty by tedy zhruba po třech hodinách používání přehrávače mělo dojít k výraznému zlepšení výsledků v oblasti výběru skladeb vůči módu *shuffle*. Při tomto počátečním používání by přitom uživatel neprováděl oproti běžnému používání přehrávače žádné aktivity navíc. Je však důležité, aby při prvních ohodnoceních prováděl činy, které budou odpovídat tomu, zda má nebo nemá skladbu v oblibě z dlouhodobého hlediska.

## 9.2 Uživatelské testování

Pomocí uživatelského testování bylo možné zjistit, zda implementovaný algoritmus měl požadovaný pozitivní vliv na uživatelský prožitek. Ke zlepšení mělo dojít především v potřebě manuálního přeskakování skladeb, přičemž tento přínos byl v této fázi testování subjektivně zhodnocen uživateli.

Uživatelé, kteří se zúčastnili testování, byli s funkcí predikce důkladně seznámeni a po několika týdnech používání byli požádáni o vyplnění dotazníku. Tento dotazník byl vytvořen tak, aby mohli testující uživatelé snadno sdělit svůj subjektivní postoj vůči implementované aplikaci. Otázky byly zaměřeny především na funkcionalitu predikce, nikoliv na běžné vlastnosti hudebních přehrávačů.

Dotazník obsahoval tyto otázky:

1. Měl(a) jste problémy v oblasti rychlosti či odezvy přehrávače?
2. Pocítil(a) jste změnu oproti běžnému módu *shuffle*?

3. Byla tato změna pozitivní?

4. Myslíte si, že se snížil počet manipulací s aplikací přehrávače?

5. Používal(a) by jste přehrávač i nadále?

První otázka se zaměřuje na implementaci výpočetně náročnějších operací do vláken na pozadí. Tedy, zda grafické uživatelské rozhraní reagovalo dostatečně rychle.

Druhá otázka se již týká predikce, zda uživatel poznal rozdíl při používaní přehrávače se zapnutou predikcí. Stav, kdy uživatel nepocítil žádný rozdíl, by mohl být způsoben velkým množstvím skladeb, které by zapříčinilo dlouhou dobou učení.

Otázka číslo tři zjišťuje jakého charakteru byl rozdíl, pokud uživatel nějaký zaznamenal. Jinými slovy je zde uživatel dotazován, zda mu predikce poskytovaná implementovaným algoritmem vyhovovala, či nikoliv.

Čtvrtá otázka vyžaduje, aby se uživatel zamyslel nad tím, zda byl splněn hlavní úkol prediktivního algoritmu. Konkrétně snížení počtu manuálních zásahů do přehrávání přehrávače.

Poslední otázka má za úkol ověřit, zda by o aplikaci přehrávače s predikcí byl mezi uživateli zájem, přestože většina z nich již svůj oblíbený přehrávač používá a je na něj zvyklá.

Výsledky dotazníku jsou zobrazeny v tabulce 9.1.

| **Otázka číslo** | **Počet odpovědí** | | | |
|---|---|---|---|---|
| | **Ano** | **Spíše ano** | **Spíše ne** | **Ne** |
| **1** | 0 | 0 | 0 | 4 |
| **2** | 4 | – | – | 0 |
| **3** | 2 | 2 | 0 | 0 |
| **4** | 4 | – | – | 0 |
| **5** | 4 | – | – | 0 |

Tabulka 9.1: Výsledky dotazníku.

Díky uživatelskému testování bylo také možné implementovanou aplikaci vyzkoušet na různých zařízeních s různou verzí operačního systému. Jednotlivá zařízení, na kterých proběhlo testování, a jejich verze operačního systému jsou uvedeny v tabulce 9.2.

| **Zařízení** | **Verze OS v době testování** |
|---|---|
| LG Nexus 4 | Android 4.4.2 |
| Sony Xperia Z | Android 4.2.2, později 4.3.0 |
| Samsung Galaxy S III | Android 4.1.2 |
| HTC Desire X | Android 4.1.1 |

Tabulka 9.2: Zařízení, na kterých probíhalo testování.

Ze zmíněné tabulky lze zjistit, že všechna zařízení, na kterých probíhalo testování, se liší verzí operačního systému. Rozdíly mezi některými z verzí měli za následek několik chyb, které byly následně opraveny.

Kromě posledního ze zmíněných platí, že zařízení disponují dostatečným výkonem, jelikož používají čtyřjádrový procesor a jejich operační paměť je větší nebo rovna 1024 MB.

Tyto specifika zajišťují výkon, který by měl být i při náročnějších operacích dostatečný. Poslední zařízení ale disponuje „pouze“ dvoujádrovým procesorem a operační pamětí o velikosti 768 MB.

Kvůli těmto slabším komponentám bylo tedy důležité zjistit, jak rychlá bude odezva aplikace v tomto zařízení. U zmíněného zařízení nebyla odezva měřena exaktně, jelikož autor této práce není jeho vlastníkem. Vlastník tohoto zařízení ale v dotazníku uvedl, že neměl žádné problémy s rychlostí ani s odezvou aplikace. Lze tedy předpokládat, že i na takto výkonných zařízeních bude aplikace dostatečně rychlá.

Na zařízení *LG Nexus 4* byla odezva měřena přesně pomocí IDE Eclipse, které bylo při vývoji použito. Doba přechodu na následující skladbu, při kterém se provádí aktualizace ohodnocení původní skladby a výběr selektivním algoritmem, je přibližně 60 ms pro 442 skladeb uložených v zařízení. Délka provádění selekce je závislá na množství skladeb v zařízení. I pro takto vysoké množství skladeb je však zpoždění 60 ms velmi těžko postřehnutelné.

Odezva a tedy i rychlost aplikace byla na všech testovaných zařízeních uspokojivá. Tento fakt potvrdily i výsledky dotazníku. Dalo by se tedy říci, že výkon většiny dnešních zařízení bude pro implementovaný přehrávač dostatečný. V případě potřeby by mohla proběhnout optimalizace algoritmu. Především části algoritmu zajišťující selekci tak, aby byl zbaven závislosti na počtu skladeb v zařízení.

# Kapitola 10

# Závěr

Cílem této práce bylo navrhnout a implementovat hudební přehrávač pro operační systém Android. Pro splnění podmínek musel tento přehrávač umět rozpoznat oblíbené/neoblíbené skladby uživatele a upravit podle získaných informací své chování v oblasti výběru skladeb pro přehrání. Tato aplikace byla implementována a dokáže již po prvním činu uživatele značně pozměnit pravděpodobnost výběru dané skladby. A to jak pozitivně, tak i negativně. Stěžejním bodem aplikace je především zvýšení komfortu používání přehrávače tím, že jeho uživatel nemusí tak často zapínat obrazovku zařízení, kvůli manuálnímu přeskočení neoblíbených skladeb. Po delší době používání by pak mohlo dojít k úplnému vymizení potřeby manuálně přeskakovat skladby.

Tato práce byla zejména zaměřena na algoritmus predikce. Ten byl vytvořen jako vhodná kombinace dvou částí, které musí být pro možnost učení přítomny. Jedná se o část pro ohodnocování stavů a o část pro výběr následujícího stavu. Pro obě části bylo více možností výběru algoritmů, ale díky provedeným experimentům bylo nalezeno současné funkční řešení. Toto řešení je ve výsledné mobilní aplikaci implementované a otestované.

Další vývoj by se měl ubírat dvěma směry. Jednak ve směru vylepšování prediktivní funkce, především by mohla predikce nabídnout různé režimy práce, kdy ne všechny by byly takto dlouhodobé. Takové rozšíření by umožnilo uživateli možnost výběru chování predikce, které by mu vyhovovalo nejvíce. Vývoj by však měl pokračovat i ve směru vylepšování aplikace přehrávače. Nejlépe dokončením knihovny skladeb, umožněním importu ohodnocení skladeb či implementací ekvalizéru. Právě možnost importovat ohodnocení skladeb by byla velmi vhodná, ale z důvodu nejednoznačnosti primárního klíče používané databázové tabulky vůči jednotlivým souborům v zařízení není implementace této funkcionality triviální. Druhou možností by byla implementace prediktivního algoritmu do jiného, již existujícího, hudebního přehrávače. Při této alternativě by mohlo být vloženo více úsilí do zlepšení predikce, za použití vyzkoušeného a odladěného přehrávače.

# Literatura


[1] Android Developers: Android Introduction [online]. [cit. 2014-01-27].
URL http://developer.android.com/tools/workflow/index.html

[2] Android Developers: Introduction to Android [online]. [cit. 2014-01-27].
URL http://developer.android.com/guide/index.html

[3] Android Developers: App Manifest [online]. [cit. 2014-01-28].
URL
http://developer.android.com/guide/topics/manifest/manifest-intro.html

[4] Android Developers: Resources Overview [online]. [cit. 2014-01-28].
URL http://developer.android.com/guide/topics/resources/overview.html

[5] Android Developers: Notifications - Handling compatibility [online]. [cit. 2014-03-23].
URL http://developer.android.com/guide/topics/ui/notifiers/
notifications.htm#Compatibility

[6] Android Developers: Media Playback [online]. [cit. 2014-03-24].
URL http://developer.android.com/guide/topics/media/mediaplayer.html

[7] Android Developers: MediaPlayer [online]. [cit. 2014-03-24].
URL
http://developer.android.com/reference/android/media/MediaPlayer.html

[8] Android Apps on Google Play: Google Play Music [online]. [cit. 2014-04-05].
URL
https://play.google.com/store/apps/details?id=com.google.android.music

[9] Android Apps on Google Play: PlayerPro Music Player Trial [online]. [cit. 2014-04-05].
URL
https://play.google.com/store/apps/details?id=com.tbig.playerprotrial

[10] Android Apps on Google Play: Poweramp Music Player (Trial) [online]. [cit. 2014-04-05].
URL
https://play.google.com/store/apps/details?id=com.maxmpz.audioplayer

[11] Ayodele, T. O.: *New Advances in Machine Learning*. InTech, 2010, ISBN 978-953-307-034-6.

[12] Giraud-Carrier, C.: A Note on the Utility of Incremental Learning. 2004-01-23 [cit. 2014-04-19].
URL http://www.cs.bris.ac.uk/Publications/Papers/1000535.pdf

[13] Goldberg, D. E.: *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., první vydání, 1989, ISBN 0201157675.

[14] Haag, M.: Stationary and Nonstationary Random Processes. In: OpenStax_CNX [online]. 2005-07-18 [cit. 2014-04-19].
URL http://cnx.org/content/m10684/2.2/

[15] Jain, A.: Introduction to Android development Using Eclipse and Android widgets [online]. 2010-11-16 [cit. 2014-01-28].
URL http://www.ibm.com/developerworks/opensource/tutorials/os-eclipse-androidwidget/index.html

[16] Koenigstein, N.; Dror, G.; Koren, Y.: Yahoo! Music Recommendations: Modeling Music Ratings with Temporal Dynamics and Item Taxonomy. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, New York, NY, USA: ACM, 2011, ISBN 978-1-4503-0683-6, s. 165–172, doi:10.1145/2043932.2043964.
URL http://doi.acm.org/10.1145/2043932.2043964

[17] Litwin, L.: Method for media popularity determination by a media playback device. Prosinec 25 2003, uS Patent App. 10/176,598.
URL http://www.google.cz/patents/US20030236695

[18] Manning, C. D.; Raghavan, P.; Schutze, H.: *Introduction to information retrieval*. New York: Cambridge University Press, 2008, ISBN 05-218-6571-9.
URL http://www-nlp.stanford.edu/IR-book/

[19] Martina, L.: Waldův intervalový odhad parametru binomického rozdělení a jeho alternativy. *Informační bulletin České Statistické Společnosti*, ročník 3, 2012: s. 1–22, ISSN 1210-8022, doi:10.5300/IB.
URL http://www.statspol.cz/bulletiny/ib-2012-3-web.pdf

[20] Sutton, R. S.; Barto, A. G.: *Reinforcement learning*. MIT Press, 1998, ISBN 0-262-19398-1.

[21] Walker, A. J.: An Efficient Method for Generating Discrete Random Variables with General Distributions. *ACM Trans. Math. Softw.*, ročník 3, č. 3, Září 1977: s. 253–256, ISSN 0098-3500, doi:10.1145/355744.355749.
URL http://doi.acm.org/10.1145/355744.355749

# Příloha A

# Obsah CD

- `apk\*` – instalační soubor pro platformu Android
- `doc\*` – programová dokumentace (Javadoc)
- `src\*` – zdrojové kódy aplikace
- `thesis\*` – technická zpráva ve formátu pdf
- `thesis-latex\*` – zdrojové kódy technické zprávy ve formátu LaTeX
- `readme.txt` – návod na instalaci aplikace na zařízení s OS Android